Projet Fil Rouge

Exploration et application des méthodes d'explicabilité pour l'amélioration des modèles de Machine Learning

---

CentraleSupélec

EXED

par:
Mathilde Boucly
Michel Daher Mansour

22/06/2025

Contents

Résumé

Cette étude s'inscrit dans le cadre du **projet fil rouge** mené dans le *Master Spécialisé IA de Confiance*, et répond à l'enjeu croissant d'un usage responsable du *Machine Learning* dans des domaines sensibles. Dans ce contexte, la capacité à interpréter et à comprendre les décisions des modèles devient aussi essentielle que leur performance prédictive.

Nous nous appuyons sur deux cas d'usage concrets — la cybersécurité et la santé — pour développer des modèles prédictifs adaptés à chaque domaine. Nous analysons ensuite en profondeur leurs mécanismes décisionnels à l'aide de différentes méthodes d'explicabilité, à la fois globales et locales.

Dans un premier temps, nous avons entraîné des modèles tels que les forêts aléatoires (*Random Forests*) et les arbres de boosting (*XGB*) pour des tâches de prédiction. Dans un second temps, afin de comprendre les décisions prises par ces modèles, nous avons mobilisé plusieurs techniques d'explicabilité. Parmi les méthodes globales, nous utilisons les *Partial Dependence Plot* (PDP), *Accumulated Local Effects* (ALE) et *Leave-One-Feature-Out* (LOFO). Du côté des méthodes locales, nous avons recours à *Local Interpretable Model-agnostic Explanations* (LIME), *Individual Conditional Expectation* (ICE) et *Anchors*. Enfin, la méthode *SHAP* (SHapley Additive exPlanations) est explorée pour sa capacité à combiner les dimensions globale et locale de l'explicabilité.

Chaque approche est évaluée selon sa pertinence, ses limites, et son adéquation aux contextes spécifiques des jeux de données utilisés : le *Parkinson Telemonitoring Dataset*, issu du domaine de la santé, et un jeu de données de cybersécurité dédié à la détection d'activités malveillantes.

Au-delà de la simple application de ces outils, ce travail propose une analyse critique de leur valeur ajoutée dans des environnements décisionnels réels. Il souligne ainsi l'importance d'une interprétabilité rigoureuse pour un déploiement responsable et éthique des systèmes d'intelligence artificielle.

1 Introduction

L'intelligence Artificielle, et plus particulièrement de le Machine Learning, a profondément transformé la manière dont les données sont exploitées dans divers domaines. Cependant, au-delà de la performance des modèles prédictifs, la transparence et la compréhension de leurs décisions deviennent des enjeux cruciaux, notamment dans des contextes sensibles ou à fort impact décisionnel.

Dans ce projet, nous nous intéressons à l'interprétation de modèles appliqués à deux jeux de données issus de domaines distincts. Ces domaines, la cybersécurité et la santé, représentent des secteurs dans lesquels le Machine Learning joue un rôle de plus en plus central. L'objectif principal de notre travail est double : d'une part, développer un modèle prédictif performant pour chaque dataset, et d'autre part, analyser et interpréter les décisions du modèle à l'aide de différentes techniques d'explicabilité, tout en apportant un retour critique sur notre expérience avec chacune de ces méthodes.

Dans ce rapport, nous mettons l'accent sur les différentes méthodes d'explicabilité appliquées aux modèles de Machine Learning. Pour des informations plus détaillées sur les aspects techniques du projet — incluant le code source, les modèles développés, les fonctions de visualisation créées et les expérimentations réalisées — nous invitons le lecteur à consulter le dépôt GitHub suivant : *XAI_methods*.

La **section 2 - Revue sur l'explicailité** de ce rapport est consacrée à un état de l'art des principales méthodes d'explicabilité que nous avons utilisées. Parmi les approches globales, nous présentons les techniques *Partial Dependence Plot* (PDP), *Accumulated Local Effects* (ALE), et *Leave-One-Feature-Out* (LOFO). Pour l'explicabilité locale, nous étudions *Local Interpretable Model-agnostic Explanations* (LIME), *Individual Conditional Expectation* (ICE), et *Anchors*. Enfin, nous analysons la méthode *SHAP* (SHapley Additive exPlanations), qui permet d'aborder à la fois les aspects globaux et locaux de l'interprétation.

Dans la **section 3 - Méthodologie et Jeux de données**, nous décrivons la méthodologie adoptée. Nous présentons en détail les deux jeux de données utilisés : le *Parkinson Telemonitoring Dataset*, issu du domaine biomédical, et le *Cybersecurity Dataset*, utilisé pour la détection des comportements malveillants. Ces jeux de données permettent d'illustrer la diversité des cas d'usage et la nécessité d'adapter les méthodes d'explicabilité au contexte.

La **section 4 - Modèles et méthodes d'explicabilité** est dédiée à la présentation des résultats et à leur discussion. Pour chaque méthode d'explicabilité, nous illustrons ses apports, ses limites et sa pertinence en fonction du type de modèle et du contexte d'application. Nous analysons également les cas où certaines méthodes peuvent produire des interprétations difficiles à exploiter.

Enfin, dans la **section 5 - Conclusion**, nous concluons ce travail en proposant plusieurs pistes de réflexion sur l'intégration des méthodes explicatives dans des environnements industriels et notre recommandation pour chacune. Nous récapitulons également toutes ces méthodes dans des tableaux, qui permettent de les comparer, et de savoir quel librairie Python utiliser.

2 Revue sur l'explicabilité

L'explicabilité des modèles d'intelligence artificielle (XAI) constitue aujourd'hui un levier stratégique dans l'implémentation de solutions basées sur l'IA en environnement industriel. Elle répond à des exigences croissantes en matière de transparence, de traçabilité des décisions, de conformité réglementaire (notamment avec le RGPD et l'AI Act européen), mais aussi de compréhension métier. L'essor des modèles complexes ou non interprétables "boîtes noires" (Random Forest, XGBoost, réseaux de neurones) a entraîné le développement de nombreuses techniques XAI visant à rendre intelligibles leurs prédictions. Ces méthodes peuvent être classées selon deux axes principaux : l'échelle d'explication (globale ou locale) et leur agnosticisme au modèle.

Parmi les méthodes globales, les premières approches comme les Partial Dependence Plots (PDP), introduites par Friedman en 2001, permettaient de visualiser l'effet moyen d'une variable sur la prédiction.Friedman (2001) Les Accumulated Local Effects (ALE), proposées par Apley et Zhu en 2020, sont venues corriger les limites des PDP en tenant compte des corrélations entre variables.Apley and Zhu (2020) La méthode LOFO (Leave-One-Feature-Out), utilisée dès 2008 avec les forêts aléatoires par Breiman, quantifie l'importance d'une variable en mesurant la dégradation de performance lorsque celle-ci est exclue du modèle.Lei et al. (2018) Ces outils sont adaptés à une vision synthétique du modèle, utile pour des audits, des analyses de robustesse ou la présentation de résultats à des responsables techniques.

Parallèlement à ces méthodes d'explication a posteriori, certaines approches cherchent à rendre le modèle lui-même intrinsèquement interprétable. C'est le cas des *Explainable Boosting Machines* (EBM), développées par l'équipe de Microsoft Research. Les EBM appartiennent à la famille des *Generalized Additive Models* (GAM), mais utilisent un boosting de fonctions en escalier (histogram-based) pour capturer des relations non linéaires tout en conservant une structure transparente. Chaque prédiction est décomposable en contributions additives des variables, offrant une compréhension directe sans techniques d'explication externes. Ces modèles proposent un excellent compromis entre précision et interprétabilité, rivalisant parfois avec des modèles complexes comme les forêts aléatoires, tout en restant lisibles par un utilisateur métier. Ils se révèlent particulièrement adaptés aux contextes à enjeux réglementaires forts (santé, finance), où l'explicabilité native est requise.Caruana et al. (2015)

Les méthodes locales, comme LIME (Local Interpretable Model-agnostic Explanations), SHAP (SHapley Additive exPlanations), ICE (Individual Conditional Expectation) et Anchors visent à expliquer des prédictions individuelles. ICE, publiée en 2015 par Goldstein et al.,Goldstein et al. (2015), permet de visualiser l'effet d'une variable sur une prédiction pour une observation donnée. LIME, proposée par Ribeiro et al. en 2016, Ribeiro et al. (2016), approxime localement le modèle complexe par un modèle interprétable (linéaire ou à base de règles). Tandis que SHAP, développée par Lundberg & Lee en 2017, Lundberg and Lee (2017), s'appuie sur la théorie des jeux pour attribuer une contribution équitable à chaque variable. Enfin, Anchors, introduit par Ribeiro en 2018, Ribeiro et al. (2018), génère des règles simples de type if-then expliquant pourquoi une prédiction tient sous certaines conditions. Ces approches sont particulièrement utiles pour des cas d'usage métiers tels que la justification de décisions automatisées (ex. refus de crédit), la compréhension d'erreurs, ou encore la détection de biais. Ces approches locales sont particulièrement utiles dans des contextes métiers où il est essentiel de justifier une décision automatisée (ex : acceptation de prêt, rejet de candidature), analyser un comportement inattendu du modèle, ou identifier un biais.

Parmi ces méthodes, SHAP se distingue par sa double capacité à fournir des explications à la fois locales et globales. En effet, en s'appuyant sur les valeurs de Shapley issues de la théorie des jeux coopératifs, Štrumbelj and Kononenko (2014); Štrumbelj and Kononenko (2011), SHAP évalue l'impact marginal de chaque variable sur la prédiction en moyennant sur toutes les permutations possibles des variables. Cela permet d'obtenir une explication locale précise pour chaque prédiction, mais aussi une vue d'ensemble globale en agrégeant ces valeurs sur l'ensemble des observations.

Un atout clé dans le monde industriel est que la majorité de ces méthodes sont modèle-agnostiques, c'est-à-dire qu'elles fonctionnent indépendamment de l'algorithme utilisé. Cela permet de les intégrer dans des pipelines IA hétérogènes, sans avoir à adapter les explications à chaque nouveau modèle. L'implémentation concrète passe souvent par des outils intégrés (ex. SHAP dans XGBoost, LIME en Python) ou via des dashboards explicatifs accessibles aux métiers.

Pour terminer, une stratégie XAI efficace doit combiner des approches globales pour la gouvernance du modèle, et des approches locales pour l'aide à la décision au cas par cas. Elle doit aussi s'adapter aux différents profils d'utilisateurs, de l'ingénieur data au professionnel métier, en fournissant des explications à la fois techniquement valides, visuellement claires, et contextuellement pertinentes.

3 Méthodologie et Jeux de données

3.1 Méthodologie

Dans ce rapport, nous avons décidé de présenter une à une les différentes méthodologies existantes. Pour chacune d'entre elles, nous commencerons par expliquer leur fonctionnement, puis les résultats que nous avons pu obtenir sur nos deux jeux de données. La théorie mathématique derrière chacune de ces méthodes est disponible dans les annexes de ce document.

Nous comparerons les méthodes d'explicabilité sur plusieurs axes comme le temps de calcul, l'applicabilité des méthodes aux différents modèles ou encore la cohérence des explications retournées, vis à vis des autres méthodes.

Pour la cohérence des explications retournées, pour les méthodes d'explicabilité locales, nous utiliserons le même index sur les différentes méthodes, afin de pouvoir comparer les variables mises en avant pour la même donnée.

Au niveau de la répartition du travail, nous avons choisi de tous les deux travailler sur les deux datasets, en croisant les méthodes, afin que chacun puisse exprimer son avis sur chacune des méthodes.

Voici un tableau récapitulatif détaillant le travail de chacun sur les données :

Méthode	**Explication théorique**	**Dataset Cybersécurité**	**Dataset Parkinson**
Arbres de decision, EBM	Mathilde	Mathilde	Mathilde
SHAP	Mathilde & Michel	Michel	Mathilde
PDP	Michel	Michel	Mathilde
ALE	Michel	Michel	Mathilde
LOFO	Mathilde & Michel	Michel	Mathilde
LIME	Mathilde & Michel	Mathilde	Michel
ICE	Mathilde & Michel	Mathilde	Michel
Anchors	Mathilde	Mathilde	Michel

Table 1: Application des méthodes d'explicabilité aux deux jeux de données

3.2 Jeux de données

Dans le cadre de notre étude, nous avons retenu deux jeux de données distincts, chacun composé de données tabulaires.

Leur sélection s'est appuyée sur plusieurs critères :

- obtenir des performances de prédiction satisfaisantes (l'analyse de l'explicabilité d'un modèle peu performant nous semblait peu pertinente)
- couvrir des domaines d'application variés, afin de tester la robustesse des méthodes d'explication
- inclure un problème de régression et un autre de classification, pour évaluer les méthodes sur les deux grandes catégories de tâches supervisées.

3.2.1 Parkinson Telemonitoring

Le jeu de données *Parkinson Telemonitoring*, disponible sur le *UCI Machine Learning Repository* (ID: 189), a pour objectif de prédire l'évolution des symptômes de la maladie de Parkinson à partir de descripteurs extraits d'enregistrements vocaux. Il se concentre notamment sur la prédiction du score *UPDRS* (*Unified Parkinson's Disease Rating Scale*), une mesure clinique utilisée pour évaluer la progression des troubles moteurs.

Ce dataset a été introduit en 2009 dans le cadre d'une étude menée par Athanasios Tsanas, Max A. Little, Patrick E. McSharry et Lorraine O. Ramig, affiliés à l'Université d'Oxford et à l'Université du Colorado.

Caractéristiques principales

- **Nombre d'observations** : 5,875 enregistrements
- **Nombre de patients** : 42 patients suivis sur une période de 6 mois
- **Nombre de variables** : 26 attributs, parmi lesquels :
  - Identifiants : `subject#`, `test_time` (ces deux variables seront bien entendues enlevées du dataset pour l'entraînement du modèle)
  - Variables cibles : `motor_UPDRS`, `total_UPDRS`
  - Caractéristiques du patient : `âge`, `sex`
  - Attributs vocaux : `Jitter`, `Shimmer`, `NHR`, `HNR`, etc.
- **Type de tâche** : Régression (prédiction de scores continus)

Nous avons trouvé ce jeu de données bien adapté à notre étude sur l'explicabilité des modèles, car il permet d'identifier les caractéristiques vocales les plus influentes dans la prédiction des scores cliniques.

3.2.2 Cybersécurité

Ce second jeu de données est constitué de fichiers JSON, chacun représentant un graphe orienté (ou *Digraph*) correspondant à une séquence d'instructions assembleur x86. Chaque graphe est annoté par un ensemble de comportements à prédire, au total 453 étiquettes correspondant à des actions potentiellement malveillantes (`thinstall`, `clipboard`, `anti-debugging`, etc.). Le corpus complet contient 23,102 fichiers.

Le dataset a été proposé dans le cadre du Data Challenge de la Sorbonne 2025, organisé en collaboration avec le Commandement du ministère de l'Intérieur dans le cyberespace (COMCYBER-MI).

Deux types d'approches peuvent être envisagés pour exploiter ces graphes :

- **Approche structurelle (basée sur le graphe)** : extraction de métriques topologiques (nombre de nœuds, de cycles, transitions, etc.) exploitables dans des modèles de type `Graph Neural Network (GNN)`.

  java Copier Modifier

- **Approche lexicale (textuelle)** : traitement des instructions assembleur comme une séquence de mots, permettant l'usage de techniques classiques (`TF-IDF`, `n-grams`) ou de modèles de langage préentraînés comme `CodeBERT` ou `GraphCodeBERT`.

Dans le cadre du challenge, nous avons expérimenté les deux méthodes. L'approche lexicale enrichie s'est finalement révélée plus performante.

Pour chaque fichier JSON, les instructions assembleur (par ex. `CALL`, `JMP`) sont extraites ligne par ligne. Les nœuds du graphe représentent ces instructions, tandis que les arêtes indiquent les transitions entre elles. Nous avons extrait plusieurs types de caractéristiques :

- *Structurelles*: nombre de nœuds, d'arêtes, de boucles, etc.
- *Spécifiques*: fréquence de certaines instructions comme `JMP`, `JCC`, etc.
- *Globales* : taille du fichier, complexité du graphe.

Par ailleurs, nous avons enrichi les données en appliquant la méthode `TF-IDF` sur les mots-clés assembleur extraits des graphes. Après réduction de dimension basée sur leur fréquence, nous avons retenu un ensemble de 250 instructions/mnémoniques significatifs (cf. Annexe pour la méthode).

Au total, chaque observation (fichier JSON) est décrite par **290 variables**, combinant les caractéristiques structurelles et les indicateurs lexicaux pondérés.

Bien que le dataset ait initialement été conçu pour prédire l'ensemble des **453 comportements** annotés, nous avons, pour notre étude, restreint l'analyse à un seul comportement cible : ***bypass Windows File Protection***. Ce choix vise à concentrer l'évaluation sur l'explicabilité des modèles de classification, plutôt que sur la détection multi-label dans son ensemble.

4 Modèles et méthodes d'explicabilité

Dans cette section, nous présentons tout d'abord les modèles utilisés dans le cadre de cette étude, en fonction du cas d'usage considéré : cybersécurité ou maladie de Parkinson. Dans un second temps, nous détaillons les différentes méthodes d'explicabilité mises en œuvre, qu'elles soient globales ou locales.

4.1 Entraînement des modèles

Dans cette partie, nous présentons les modèles de Machine Learning entraînés pour chaque jeu de données. Bien qu'une phase de prétraitement des données ait été réalisée en amont, ce rapport se concentre principalement sur les modèles de prédiction et les techniques d'explicabilité associées. Pour chaque jeu de données, un modèle distinct a été développé afin de s'adapter aux spécificités des données.

Modèle du jeu de données Parkinson Telemonitoring

Pour le cas d'usage lié à la maladie de Parkinson, plusieurs modèles de **régression** ont été testés dans le but d'identifier celui offrant les meilleures performances prédictives. Nous avons évalué une régression linéaire, un modèle *Random Forest* et un modèle *XGBoost*. Afin de limiter le coût computationnel, le nombre d'arbres a été restreint à 100 pour les modèles basés sur des arbres de décision.

L'évaluation des performances a été réalisée à l'aide de deux métriques : la racine de l'erreur quadratique moyenne (RMSE) et le coefficient de détermination (R^2). Les résultats obtenus sont présentés dans le tableau 2.

Modèle	**RMSE**	R^2 **score**
Régression linéaire	9.70	0.15
Random Forest	2.96	0.92
XGBoost	3.35	0.90

Table 2: Performances des modèles appliqués au dataset Parkinson

Au vu de ces résultats, le modèle *Random Forest* a été retenu pour la suite de l'étude, car il présente les meilleures performances, notamment un score R^2 élevé, indiquant une capacité à bien ajuster les données.

Modèle sur le jeu de données de CyberSecurité

À partir des caractéristiques extraites des fichiers CFG, plusieurs modèles de **classification** ont été évalués, notamment la régression logistique, les réseaux de neurones multi-couches (*MLP*), les forêts aléatoires et les modèles de Boosting. Suite à une phase d'optimisation des hyperparamètres (incluant, par exemple, l'utilisation de *class_weight=balanced* pour compenser le déséquilibre des classes), le meilleur score F1 a été obtenu avec le modèle *HistGradientBoostingClassifier*.

Un modèle *HistGradientBoosting* a été entraîné pour chacun des comportements analysés, soit un total de 453 modèles.

Dans le cadre de cette étude, nous nous concentrons sur le comportement **bypass Windows File Protection**, soit un seul modèle sur les 453.

Après optimisation, le modèle sélectionné présente les hyperparamètres suivants :

```
HistGradientBoostingClassifier(
    class_weight='balanced',
    l2_regularization=0.5,
    learning_rate=0.25,
    max_depth=8,
    max_features=0.8,
    max_iter=400,
    max_leaf_nodes=53,
    min_samples_leaf=140,
    validation_fraction=None
)
```

Ce modèle a atteint un **F1-score de 0.91**, ce qui justifie son utilisation pour l'analyse d'explicabilité dans ce cas spécifique.

4.2 Modèles intrinsèquement explicables

4.2.1 Définition générale

Les **modèles intrinsèquement explicables** sont des modèles dont le fonctionnement peut être compris sans recourir à des techniques d'explication post-hoc. Leur structure ou leur forme permet d'identifier directement l'influence des variables sur la prédiction.

Ils s'opposent aux **modèles boîtes noires** (comme les forêts aléatoires, les réseaux de neurones ou les gradient boosting machines), dont les mécanismes internes sont plus complexes ou opaques.

Modèles principaux

- **Régression linéaire et logistique** :
  - Chaque variable a un **coefficient interprétable** (effet directionnel et proportionnel).
  - Permet une lecture directe de l'impact de chaque variable sur la sortie.
  - Hypothèse de linéarité et d'indépendance des variables.
- **Arbres de décision** :
  - Structure en **règles if-then-else**.
  - Chaque chemin du nœud racine à une feuille représente une **règle explicite de décision**.
  - La prédiction est donnée par la feuille atteinte.
  - Possibilité de visualisation complète de l'arbre (`plot_tree`)
- **EBM (Explainable Boosting Machine)** :
  - Variante des modèles additifs généralisés (GAM).
  - Prédit comme une **somme de fonctions apprises séparément pour chaque variable**.
  - Permet une **explication univariée et lisible** des effets non linéaires.
  - Très performant tout en restant lisible.

Avantages

- **Interprétabilité directe** : pas besoin de méthodes supplémentaires pour comprendre une prédiction.
- **Fidélité parfaite** : les règles ou coefficients décrivent exactement le comportement du modèle.
- **Lisibilité** : facilite la communication avec des experts métier ou non-spécialistes.

Limites

- **Moins performants** sur des problèmes complexes ou non linéaires.
- Sensibles au **bruit** et aux **corrélations** entre variables.
- Peu adaptés aux très grands ensembles de données sans régularisation ou simplification.

4.2.2 Notre utilisation

Avant d'appliquer les méthodes d'explicabilité connues, nous avons d'abord exploré l'interprétabilité directe offerte par les modèles intrinsèquement explicables. Les modèles de régression linéaire et logistique se sont révélés insuffisamment performants sur nos jeux de données, ce qui a limité leur intérêt pour l'analyse. En revanche, les arbres de décision obtenaient des résultats comparables à ceux de modèles plus complexes comme les forêts aléatoires ou les modèles de boosting. C'est pourquoi nous avons décidé de nous pencher sur ce qui influençait les décisions de ces arbres.

CyberSécurité Dataset

Les arbres de décision

Le modèle d'arbre de décision avait un **F1-score de 0.86**, contre des F1-scores de 0.91 pour notre meilleur modèle de boosting.

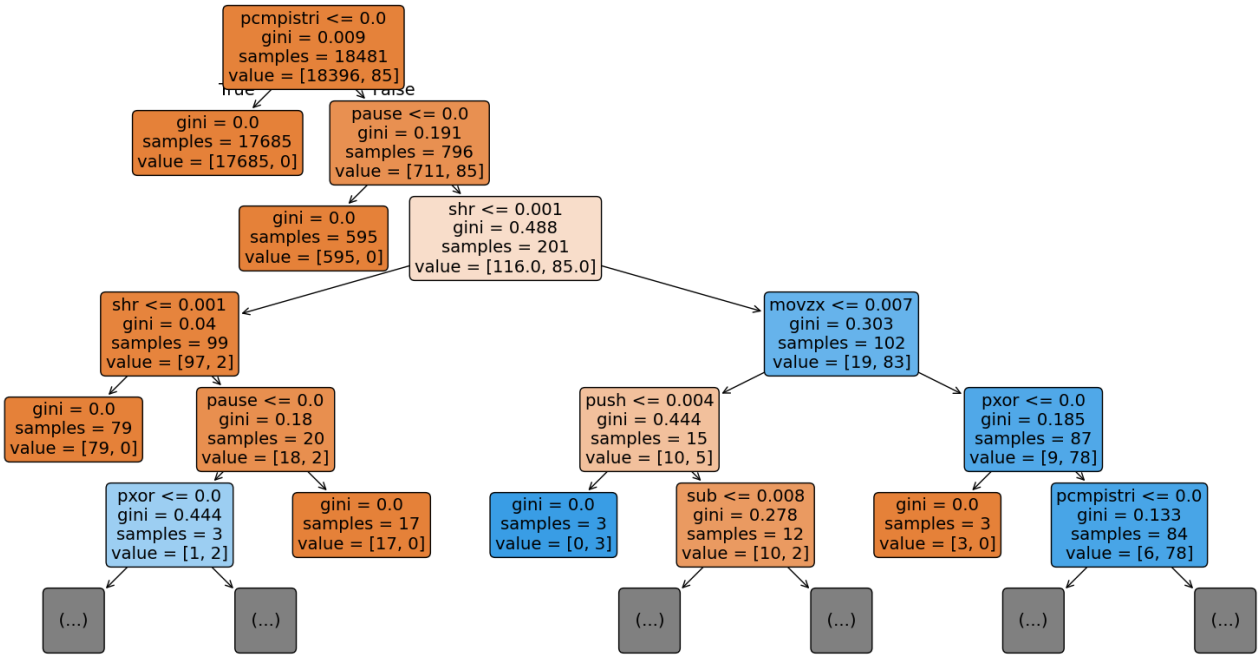

Figure 1: Décisions et critères de l'arbre de décision (profondeur à 5)

Un premier point très intéressant est le premier critère sur la variable `pcmpistri` (instruction utilisée pour comparer des chaînes de caractères de manière vectorielle) : si elle n'apparaît pas dans

le graphe d'exécution, le comportement est automatiquement absent. Ce critère permet d'éliminer directement 96% des fichiers de nos graphes d'exécution.

Nous pouvons aussi remarquer, que sur les 5 premiers niveaux, seules les occurences des mots clés TF-IDF servent au modèle, et souvent le simple fait de savoir si l'occurence est présente au moins une fois permet de discriminer une grande partie des fichiers.

Les EBM (Explainable Boosting Machine)

Avec la librairie Python `interpret`, il est assez simple d'entraîner un modèle de classification. Nous avons pour cela utilisé la fonction `ExplainableBoostingClassifier`. Son **F1-score est de 0.90**.

La librairie `interpret` offre une vue d'ensemble, avec une **explicabilité globale**, afin de repérer directement quelles features sont les plus influentes sur la décision du modèle :

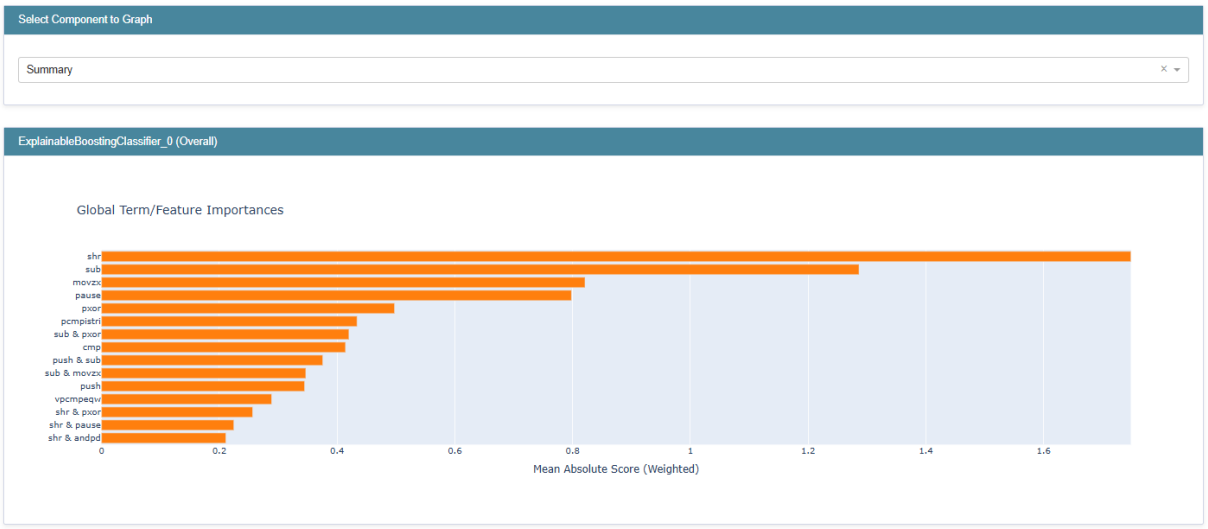

Figure 2: Sommaire de l'influence des variables sur le modèle EBM

Les features les plus importantes correspondent aux features qui étaient utilisées sur les 5 premiers niveaux de notre arbre de décision. Toutefois, dans le sommaire on ne retrouve pas les valeurs qui permettent de faire basculer le modèle, il faut pour cela regarder plus en détails l'influence de chacune des variables.

Voici un graphique, qui peut être vu pour chacune des features, indiquant l'influence de chacune des variables sur le modèle.

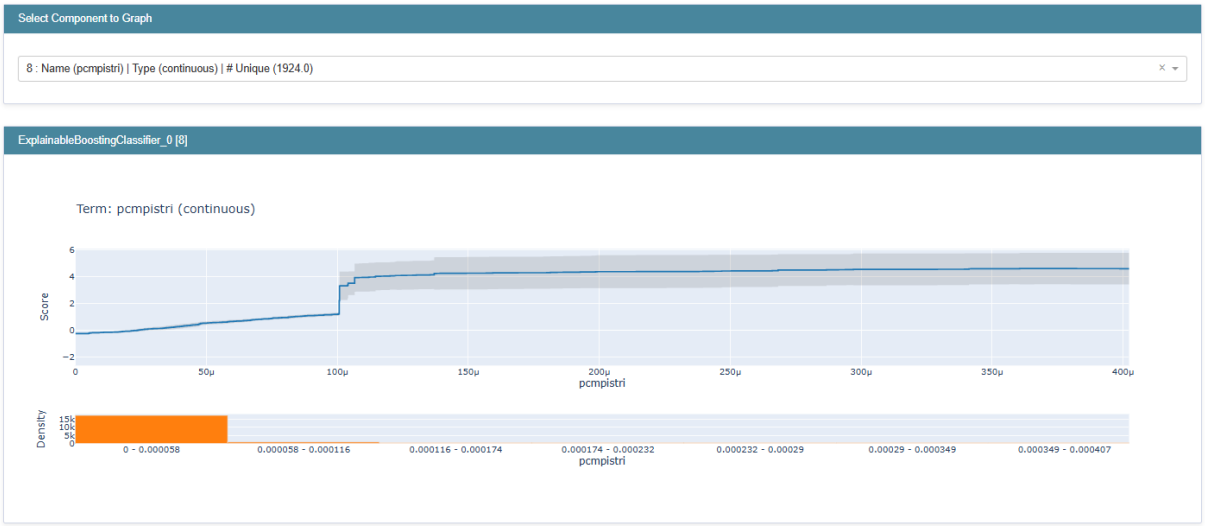

Figure 3: Influence de la variable `pcmpistri` sur le modèle EBM

Comme précédemment, on retrouve bien la décision du modèle qui augmente dès lors que la variable `pcmpistri` n'est plus à 0.

Il est également possible de faire une analyse **d'expliabilité locale**, instance par instance.

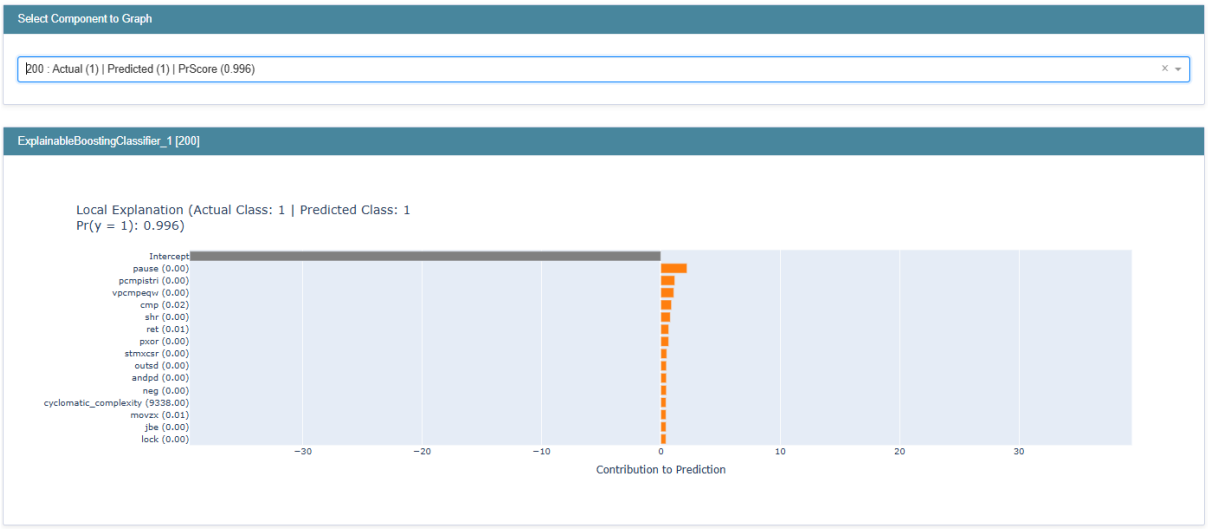

Figure 4: Explicabilité locale, de la ligne 200 sur le modèle EBM

Ici, nous avons sélectionné une variable qui s'avère positive : c'est à dire exprimer le comportement. On voit bien que toutes les features tendent à pousser vers ce comportement.

Les features les plus utilisées par le modèle sont bien similaire à celles trouvées pour l'explicabilité gloable. Toutefois, ils s'avère que ces fetaures ont des valeurs faibles mais non nulles. Malheureusement avec l'arrondi réalisé par la librairie `interpret`, sans faire attention il est possible de ne pas remarquer que ces features ne valent pas exactement 0.

Les données Parkinsons telemonitoring

Nous allons maintenant tester ces modèles sur un problème de régression, et des données entièrement différentes.

Les arbres de décision

Sur ces données, le modèle d'arbre de décision a un **R^2 de 0.83**, contre des R^2 de 0.92 et 0.90 pour respectivement la forêt aléatoire et le modèle de boosting.

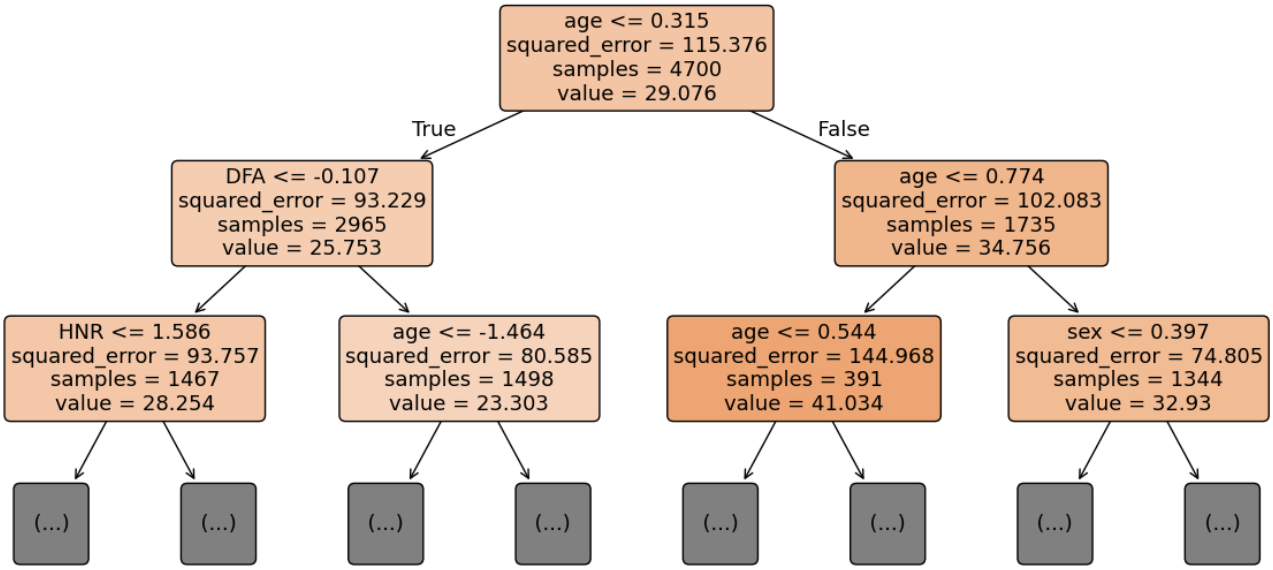

Figure 5: Décisions et critères de l'arbre de décision (profondeur à 2)

Ici nous pouvons voir que la variable `âge` joue un rôle principal : sur les 3 premiers niveaux elle est déjà présente 4 fois ! Nous voyons également que les mesures `DFA` et `HNR` apparaissent dès les premiers niveaux de l'arbre. Ce sont des caractéristiques extraites de la voix des patients : HNR représente la clarté vocale, et DFA si la voix est régulière.

Nous regarderons par la suite, si ces variables apparaissent à nouveau.

Les EBM (Explainable Boosting Machine)

Avec la librairie Python `interpret`, il est également possible d'entraîner un modèle de régression avec `ExplainableBoostingRegressor`. Son **R^2 est de 0.87**. Toutefois, ce modèle a pris plus de temps à l'entraînement que les précédents (2 minutes et 13 secondes, contre 3 secondes pour une forêt aléatoire et moins d'une seconde pour XGBoost)

Au niveau de **l'explicabilité globale**, ces features sont mises en avant :

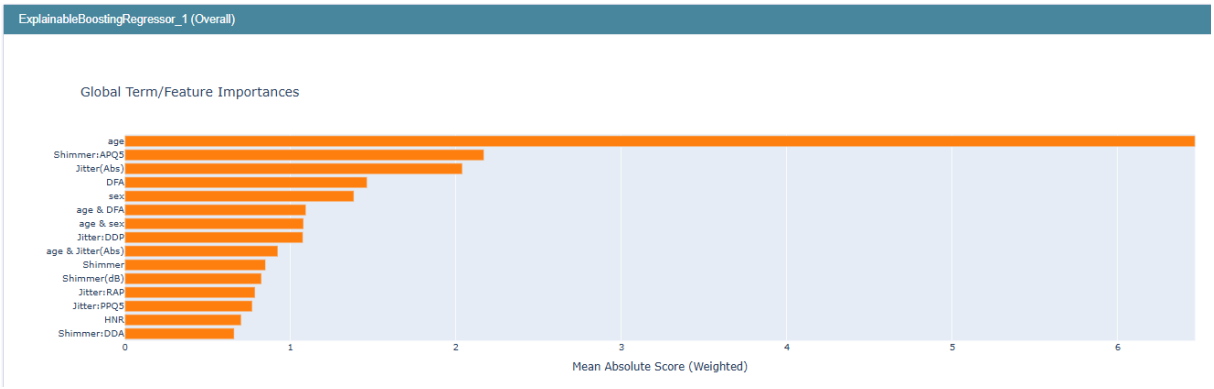

Figure 6: Sommaire de l'influence des variables sur le modèle EBM

De nouvelles features par rapport à celles mises en avant sur les premiers niveaux de l'arbre de décision apparaissent : c'est le cas de `Shimmer:APQ5` ou `Jitter(Abs)`.

On observe également que la variable `âge` conserve une influence majeure sur le modèle.

En regardant plus en détail la feature `âge`, on observe un comportement non linéaire. Certaines tranches d'âge rendent donc la présence de la maladie de Parkinson plus probable.

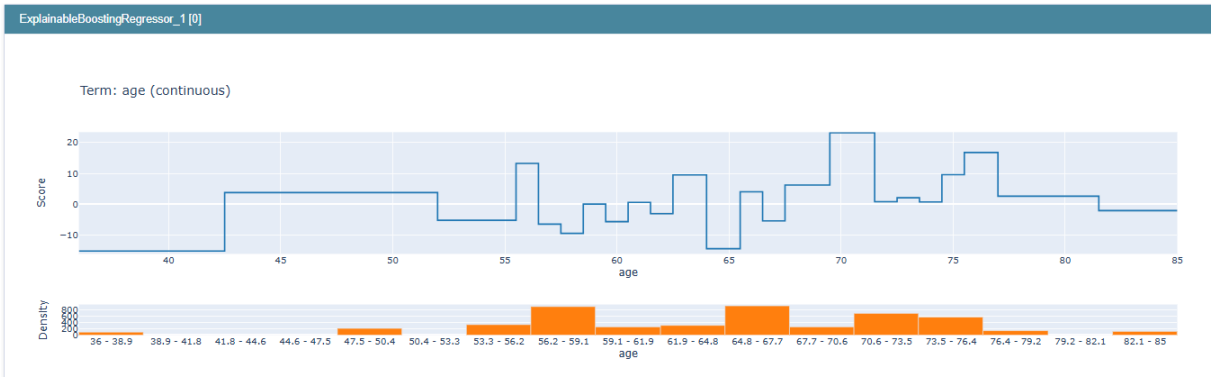

Figure 7: Influence de la variable `âge` sur le modèle EBM

Enfin, pour l'analyse **d'expliabilité locale**, instance par instance :

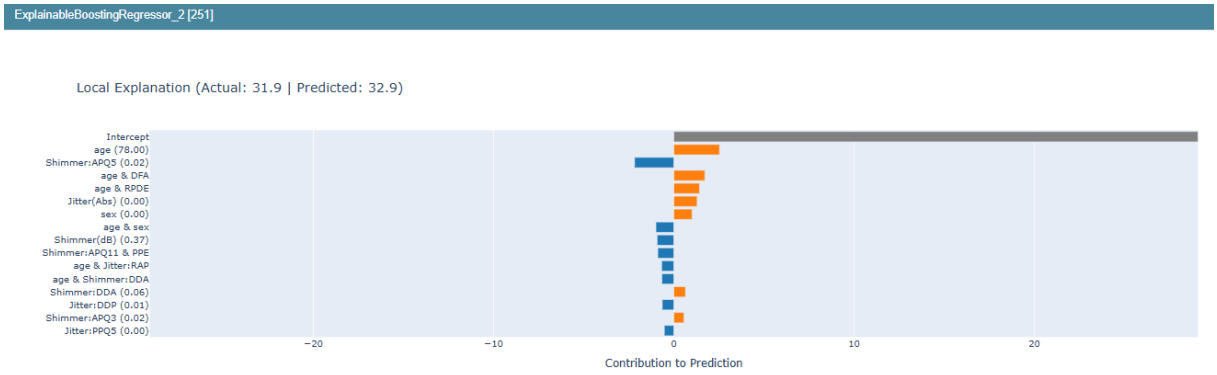

Figure 8: Explicabilité locale, de la ligne 9 sur le modèle EBM

C'et encore une fois principalement la variable âge qui joue sur la prédiction. On remarque également que les effets des différentes mesures réalisées sur la voix de ce patient ont tendance à se contrebalancer.

4.3 SHAP – méthode locale et globale

4.3.1 Théorie

La méthode **SHAP** (SHapley Additive exPlanations) est une approche d'interprétabilité des modèles d'apprentissage automatique basée sur la théorie des jeux coopératifs. Elle attribue à chaque caractéristique (feature) une valeur d'importance, appelée *valeur de Shapley*, qui reflète sa contribution à la prédiction du modèle.

Objectif

L'objectif principal de SHAP est d'**expliquer la prédiction d'un modèle pour une instance** x donnée en la décomposant comme une **somme de contributions individuelles de chaque caractéristique**.
SHAP calcule les valeurs de Shapley en considérant les valeurs des caractéristiques comme des joueurs dans une coalition, et la prédiction comme un gain à répartir équitablement entre eux.

Le modèle explicatif est défini comme suit :

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

où :

- g est le modèle explicatif,
- $z' \in \{0,1\}^M$ est le **vecteur de coalition** (indiquant la présence ou l'absence d'une caractéristique),
- M est le nombre total de caractéristiques,
- ϕ_j est la valeur de Shapley pour la caractéristique j.

Si toutes les caractéristiques sont présentes dans la coalition (z' est un vecteur de 1), l'équation devient :

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j$$

L'innovation majeure de cette approche est d'exprimer les valeurs de Shapley sous forme de **modèle additif linéaire**, ce qui les rend compatibles avec des méthodes locales d'explication.

Les valeurs de Shapley

Les **valeurs de Shapley** proviennent de la théorie des jeux coopératifs. L'idée est d'estimer la **contribution marginale moyenne** d'une caractéristique à toutes les coalitions possibles.

Formellement, la valeur de Shapley pour la caractéristique i est définie comme :

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \cdot \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]$$

où :

- N est l'ensemble des caractéristiques,
- S est un sous-ensemble de N ne contenant pas i,
- $f_S(x_S)$ est la prédiction du modèle en ne considérant que les caractéristiques dans S.

Avantages

Les valeurs de Shapley possèdent plusieurs **propriétés théoriques désirables** :

- **Efficience** : La somme des contributions est égale à la différence entre la prédiction et la valeur de base.
- **Symétrie** : Si deux caractéristiques contribuent de manière identique, elles obtiennent la même valeur.
- **Nullité** : Si une caractéristique n'affecte jamais la prédiction, sa valeur SHAP est nulle.
- **Additivité** : Pour deux modèles combinés, les valeurs SHAP se combinent également.

Limites

Le calcul exact des valeurs de Shapley est **combinatoire** : il nécessite d'évaluer toutes les coalitions possibles, ce qui engendre une complexité exponentielle $O(2^M)$. Cela rend le calcul intractable pour des modèles comportant de nombreuses variables. Pour pallier ce problème, plusieurs **approximations** ont été proposées.

Les différentes variantes de SHAP

Plusieurs méthodes ont été développées pour adapter SHAP à différents types de modèles :

- **KernelSHAP** :
  - Méthode générale, applicable à tout modèle (boîte noire).
  - Basée sur une régression localement pondérée.
  - Approximative mais très flexible.

- **TreeSHAP** :
  - Spécialisée pour les modèles d'arbres (comme XGBoost, LightGBM, CatBoost).
  - Permet un **calcul exact et rapide** des valeurs de Shapley.
  - Exploite la structure arborescente pour éviter l'énumération explicite des coalitions.
- **DeepSHAP** :
  - Conçu pour les réseaux de neurones.
  - Combine des idées de SHAP avec DeepLIFT pour approximer les valeurs de Shapley.
  - Nécessite certaines hypothèses sur l'architecture (comme la différentiabilité).
- **LinearSHAP** :
  - Méthode exacte pour les modèles linéaires.
  - Très rapide et peu coûteuse computationnellement.
- **PartitionExplainer** :
  - Variante pour les modèles d'arbres avec interactions complexes.
  - Utilise une stratégie de partitionnement récursif de l'arbre.
- **SamplingSHAP** :
  - Version échantillonnée de KernelSHAP pour les cas avec beaucoup de variables.
  - Approximative mais plus scalable.

4.3.2 Les librairies Python

- Le package principal est `shap`. C'est la librairie officielle développée par Scott Lundberg (auteur principal de SHAP). Elle permet de :
  - Calculer les valeurs SHAP avec toutes les méthodes mentionnées (kernel, tree, deep, linear).
  - Générer des visualisations :
    - **summary plot** (vue d'ensemble),
    - **dependence plot** (relation entre une variable et sa valeur SHAP),
    - **force plot** (visualisation locale),
    - **decision plot** (chemin de décision cumulatif).

  Exemple d'utilisation :

```
import shap
explainer = shap.Explainer(model)
shap_values = explainer(X_test)
shap.summary_plot(shap_values, X_test)
```

- Nous avons également découvert le package `shapash`, développé par la MAIF.
  - Il propose une surcouche conviviale à `shap` et d'autres outils XAI.
  - Il génère des visualisations interactives et compréhensibles par les utilisateurs métiers.
  - Il facilite la construction et le déploiement de dashboards explicatifs intégrant les variables, leurs contributions, et les prédictions.

Exemple d'utilisation :

```
from shapash.explainer.smart_explainer import SmartExplainer

xpl = SmartExplainer(model=clf)  # clf : modèle déjà entraîné
xpl.compile(x=X_test, y_pred=clf.predict(X_test), y_target=y_test)
xpl.plot.features_importance()
```

4.3.3 Notre utilisation

Nous avons pu tester SHAP sur nos deux jeux de données, avec des méthodes globales et locales.

En terme de rapidité, nous l'avons entraîné sur un modèle de `TreeExplainer`. Le temps dépend de la taille des arbres, et de la taille du dataset à expliquer. Pour donner un ordre de grandeur, il a fallu **25 secondes** pour un `RandomForest` avec 100 arbres, et un dataset à expliquer de 1175 lignes et 19 colonnes.

Néanmoins, si vous souhaitez l'utiliser avec des modèles à Kernel, la complexité est exponentielle!

CyberSécurité Dataset

Pour l'**analyse globale**, nous avons utilisé le **Summary Plot**. Voici le graphique que nous avons pu obtenir :

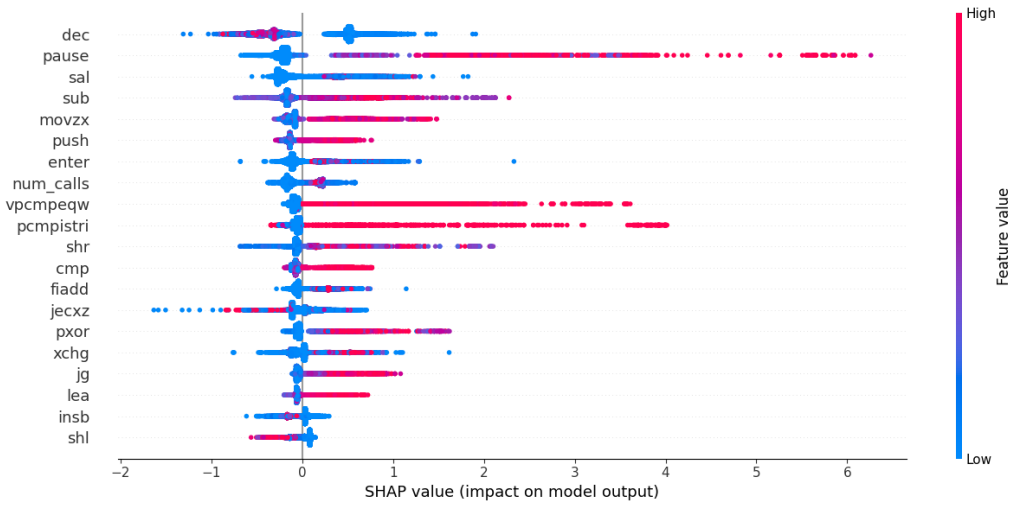

Figure 9: SHAP Summary Plot sur le dataset de cybersécurité

Le summary plot montre l'impact des différentes variables sur la prédiction. La figure met en évidence que l'effet de la variable `dec` sur la prédiction n'est pas linéaire. En revanche, la variable `pause` présente une relation plutôt linéaire avec la prédiction. Pour `sub`, les valeurs élevées ont un impact positif marqué, les valeurs moyennes un effet mitigé, tandis que les faibles valeurs ont un effet plus limité.

SHAP permettant également de réaliser des analyses d'explicabilité locales, voici le **force plot** obtenu : (au cours de notre étude, nous avons d'ailleurs créé nos fonctions de visualisation, permettant de créer une graphique dynamique, pour lequel l'utilisateur peut choisir l'index souhaité à l'aide du curseur en haut à gauche)

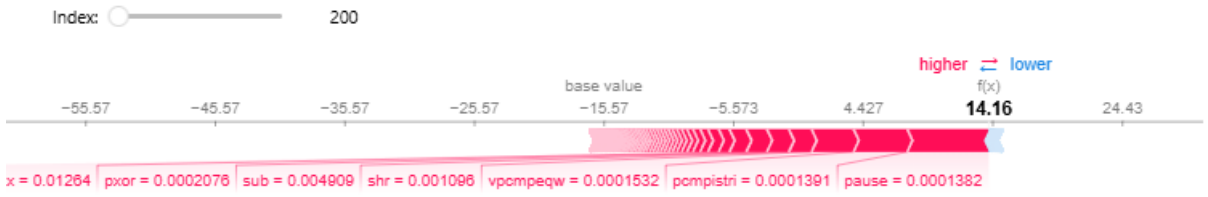

Figure 10: SHAP Summary Plot sur le dataset de cybersécurité

On peut voir que SHAP reprend très fidèlement les features mises en avant dans notre analyse sur les modèles intrinsèquement explicables.

Un aspect pratique pour nous a été la possibilité de définir le niveau de précision lors de la création du graphique, ce qui permettait d'afficher autant de chiffres décimaux que souhaité.

Nous avons également testé le rendu de la librairie `shapash`. Le calcul a été très rapide, et en quelques lignes de code nous avons pu avec dans notre navigateur les dahsboards suivants :

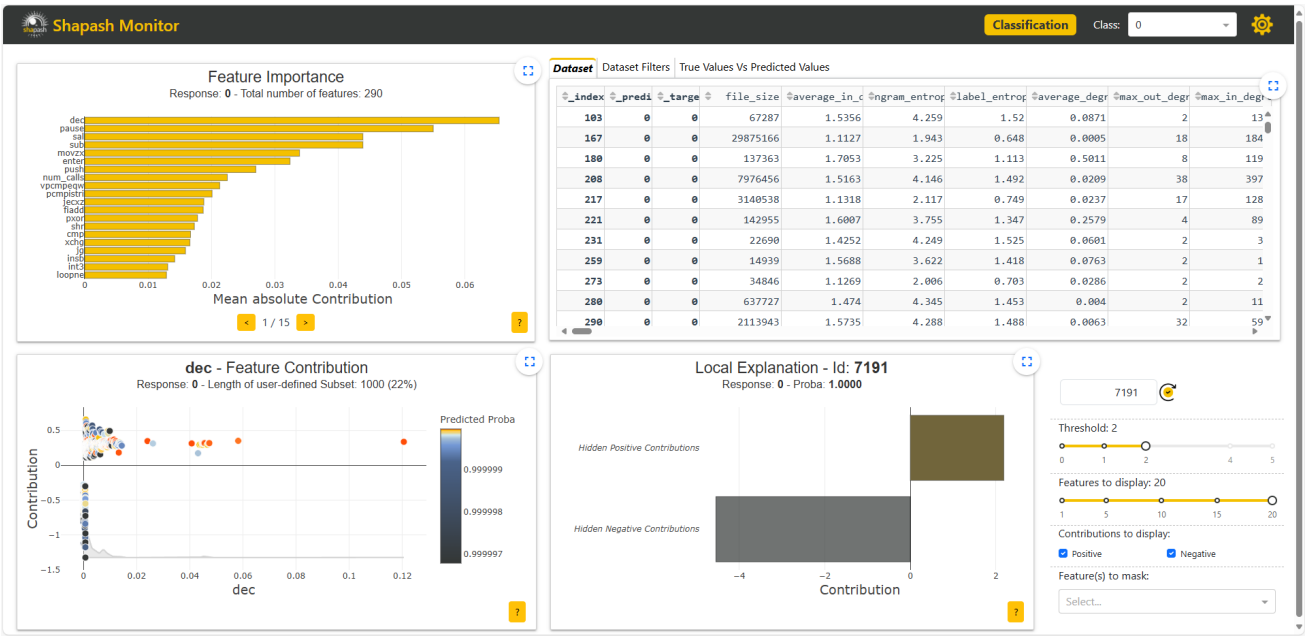

Figure 11: Shapash dashboards sur le dataset de cybersécurité

Dessus, nous y retrouvons à la fois l'explicabilité locale et globale de shap.

Les données Parkinsons telemonitoring

Pour l'**analyse globale**, voici le graphique que nous avons pu obtenir :

Figure 12: SHAP Summary Plot

Il démontre un fort impact de la variable `âge`, sur les autres variables. Nous pouvons également voir que le lien de la variable `âge` avec la variable à prédire n'est pas linéaire, contrairement à d'autres variables les plus importantes comme `DFA`.

Au niveau de l'analyse d'explicabilité **locale**, voici le force plot obtenu :

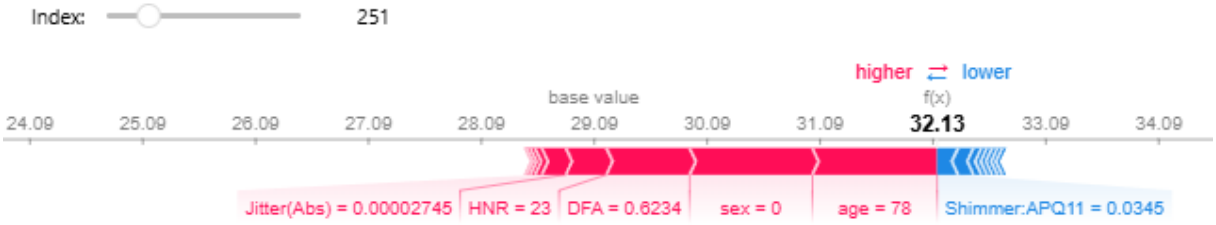

Figure 13: SHAP Force Plot

Le modèle prédit une valeur de 32.13, supérieure à la valeur de base (29.09). Cette augmentation est principalement due à `sex`, `age` et `DFA` qui ont poussé de (+0.14) la prédiction vers le haut.

Nous pourrons voir par la suite si les autres méthodes d'explicabilité ont des explications similaires.

Comme pour le dataset précédent, nous avons également testé la librairie `shapash` :

Figure 14: Shapash dashboards sur le dataset de Parkinson

De même, il a été très simple et très rapide à mettre en place. Pour la régression, un dahsboard permet également de voir les réelles valeurs en comparaison avec les valeurs prédites, en plus des méthodes d'explicabilité SHAP. Cette librairie permet également de tracer des graphiques de façon unitaire, et pas uniquement sous la forme de dashboards, mais nous avons trouvé cette fonction de dahsboard très sympathique !

4.4 Partial Dependence Plot (PDP) - méthode globale

Le **Partial Dependence Plot (PDP)** est une méthode d'interprétabilité **globale** permettant de visualiser la relation entre une ou plusieurs variables d'entrée et la sortie d'un modèle d'apprentissage automatique. Il répond à la question :

Quelle est l'influence moyenne d'une variable sur la prédiction du modèle ?

Le PDP estime **l'effet marginal moyen** d'une ou plusieurs caractéristiques $x_S \subset x$ sur la prédiction $f(x)$, en **marginalisant** les autres variables $x_C = x \setminus x_S$.

4.4.1 Définition formelle

La **fonction de dépendance partielle** est définie comme :

$$\hat{f}_S(x_S) = \mathbb{E}_{x_C}[\hat{f}(x_S, X_C)]$$

où :

- S est le sous-ensemble de caractéristiques d'intérêt,
- C est le complément de S,
- $\hat{f}_S(x_S)$ représente la prédiction moyenne lorsque x_S est fixé et x_C varie selon sa distribution dans les données.

Méthode de calcul (approximation empirique)

En pratique, cette espérance est approximée à partir du jeu de données $\{x^{(i)}\}_{i=1}^n$:

$$\hat{f}_S(x_S) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_S, x_C^{(i)})$$

Pour chaque valeur (ou grille de valeurs) de x_S, on crée des **instances artificielles** en combinant cette valeur avec tous les $x_C^{(i)}$ observés, puis on calcule la moyenne des prédictions du modèle.

Applications

Le PDP est principalement utilisé pour :

- **Comprendre les effets moyens** d'une variable sur la prédiction.
- **Visualiser la forme fonctionnelle** : linéaire, monotone, en U, etc.

Limites

Malgré son utilité, le PDP présente plusieurs **limitations importantes** :

- **Hypothèse d'indépendance** :
  Il suppose que les variables x_S et x_C sont **indépendantes**. En présence de **corrélations fortes**, les combinaisons générées peuvent être **non réalistes**, voire **inexistantes** dans les données.
- **Effets locaux masqués** :
  Le PDP **moyenne** les prédictions sur tout le jeu de données, ce qui peut **masquer des comportements locaux** ou des **interactions complexes**.
  Pour remédier à cela, on utilise les **ICE plots** (Individual Conditional Expectation), qui montrent les courbes individuelles pour chaque instance. ICE est donc complémentaire à PDP, et plus adaptée à l'analyse **locale**.

PDP et importance des variables

En 2018, Greenwell, Boehmke et McCarthy ont proposé une métrique appelée **PD-based Feature Importance**, qui consiste à mesurer la **variabilité du PDP** d'une caractéristique. Cette mesure peut servir comme **indicateur d'importance globale**.
Cependant, elle repose elle aussi sur la **supposition d'indépendance** des variables, et souffre donc des mêmes limites.

Une alternative plus robuste à cette hypothèse est l'**Accumulated Local Effects (ALE)**, qui se base sur la **distribution conditionnelle** plutôt que marginale. ALE est présenté dans la section suivante.

4.4.2 Librairies Python

Les PDP peuvent être facilement générés avec les bibliothèques suivantes :

- `scikit-learn` :
  - Fonction : `sklearn.inspection.PartialDependenceDisplay.from_estimator`
  - Supporte les modèles de type `sklearn`.
  - Exemple :

```
from sklearn.inspection import PartialDependenceDisplay
PartialDependenceDisplay.from_estimator(model, X, features=[0])
```

- `pdpbox` :
  - Outil spécialisé développé par Dr. Terence Parr.
  - Permet une plus grande personnalisation (PDP 1D, 2D, ICE inclus).

4.4.3 Notre utilisation

En terme de rapidité, il s'exécute de façon quasi instantanée (moins d'une seconde) sur nos deux datasets.

Au niveau des résultats, nous avons obtenu un graphique par feature, indiquant la variation de notre variable à expliquer en fonction de notre feature.

CyberSecurity Dataset

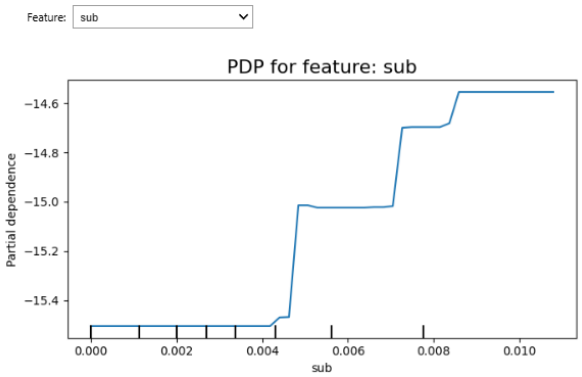

Figure 15: PDP for feature `sub`

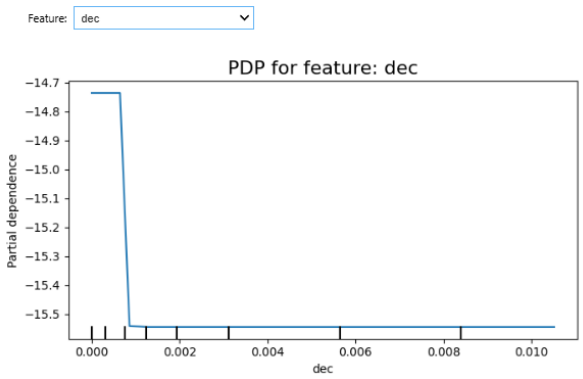

Figure 16: PDP for feature `dec`

Les figures ci-dessus nous permettent de suivre en détail l'impact des valeurs des variables concernées (`sub` et `dec` dans notre cas) sur la prédiction du modèle. Pour la variable `sub`, les résultats

sont plutôt en accord avec la méthode SHAP : les faibles valeurs vont avoir peu d'effets sur le modèle, tandis que les valeurs plus élevées auront un effet plus important qui influencera le modèle vers la classe 1. Pareil pour la variable `dec`, ou les faibles valeurs ont un impact plus important sur la prédiction que les valeurs plus élevées, les résultats sont en accord avec la méthode SHAP.

Les données Parkinsons telemonitoring

Voici deux graphiques que nous avons pu obtenir avec la librairie `scikit-learn`, sur deux variables : `âge`, et `HNR` :

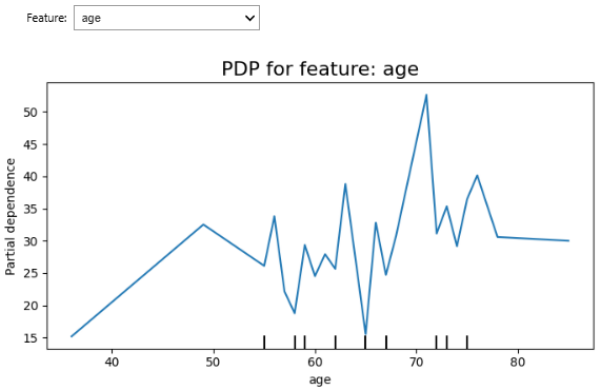

Figure 17: PDP pour la feature `âge`

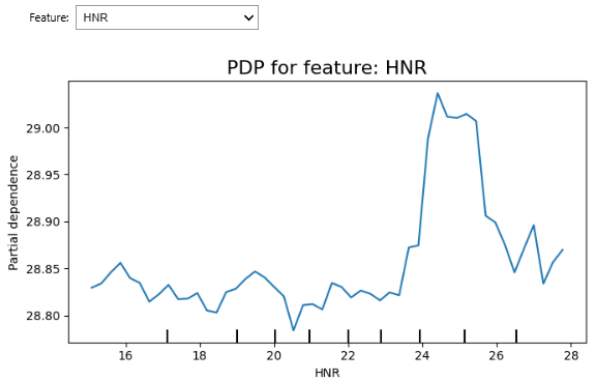

Figure 18: PDP pour la feature `HNR`

Les graphiques ont également été tracés avec la librairie `pdpbox`. Cette librairie permet d'obtenir différents types de graphiques.

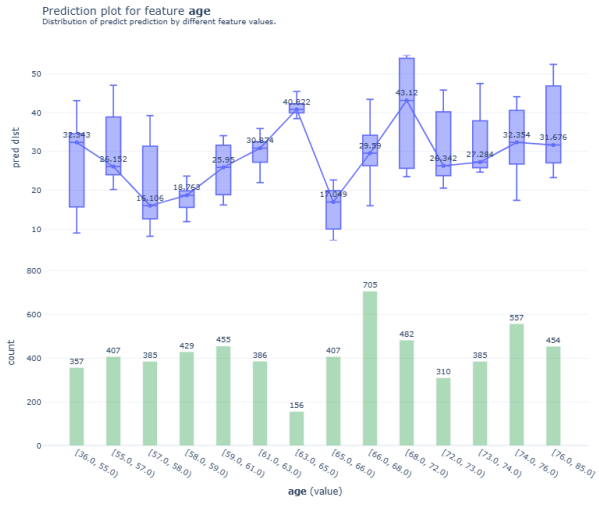

Figure 19: PDP pour la feature `âge` avec `pdpbox.pdp.PDPIsolate`

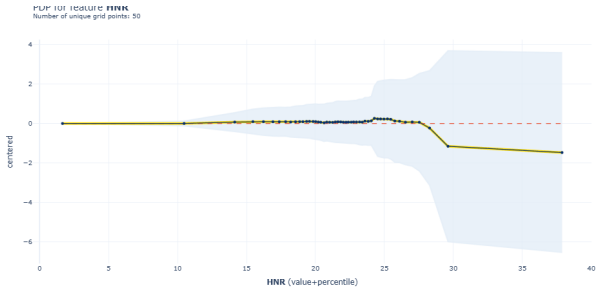

Figure 20: PDP pour la feature `HNR` avec `pdpbox.info_plots.PredictPlot`

Le graphique à gauche représente à la fois la Partial dependence en fonction de la feature choisie (ici l'âge), mais également la distribution de la feature sélectionnée dans les données. Des boîtes à moustache permettent également d'avoir l'incertitude pour chacun des points.

Le graphique à droite ne montre que la partial dependance selon la feature choisie (ici HNR). La zone grise représenté l'incertitude : concrètement ce sont toutes les instances de chaque individu (dont PDP en est la moyenne). Un paramètre permet également d'afficher toutes les instances, afin de plutôt partir sur la méthode ICE, qui sera détaillée plus tard dans le rapport.

Comme indiqué par SHAP, la feature `âge` est celle qui a le plus d'impact sur la prédiction du modèle (on retrouve, dans le cas de PDP la plus grande variation de valeurs de Partial Dependance pour cette feature). De plus, on voit ici qu'elle n'évolue pas de façon linéaire. La feature `HNR` influe moins le modèle, et sa corrélation avec la dépendance partielle n'est pas vraiment linéaire, mais on retrouve un très léger pic vers 25. Pour les valeurs au-dessus l'incertitude est trop importante.

Par rapport au *summary plot* de SHAP, les PDP (Partial Dependence Plots) offrent une visualisation moins synthétique de l'ensemble des variables. En effet, si l'on affiche tous les PDP sur un même graphique avec un axe des ordonnées commun, les variations de nombreuses *features* seront à peine visibles, surtout si certaines ont un effet bien plus marqué que d'autres. À l'inverse, adapter l'échelle des ordonnées à chaque PDP permet de mieux observer l'effet marginal de chaque variable, mais rend plus difficile la comparaison directe entre les *features* pour évaluer leur importance relative.

De plus, cette méthode est exclusivement globale, et suppose qu'il n'y a aucune corrélation entre les variables, contrairement a la méthode ALE qui prend en compte les corrélations.

4.5 ALE – Accumulated Local Effects - méthode globale

L'**Accumulated Local Effects (ALE)** est une méthode d'interprétabilité **globale** conçue pour analyser l'effet d'une caractéristique d'entrée sur la prédiction d'un modèle tout en étant **robuste aux corrélations** entre variables. Contrairement au *Partial Dependence Plot (PDP)*, qui repose sur une **marginalisation** naïve, ALE utilise la **distribution conditionnelle**, évitant ainsi les combinaisons irréalistes de variables.

Elle répond à la question :

> *Quel est l'effet moyen local d'une variable sur la prédiction, en tenant compte de la distribution réelle des données ?*

Intuition

- Le PDP calcule l'effet moyen **en remplaçant** une variable dans toutes les observations — sans tenir compte des relations avec les autres.
- ALE, lui, estime les **variations locales du modèle** dans des **zones de données réellement observées**, puis **accumule** ces effets.

4.5.1 Définition formelle

Soit x_j la variable que l'on souhaite interpréter. La procédure ALE se déroule en quatre étapes principales :

1. **Partitionnement**

   Le domaine de x_j est divisé en K intervalles $[z_{k-1}, z_k]$, souvent définis à partir des **quantiles** des données (bins équi-fréquentiels).

2. **Effets locaux**

   Pour chaque intervalle $[z_{k-1}, z_k]$, on estime la variation moyenne du modèle :

   $$\Delta f_j(z_k) = \mathbb{E}_{\mathbf{x}_{\setminus j} | x_j \in [z_{k-1}, z_k]} \left[f(z_k, \mathbf{x}_{\setminus j}) - f(z_{k-1}, \mathbf{x}_{\setminus j}) \right]$$

   Cela correspond à la **variation locale moyenne** dans chaque bin.

3. **Effets accumulés**

   Les effets sont ensuite **intégrés (accumulés)** à partir de la borne inférieure :

   $$ALE_j(z_k) = \sum_{i=1}^{k} \Delta f_j(z_i)$$

4. **Centrage**

   Pour rendre l'effet **interprétable** (comparaison entre variables), la fonction est centrée autour de zéro :

   $$\tilde{ALE}_j(z_k) = ALE_j(z_k) - \frac{1}{K} \sum_{i=1}^{K} ALE_j(z_i)$$

4.5.2 Formule continue (interprétation dérivée)

L'approche peut être vue comme une **intégration de la dérivée partielle moyenne** du modèle :

$$\hat{f}_{j,ALE}(x_j) = \int_{z_0}^{x_j} \mathbb{E}_{X_C | X_j = z} \left[\frac{\partial \hat{f}(X_j, X_C)}{\partial X_j} \right] dz - \text{constante}$$

où :

- $\hat{f}$ est le modèle de prédiction,
- X_C représente toutes les autres variables,
- La dérivée mesure **l'effet local instantané** de X_j.

Avantages

- **Robuste aux corrélations** entre variables (contrairement au PDP).
- **Respecte la distribution réelle des données**.
- **Plus rapide à calculer** que SHAP pour les grands modèles.
- Permet une **interprétation locale agrégée** (effets locaux → vue globale).

- Facilement visualisable (graphique 2D, parfois 3D pour les interactions).

Inconvénients

- Ne fournit pas d'attribution individuelle (contrairement à SHAP).
- Les effets **ne sont pas additifs** (on ne peut pas sommer les contributions).
- La qualité dépend du **choix du nombre de bins** (partitionnement).
- Méthode encore **peu connue et moins diffusée** que SHAP ou PDP.

4.5.3 Librairies Python

Plusieurs bibliothèques permettent de générer des graphiques ALE :

- `pyALE` :
  - Calcul ALE 1D ou 2D pour détecter des interactions.
  - Supporte nativement les pipelines scikit-learn.
  - Inclut un intervalle de confiance.

```
from pyALE import ale
ale_eff = ale(X=X, model=clf, feature=['feature_name'], include_CI=True)
```

- `alibi` : Attention, ce package nécessite une version de Numpy inférieure à la version 2, ce qui peut poser des problèmes de dépendance vis à vis des autres packages !
  - Permet une analyse 1D avec des estimateurs scikit-learn ou tout modèle compatible.
  - Supporte les explications sur des tableaux `numpy` ou `pandas`.

```
from alibi.explainers import ALE

ale = ALE(predict_fn=clf.predict, feature_names=feature_names)
explanation = ale.explain(X)
```

4.5.4 Notre utilisation

CyberSecurité Dataset

La méthode ALE présente également un coût computationnel élevé. Pour ce jeu de données comportant 290 variables, le calcul des effets à l'aide de cette méthode nécessite **10** minutes, ce qui peut constituer une limitation importante dans un contexte d'analyse à grande échelle.

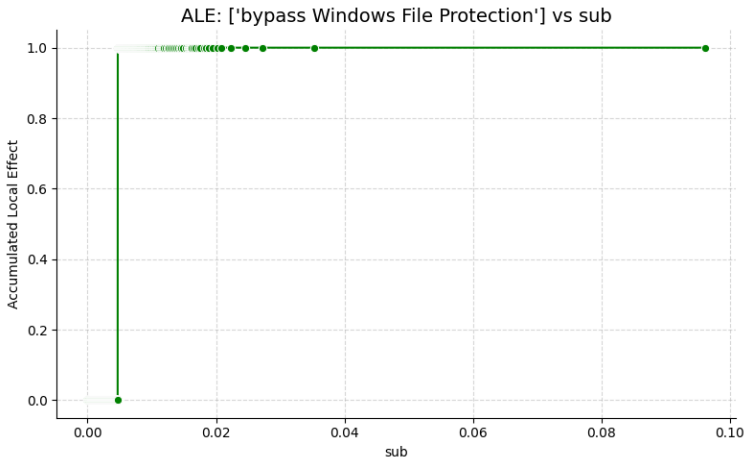

Figure 21: ALE plot for sub variant

Concernant la méthode ALE, l'impact de la variable *sub* sur le comportement *bypass Windows File Protection* est quasi nul pour les faibles valeurs de *sub*, tandis qu'il atteint presque 1 lorsque celles-ci dépassent 0,001. Cette tendance, où les valeurs élevées de la variable sont associées à un impact significatif, est cohérente avec les observations issues des méthodes d'explicabilité précédemment présentées. Cependant, la transition abrupte de l'effet observé dans l'ALE semble davantage liée à la manière dont cette méthode calcule les effets locaux. En comparaison, les méthodes SHAP et PDP présentent des variations plus progressives et moins binaires, ce qui soulève des interrogations quant à la stabilité et la fiabilité de l'interprétation fournie par ALE dans ce cas particulier.

Les données Parkinsons telemonitoring

Voici deux graphiques que nous avons pu obtenir avec la librairie PyALE, sur les deux variables `âge`, et `HNR` :

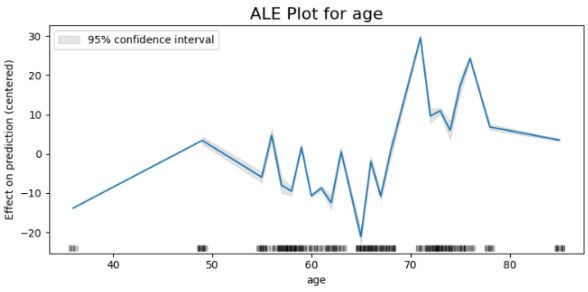

Figure 22: ALE pour la feature `âge`

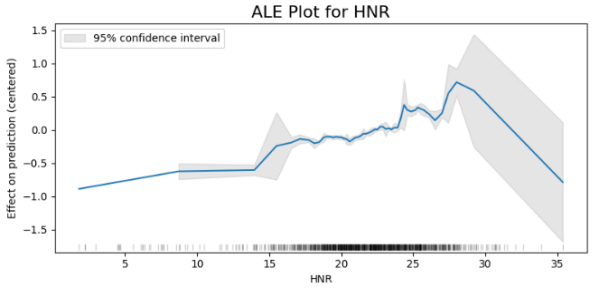

Figure 23: ALE for feature `HNR`

Ces graphiques sont très similaires aux graphiques obtenus avec la méthode PDP, ce qui est plutôt rassurant !

L'ajout de l'intervalle de confiance nous permet de voir qu'il y n'y a quasiment pas d'incertitude sur l'impact de la variable âge sur la prédiction du modèle.

4.6 LOFO – Leave One Feature Out - méthode globale

La méthode **LOFO (Leave One Feature Out)** est une approche d'interprétabilité visant à estimer l'importance d'une variable en évaluant la dégradation des performances d'un modèle lorsqu'on la retire. Pour chaque variable, un nouveau modèle est entraîné sans celle-ci, puis sa performance est comparée à celle du modèle initial. Un changement de performance indique que la variable retirée est importante pour les prédictions.

Cette technique est particulièrement utile pour détecter les variables influentes, mais aussi celles qui ont une *importance négative* : leur suppression peut parfois améliorer les performances, suggérant qu'elles introduisaient du bruit ou de la redondance. LOFO permet donc d'identifier à la fois les attributs essentiels et ceux pouvant être supprimés sans perte (voire avec gain) de qualité prédictive.

Bien que LOFO soit conceptuellement simple et intuitif, elle implique un coût computationnel élevé. En effet, elle nécessite de réentraîner le modèle **autant de fois qu'il y a de variables**, ce qui peut devenir prohibitif dans le cas de modèles complexes ou de jeux de données à haute dimensionnalité. Malgré cela, LOFO reste une méthode robuste pour obtenir une estimation conditionnelle de l'importance des variables.

4.6.1 Fast-LOFO

La méthode LOFO, bien qu'appréciée pour sa robustesse, présente un inconvénient majeur : son coût computationnel élevé. En effet, sa complexité croît linéairement avec le nombre de variables du jeu de données. Par exemple, dans notre cas d'étude en cybersécurité, l'application de LOFO sur un dataset contenant **290 variables** a nécessité environ **97 minutes** de calcul. Sur notre jeu de données Parkinson Telemonitoring, il a fallu 5 minutes de calcul, là où les méthodes précédentes sont à moins de 30 secondes de calcul.

Pour pallier à ce problème, il existe une implémentation Fast-LOFO permettant de réduire le temps d'exécutions. Grâce à cette implémentation, nous avons pu passer de **97 minutes à 5 minutes** pour les données de cybersécurité, et de **5 minutes à 4 secondes** pour les données Parkinson telemonitoring.

Fast-LOFO est plus rapide que la méthode LOFO classique car le modèle n'est pas ré-entraîné à chaque fois qu'une feature est enlevée. Pour chaque variable, ses valeurs sont permutées de manière pseudo-aléatoire, mais au sein de groupes de données similaires, formés à partir de paires de variables choisies aléatoirement. Le modèle prédit ensuite sur ces jeux modifiés, et la moyenne de la baisse de performance mesure l'importance de la variable.

Cette méthode Fast-LOFO a été implémentée dans la librairie `lofo-importance` : https://github.com/aerdem4/lofo-importance?tab=readme-ov-file#flofo-importance

4.6.2 Théorie

Définition des variables

- Modèle initial : $\hat{f}$
- Données d'entraînement : $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$
- Données de test : $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$

- Fonction de perte : $L(y, \hat{y})$
- Nombre d'exemples de test : n_{test}

1. Erreur du modèle initial

On commence par calculer l'erreur moyenne du modèle initial :

$$e_{\text{orig}} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} L\left(y_{\text{test}}^{(i)}, \hat{f}(x_{\text{test}}^{(i)})\right)$$

2. Boucle sur chaque variable $j \in \{1, ..., p\}$

Pour chaque variable j :

- Supprimer la variable j des jeux d'entraînement et de test pour obtenir $\mathbf{X}_{\text{train},-j}$ et $\mathbf{X}_{\text{test},-j}$
- Réentraîner un nouveau modèle $\hat{f}_{-j}$ sur $(\mathbf{X}_{\text{train},-j}, \mathbf{y}_{\text{train}})$
- Calculer l'erreur :

$$e_{-j} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} L\left(y_{\text{test}}^{(i)}, \hat{f}_{-j}(x_{\text{test},-j}^{(i)})\right)$$

3. Importance LOFO de la variable j

On peut l'exprimer de deux manières :

- **Comme un rapport** :
$$\text{LOFO}_j = \frac{e_{-j}}{e_{\text{orig}}}$$
- **Ou comme une différence** :
$$\text{LOFO}_j = e_{-j} - e_{\text{orig}}$$

Plus LOFO_j est élevé, plus la variable j est jugée importante.

4. Tri et Visualisation

On trie les variables selon leurs scores LOFO_j décroissants, puis on les visualise à l'aide de graphiques (par exemple des barres horizontales avec `plot_importance`).

Avantages

- Méthode simple et intuitive : chaque variable est testée en conditions réelles.
- Prend en compte les interactions entre variables.
- Fournit une **importance conditionnelle**, contrairement aux approches univariées.
- Permet de détecter des variables nuisibles (importance négative).
- Possibilité d'optimisation par parallélisme ou sous-échantillonnage.

Inconvénients

- Coût computationnel élevé : nécessite de réentraîner le modèle pour chaque variable.
- Inadaptée pour les modèles très lents ou les jeux de données massifs sans optimisation.
- Sensibilité à la taille et à la qualité du jeu de test.
- Instabilité possible avec des modèles peu robustes ou des données bruitées.

4.6.3 Librairies Python

Nous n'avons trouvé qu'une seule librairie permettant d'implémenter la méthode LOFO : `lofo-importance`. Cette librairie nous a permis d'obtenir directement les graphiques souhaités, elle implémente également la méthode Fast-LOFO.

4.6.4 Notre utilisation

Les données Cyber Securité

Les figures 24 montrent l'importance de chaque feature dans la prédiction du modèle avec une incertitude calculée. Ici, pour des raisons de visibilité, seulement les 7 variables les plus importantes sont affichées.

Les valeurs de l'importance des variables ont changé avec la méthode de calcul qui est différente, ce qui met en relief le choix des méthodes d'explicabilité. Un expert en Cybersecurité peut confirmer les résultats obtenues et du coup évaluer la méthode et confirmer le choix de la méthode.

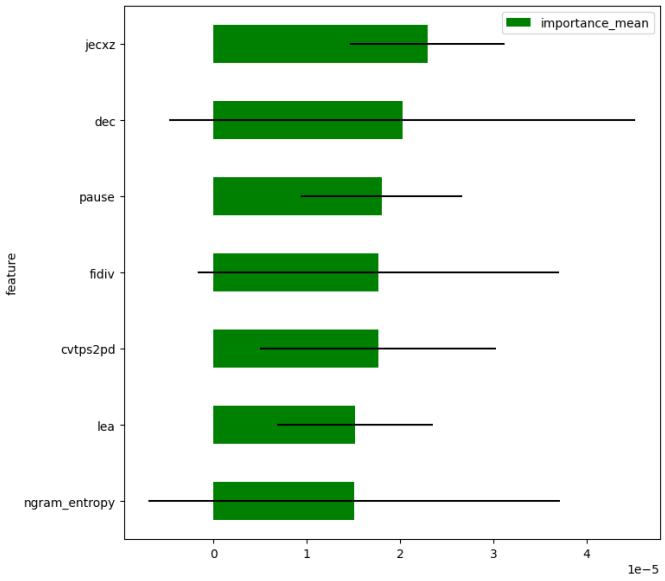

Figure 24: LOFO importance pour les sept premieres features.

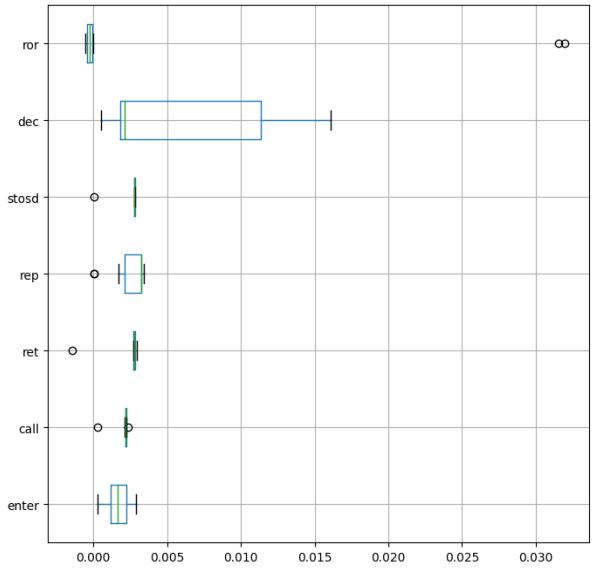

Figure 25: Fast-LOFO importance pour les sept premieres features.

Les données Parkinsons telemonitoring

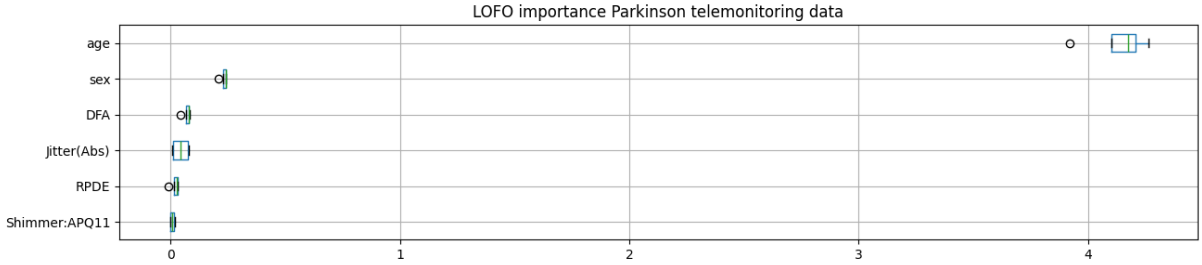

Figure 26: LOFO importance pour les 6 premières features

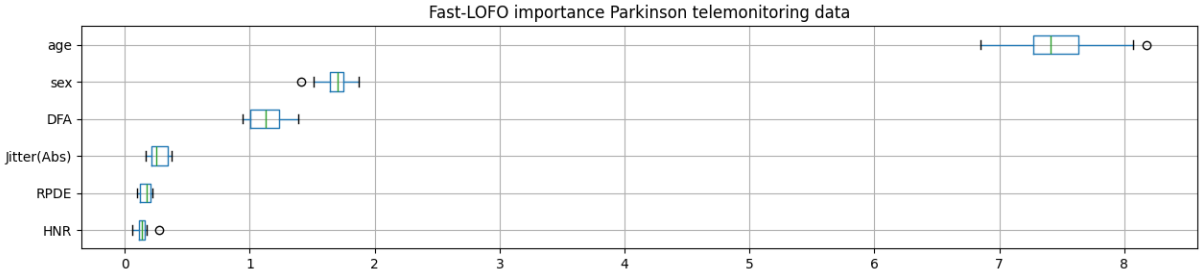

Figure 27: Fast-LOFO importance pour les 6 premières features

Voici, pour les deux graphiques ci-dessus les 6 features les plus importantes trouvées avec la méthode LOFO.

Cette fois ci, 5 des premières variables sont exactement les mêmes entre LOFO et Fast-LOFO. Ce constat est positif, car ces mêmes variables sont également mises en avant par la méthode

SHAP globale, ce qui renforce leur pertinence. De plus, la variable de l'`âge` reste encore une fois, de loin, la plus influente dans le modèle.

Notre conclusion sur LOFO et Fast-LOFO

Les différences entre les résultats de LOFO et Fast-LOFO peuvent s'expliquer par la nature même de Fast-LOFO, qui repose sur des approximations destinées à accélérer le calcul. Contrairement à LOFO, qui évalue rigoureusement la contribution individuelle de chaque variable en mesurant la perte de performance après exclusion d'une feature, Fast-LOFO emploie des méthodes heuristiques ou des sous-échantillonnages pour réduire le temps de calcul. Ces simplifications peuvent introduire une certaine variabilité dans le classement des variables, surtout lorsque leur importance relative est proche.

4.7 LIME - Local Interpretable Model-agnostic Explanations - méthode locale

LIME est une méthode d'interprétabilité **locale** qui vise à expliquer la prédiction d'un modèle d'apprentissage automatique en approximant localement ce modèle par un modèle linéaire interprétable.

L'idée est de **générer des données synthétiques** proches de l'instance à expliquer, d'**évaluer leur prédiction** avec le modèle initial, puis d'**entraîner un modèle simple et interprétable** sur ces exemples, en les **pondérant selon leur similarité** avec l'instance d'origine.

Elle est *agnostique au modèle* (*model-agnostic*) : elle peut être utilisée avec n'importe quel modèle complexe.

Elle répond à la question :

Pourquoi le modèle a-t-il pris cette décision pour cette instance particulière ?

4.7.1 Définition formelle

LIME cherche à expliquer la prédiction $f(x)$ d'une instance x par une approximation locale $g(x')$:

- f est le modèle complexe et potentiellement non interprétable.
- x est l'instance originale.
- x' est une représentation simplifiée (binaire ou discrète) de x.
- g est un modèle simple (souvent linéaire ou arbre peu profond), interprétable.

L'idée est de générer des **données synthétiques autour de** x, d'évaluer leur prédiction avec f, puis d'ajuster un modèle g sur ces exemples **pondérés selon leur proximité** à x.

Formulation mathématique

Le problème d'optimisation résolu par LIME est le suivant :

$$\xi(x) = \arg\min_{g \in G} \ \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

où :

- G est la famille des modèles interprétables (e.g., modèles linéaires).
- $\mathcal{L}(f, g, \pi_x)$ est une mesure de fidélité locale entre f et g, pondérée par la proximité π_x entre les instances générées et x.
- $\Omega(g)$ est une régularisation imposant la simplicité de g.

Méthodologie

1. **Binarisation ou simplification de l'entrée** $x \to x'$ (texte $\to$ présence/absence de mots, image $\to$ superpixels, etc.).
2. **Perturbation locale** : génération d'un grand nombre d'instances proches de x' en modifiant certaines caractéristiques.
3. **Évaluation du modèle** f sur ces instances perturbées.
4. **Pondération par proximité** : les instances proches de x ont un poids plus élevé.
5. **Ajustement d'un modèle simple** g sur ces données pondérées.
6. **Interprétation** de la prédiction par les coefficients de g.

Exemple d'interprétation linéaire

Dans le cas d'un modèle linéaire local, la prédiction expliquée s'écrit :

$$g(x') = w_0 + \sum_{j=1}^{M} w_j x'_j$$

où w_j reflète l'influence de la caractéristique j sur la prédiction locale.

Avantages

- **Universel** : s'applique à tout type de modèle (black-box).
- **Flexible** : fonctionne pour les données tabulaires, textes, images.
- **Local et intuitif** : donne une explication ciblée instance par instance.
- **Rapide** à calculer pour une instance donnée.

Limites

- **Stabilité faible** : les explications peuvent varier fortement selon les perturbations.
- **Qualité dépendante de la représentation** : le choix du mapping $x \to x'$ est crucial.
- **Modèle local potentiellement mal ajusté** si la surface de décision est trop complexe localement.
- **Pas d'explication globale** (uniquement locale).

4.7.2 Librairies Python

- `lime` : bibliothèque officielle pour LIME développée par Ribeiro et al.

```
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(X_train, feature_names=feature_names,
                                 class_names=class_names, mode='classification')

exp = explainer.explain_instance(X_test[i], model.predict_proba, num_features=5)
exp.show_in_notebook()
```

- `alibi` (par Seldon) : alternative avec davantage de contrôles, compatible scikit-learn, XGBoost, etc.

```
from alibi.explainers import Lime

explainer = Lime(predict_fn=model.predict_proba, feature_names=feature_names)
explanation = explainer.explain(X[i])
```

4.7.3 Notre utilisation

Pour l'application de `lime`, le package `lime` ne semble plus maintenu depuis 2021. Nous avons tout de même pu l'utiliser, mais cela rend l'utilisation moins optimale.

Nous avons donc décidé de tester également la partie lime de la librairie `interpret`.

CyberSécurité Dataset

Voici les graphiques obtenus à l'aide de la librairie `lime` :

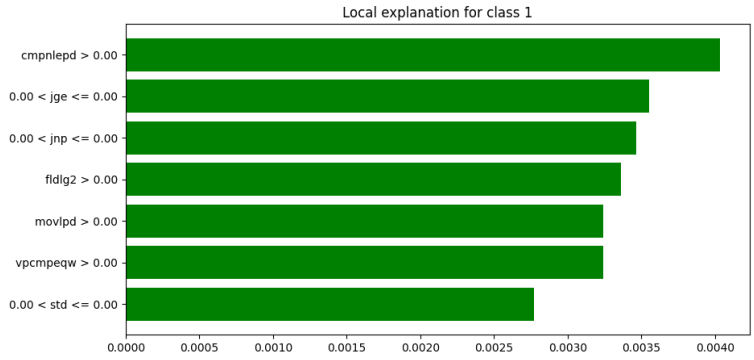

Figure 28: LIME feature importance

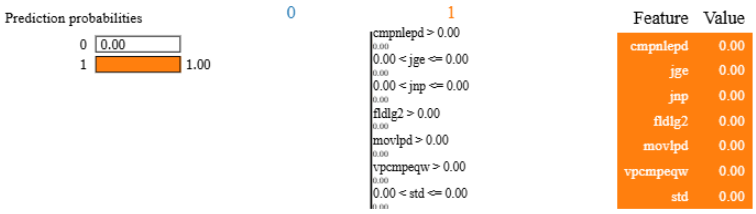

Figure 29: LIME feature model decision

Une fois encore, nous avons sélectionné un fichier présentant le comportement cible, et le modèle l'a bien prédit comme tel.

Cependant, cette fois-ci, de nouvelles variables explicatives émergent, différentes de celles mises en évidence par les autres méthodes d'explicabilité.

En comparaison, voici ci-dessous ce que nous avons obtenu avec la librairie `intepret`. Il faut tout de fois noter que le temps de calcul est bien plus long avec la librairie `intepret`, si on cherche à l'appliquer sur les 290 lignes de notre jeu de test. Il est également possible de ne l'appliquer que sur cetaines lignes, afin de réduire ce temps de calcul.

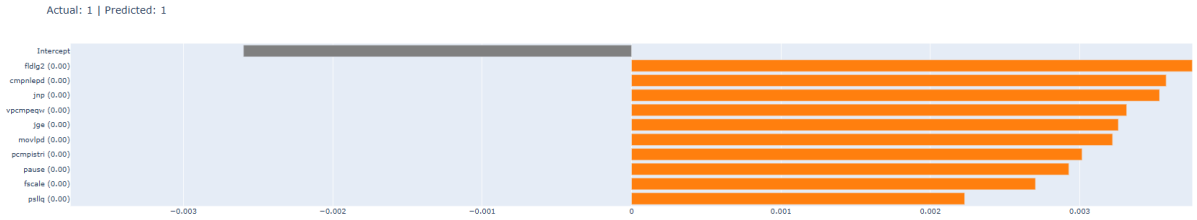

Figure 30: LIME feature model decision with `interpret` package

Ici, les caractéristiques mises en évidence ne correspondent pas exactement à celles obtenues avec la librairie `lime`. Cela peut s'expliquer notamment par le caractère non déterministe de LIME : en effet, deux exécutions avec les mêmes données peuvent conduire à des résultats différents.

Dans ce contexte, il semble donc difficile d'évaluer avec certitude la fiabilité des explications fournies par LIME sur ce jeu de données.

Parkinson Dataset

L'importance absolue des variables est présentée dans la Figure 29. Elle a été obtenue en appliquant LIME localement à chaque instance de l'ensemble de test, puis en agrégeant les contributions absolues de chaque variable sur l'ensemble des prédictions. Ce processus permet d'estimer une importance globale des variables à partir d'explications locales.

Comme on peut le constater, la variable **âge** ressort comme la plus influente dans les prédictions du modèle. Elle est segmentée en quatre tranches, ce qui est cohérent avec les résultats obtenus par d'autres méthodes présentées dans ce rapport. De manière similaire, d'autres variables sont également représentées sous forme de catégories ou de phases.

Le graphique met en évidence les 10 variables les plus importantes. On observe une bonne concordance avec les résultats obtenus via l'analyse SHAP, ce qui renforce la robustesse de l'interprétation. Il est toutefois important de noter que ce graphique, basé sur des valeurs absolues, ne permet pas

de distinguer si une variable contribue positivement ou négativement à la prédiction. Pour cela, une analyse locale ou l'utilisation de SHAP est plus adaptée.

Cette représentation globale permet néanmoins d'identifier les variables les plus déterminantes pour le modèle, tout en facilitant la validation par rapport aux connaissances métier.

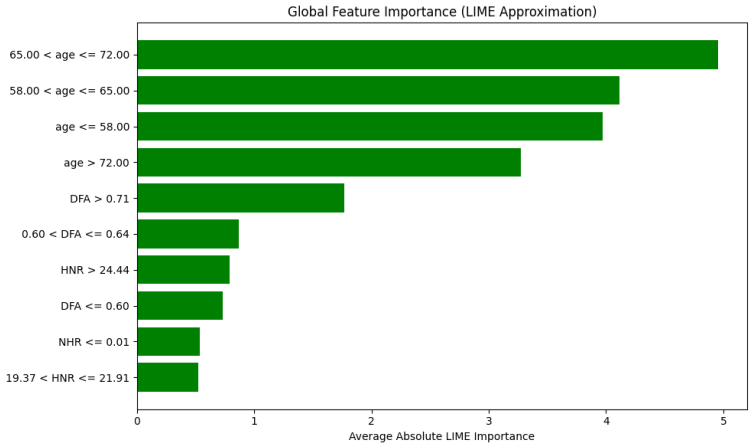

Figure 31: Absolute Feature importance using Lime

La Figure 30 présente l'explication locale générée par LIME pour l'instance 2. Cette visualisation décompose la prédiction du modèle en contributions individuelles de chaque variable. La prédiction estimée est de **15,28**, dans un intervalle global allant de **8,60** à **54,46**. Chaque barre horizontale représente l'effet d'une variable sur la prédiction finale : en bleu, les variables qui tirent la prédiction vers le bas, et en orange, celles qui la tirent vers le haut .

Dans ce cas, la variable **age <= 58.00** est le facteur principal qui réduit la prédiction. D'autres variables comme **DFA > 0.71**, **Jitter(%) > 0.01**, ou **Shimmer:DDA > 0.06** ont également un effet modérément négatif. À l'opposé, des variables telles que **Shimmer:APQ3 > 0.02**, **sex <= 0.0** (potentiellement le sexe féminin), et **NHR > 0.04** contribuent positivement à la valeur prédite. Ces résultats montrent que même si l'âge tend à faire baisser la prédiction, certaines variables acoustiques et démographiques compensent partiellement cet effet.

Ce type d'explication locale permet de mieux comprendre les facteurs spécifiques qui influencent la prédiction pour un individu donné, ce qui est particulièrement utile dans des contextes médicaux ou décisionnels nécessitant de la transparence.

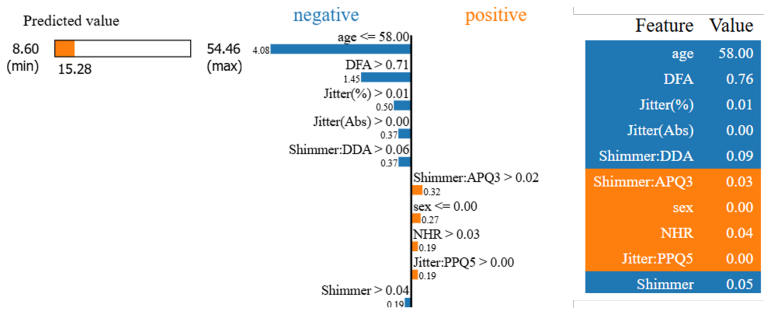

Figure 32: LIME on Parkinson dataset

4.8 ICE – Individual Conditional Expectation - méthode locale

L'**Individual Conditional Expectation (ICE)** est une méthode d'interprétabilité **locale** qui permet de comprendre l'effet d'une variable sur la prédiction **pour chaque instance** individuellement.

Elle est complémentaire au **PDP (Partial Dependence Plot)**. En effet, **PDP** réalise une moyenne de toutes les courbes ICE, afin d'obtenir un effet moyen global, alors que **ICE** montre les **variations individuelles** pour chaque instance. ICE permet, entre autre de **vérifier la fiabilité du PDP** (en cas de grande variabilité des ICE).

Elle répond à la question :

> *Comment la prédiction varie-t-elle pour une instance donnée lorsqu'on modifie une caractéristique d'entrée ?*

4.8.1 Principe général

Pour une variable x_j, l'ICE trace la fonction :

$$\hat{f}^{(i)}(x_j) = \hat{f}(x_j, x_C^{(i)})$$

pour chaque individu i, où $x_C^{(i)}$ correspond aux autres variables de l'individu. On observe donc l'évolution de la prédiction f en faisant varier x_j tandis que toutes les autres caractéristiques sont **fixées**.

Interprétation

- Les **différences entre courbes ICE** révèlent des **interactions** : si toutes les courbes sont parallèles, il n'y a pas d'interaction entre x_j et les autres variables.
- Des courbes **divergentes** indiquent que l'effet de x_j dépend des autres caractéristiques.

Variantes

- **Centered ICE (c-ICE)** : les courbes sont centrées à une valeur de référence de x_j (souvent la valeur réelle de l'individu), pour mieux observer les écarts relatifs.
- **Derivative ICE** : trace la dérivée des courbes ICE pour observer l'influence locale.

Avantages

- Fournit une **vue locale fine** du comportement du modèle.
- Permet de **détecter des interactions** et des **effets hétérogènes**.
- **Plus fidèle** que PDP dans les contextes non linéaires ou non additifs.

Limites

- Peu lisible si le nombre d'instances affichées est trop grand.
- Peut être instable si les données sont bruitées ou rares dans certaines zones.
- Ne donne pas une mesure résumée ou un score numérique (interprétation visuelle uniquement).

4.8.2 Librairies Python

- `scikit-learn` :

```
from sklearn.inspection import PartialDependenceDisplay
PartialDependenceDisplay.from_estimator(
    model, X, features=[feature_index], kind='individual'
)
```

- `pdpbox` :

  Permet de générer PDP et ICE dans un même graphique :

```
from pdpbox.pdp import pdp_isolate, pdp_plot

pdp_iso = pdp_isolate(model=model, dataset=X, model_features=features,
                      feature='feature_name', num_grid_points=20)

pdp_plot(pdp_iso, plot_lines=True, frac_to_plot=1.0)
```

- `interpret` (Microsoft) : supporte l'affichage ICE interactif dans un navigateur.

4.8.3 Notre utilisation

Nous avons utilisé la librairie `scikit-learn`, qui offre des visualisations ICE.

CyberSécurité Dataset

Pour ce fichier (courbe verte), la valeur de la variable `sub` n'impacte pas vraiment la prédiction du modèle. D'ailleurs, c'est très souvent le cas, puisque en général, peu importe la valeur de `sub`, les probabilités prédites sont soit à 0, soit à 1.

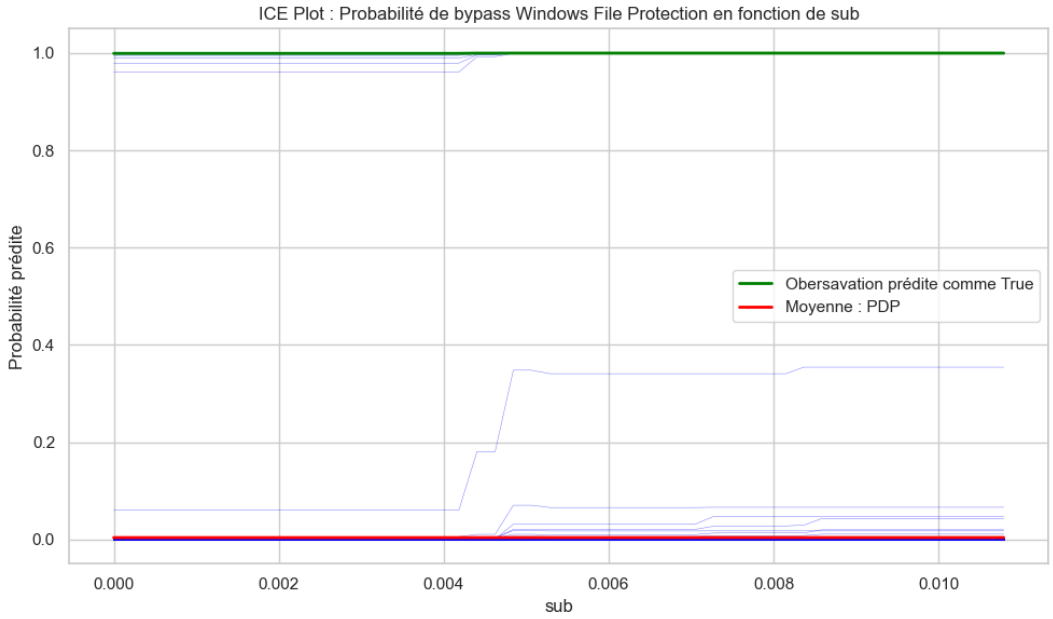

Figure 33: Graphique ICE pour la variable `sub`

Nous avons généré des graphiques ICE pour de nombreuses variables qui présentent pourtant un effet significatif. À chaque fois, les courbes obtenues étaient similaires à celle présentée ici. Cela peut s'expliquer par le très grand nombre de variables : isolément, chaque feature a peu d'influence sur la prédiction, tandis que c'est la combinaison de plusieurs d'entre elles qui joue un rôle déterminant dans le comportement du modèle.

Ce type de situation illustre bien la nécessité de rester prudent dans l'interprétation des résultats issus de la méthode PDP, qui peut masquer des effets d'interaction importants et ne pas être très représentatif.

Parkinson Dataset

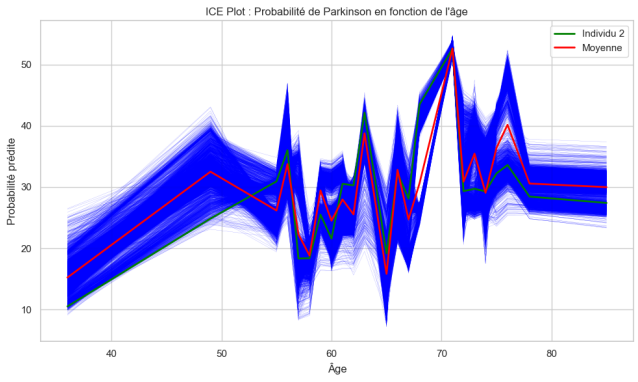

Figure 34: ICE plot pour la variable age

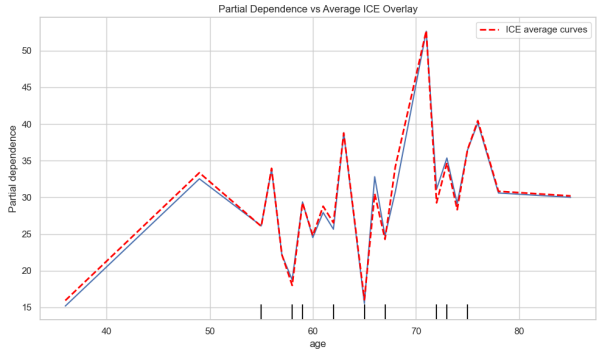

Figure 35: ICE average et PDP Comparaison pour la variable age.

La Figure 34 illustre l'effet de la variable *âge* sur la prédiction du diagnostic de la maladie de Parkinson à l'aide des courbes ICE. Chaque ligne bleue représente l'effet marginal de l'âge sur la

prédiction pour un individu spécifique. Par exemple, en vert la courbe représente l'individu 2. On observe que les courbes ne sont pas parfaitement parallèles, ce qui indique une certaine interaction entre l'âge et d'autres variables dans le modèle. Certaines courbes présentent une augmentation ou une diminution plus marquée de la prédiction en fonction de l'âge, suggérant une hétérogénéité de comportement entre individus. La courbe rouge représente la moyenne de toutes les courbes ICE, ce qui donne une estimation globale de l'effet de l'âge. Cette moyenne permet de dégager une tendance générale : par exemple, si la courbe rouge monte avec l'âge, cela indique que le risque prédit de Parkinson augmente en moyenne avec l'âge.

La Figure 35 compare cette moyenne des courbes ICE avec la courbe PDP, précédement introduit, pour la même variable. Le PDP fournit une estimation de l'effet moyen de l'âge sur la prédiction en marginalisant sur les autres variables. Comme la moyenne des courbes ICE revient conceptuellement à la même opération que le PDP, ces deux approches devraient produire des résultats similaires sous certaines hypothèses, notamment l'indépendance relative entre les variables. La comparaison entre les deux permet donc de valider la cohérence de l'analyse. En cas de divergence notable entre la moyenne des ICE et le PDP, cela peut révéler des interactions importantes entre l'âge et d'autres variables, ou bien des effets non linéaires complexes que le PDP ne capture pas entièrement. Cette analyse conjointe permet ainsi de mieux comprendre le comportement du modèle et la robustesse des interprétations.

Dans notre cas, on observe que les courbes ICE présentent des tendances similaires dans différentes régions du graphe. Elles suivent globalement la même forme, avec un nombre comparable de pics localisés à des âges proches. Cela suggère une certaine cohérence dans la manière dont l'âge influence la prédiction à travers les individus. Toutefois, on note également des divergences dans certaines zones, notamment entre 40 et 55 ans, où les valeurs peuvent varier sensiblement d'un individu à l'autre. Cette variabilité indique que d'autres facteurs que l'âge influencent la prédiction du modèle dans cette tranche d'âge, suggérant des interactions avec d'autres variables explicatives.

4.9 Anchors – Scoped rules - méthode locale

La méthode des **Anchors** est une technique d'interprétabilité **locale**, fondée sur des **règles conditionnelles simples**, qui vise à expliquer la prédiction d'un modèle pour une instance particulière. Elle a été proposée par l'équipe de Marco Ribeiro (créateur de LIME) comme alternative plus robuste aux approches par approximation linéaire.

Elle répond à la question :

> *Sous quelles conditions (règles), la prédiction du modèle reste-t-elle globalement inchangée ?*

4.9.1 Principe général

Pour une instance donnée, un **anchor** est une règle logique (de type "SI... ALORS...") qui fixe certaines caractéristiques d'entrée. Si cette règle est satisfaite, le modèle donne la **même prédiction** avec une forte probabilité, même si les autres caractéristiques changent.

- Exemple : *SI âge* < 60 *ET statut = retraité, ALORS prédiction = 'non-remboursement' avec 95% de précision.*

- L'ancre identifie un **sous-espace local** où le comportement du modèle est **stable**.

Fonctionnement

- L'algorithme cherche une règle minimaliste (peu de conditions) mais suffisamment **précise** (prédiction stable).
- Pour cela, il génère de nombreuses instances synthétiques proches de l'observation cible (par perturbation aléatoire) et évalue la **précision conditionnelle** de chaque règle candidate.
- La meilleure ancre maximise un compromis entre **précision** (fidelity) et **couverture** (proportion d'instances concernées).

Interprétation

- Les ancres permettent de **comprendre les décisions locales** sous forme de règles simples, compréhensibles par un humain.
- Une règle avec une précision de 98% signifie que, dans 98% des cas où la règle est satisfaite, le modèle prédit la même classe.
- La **couverture** indique la proportion d'instances (dans le voisinage local) pour lesquelles la règle est applicable. Une couverture élevée signifie que la règle explique un grand nombre de cas similaires, tandis qu'une faible couverture indique une explication plus spécifique mais moins généralisable.

Utilisation recommandée

- Pour fournir une **explication claire et textuelle** d'une prédiction.
- Pour **auditer localement** le modèle (stabilité, robustesse).
- Pour aider à la prise de décision humaine dans des contextes sensibles (santé, crédit, etc.).

Avantages

- Produit des explications **simples, compréhensibles et actionnables**.
- Capte les **zones de stabilité locale** du modèle.
- Moins sensible au bruit que les méthodes linéaires comme LIME.
- Permet de savoir quand le modèle est **confiant** dans sa prédiction.

Limites

- Peut ne pas trouver d'ancre satisfaisante (avec assez de précision et couverture) pour certains cas.
- Explications locales uniquement : ne donne pas de vue globale du modèle.
- Nécessite de générer beaucoup d'instances proches (coût computationnel).

4.9.2 Librairie Python

Le package `alibi` (de Seldon) est le seul package qui offre une implémentation des *anchors*.

```
from alibi.explainers import AnchorTabular

explainer = AnchorTabular(predict_fn=model.predict, feature_names=feature_names)
explainer.fit(X_train)
explanation = explainer.explain(instance)

print(explanation.anchor)
print(explanation.precision)
```

4.9.3 Notre utilisation

CyberSécurité Dataset

Pour le dataset de cybersécurité, nous avons été confrontés à deux problèmes lors de l'utilisation des *Anchors* :

- **Le temps de calcul pour chercher des ancres** : notre jeu de données ayant beaucoup de colonnes (290), il existe de nombreuses combinaisons possibles. Ainsi, le temps peut vite être exponentiel, si l'on n'est pas trop restrictif sur la couverture de notre règle. Pour pallier à ce problème, nous avons ré-entraîné des modèles sur les 11 features les plus influentes uniquement. Le meilleur modèle était un `RandomForest`, avec un F1-score de 0.88.
- **Trouver des règles avec une bonne couverture** : Une fois les nouveaux modèles ré-entraînés, nous avons de nouveau cherché des règles. Néanmoins, nous n'avons pas réussi à trouver une règle avec une couverture et une précision satisfaisante. Voici ci-dessous la meilleure règle trouvée :

Règle d'ancrage trouvée:
`vpcmpeqw > 0.00`, `shr > 0.00`, `pause > 0.00`, `num_jumps <= 3014.00`,
`pxor > 0.00`, `pcmpistri > 0.00`, `andpd > 0.00`, `movzx > 0.01`

Prédiction :	1
Précision de la règle :	0.56
Couverture de la règle :	0.01

Malheureusement, cette règle ne couvre qu'1% des données, et n'est précise qu'à 56%, ce qui ne permet pas d'en fait une règle très fiable.

Parkinson Dataset

Comme le Parkinson Dataset est de taille plus petite, le calcul de l'anchor était plus rapide et on a obtenu la réponse en quelque seconde. La Figure ci-dessous montre la règle d'ancrage trouvé pour l'individu 2.

Pour cette règle : age <= 65.00, DFA > 0.71, RPDE <= 0.54, Shimmer:APQ5 > 0.01, Shimmer:DDA > 0.03, Jitter(%) > 0.01. Le modèle a effectué une **prédiction de classe 0** avec une **précision de 96 %** pour la règle d'ancrage identifiée. Toutefois, cette règle ne couvre qu'environ **1 %** des données, ce qui reflète une **forte hétérogénéité** du jeu de données. En conséquence, bien que la règle soit très précise localement, sa **portée limitée** réduit sa fiabilité en tant qu'explication généralisable à d'autres cas.

5 Conclusion

Pour conclure, ce fil rouge nous a offert l'opportunité de réaliser une revue approfondie des différentes méthodes d'explicabilité des modèles, tout en les mettant concrètement en œuvre.

Nous avons ainsi pu identifier les approches les plus pertinentes, robustes et adaptées à nos deux jeux de données. Le fait de les manipuler directement nous a permis d'en comprendre finement le fonctionnement, les paramètres influents, ainsi que leurs limites.

Le fait que nous ayons tous les deux travaillé sur l'ensemble des méthodes et exploré les deux datasets nous permet aujourd'hui d'avoir une vision globale, à la fois théorique et pratique, de ces techniques d'explicabilité.

Vous trouverez ci-dessous deux tableaux résumant notre travail. Le premier compare chacune des méthodes explorées, avec notamment notre niveau de recommandation. Le second recense pour chaque méthode les librairies Python disponibles afin de pouvoir les implémenter rapidement.

Méthode	**Locale**	**Globale**	**Temps de calcul**	**Résultats consistants**	**Nous recommandons ?**
SHAP	✓	✓	Moyen	Oui (théorie des jeux)	Oui, analyse très complète
LIME	✓	✗	Moyen à élevé	Non	Moyen, peu maintenu
Anchors	✓	✗	Moyen	Oui (plus stable que LIME)	Oui, pour certains cas
ICE	✓	✗	Élevé si nombreuses variables	Oui (sensibilité aux interactions)	Oui, donne une vision intéressante
PDP	✗	✓	Moyen	Variable (corrélations possibles)	Non, plutôt privilégier ICE pour du local ou ALE pour du global
ALE	✗	✓	Faible à moyen	Oui (corrige biais de PDP)	Oui, plus robuste que PDP
LOFO	✗	✓	Très élevé	Oui (si bien paramétré)	Oui, angle d'approche différents des autres méthodes
EBM	✓	✓	Elevé	Oui (stable et transparent)	Oui, si les modèles offrent des bonnes performances

Table 3: Comparaison des principales méthodes d'explicabilité selon leur usage, leur coût computationnel et leur fiabilité.

Package	EBM	SHAP	LOFO	ALE	PDP	ICE	LIME	Anchors
`interpret`	♡	✓					✓	
`shap`		♡						
`shapash`		♡						
`lofo-importance`			♡					
`PyALE`				♡				
`pdpbox`					♡	✓		
`scikit-learn`					✓	✓		
`lime`							✓	
`alibi`		✓		✓			✓	✓

Table 4: Couverture des méthodes d'explicabilité par package Python.
♡: nos coups de cœur, ✓: implémenté

6 Annexe

6.0.1 TF-IDF

Les équations suivantes ont été utilisées pour calculer les indicateurs TF-IDF (le paramètre t est un *terme*, et le paramètre d un *document*).

$$\text{TF}(t, d) = \frac{\text{nombre d'occurrences de } t \text{ dans } d}{\text{nombre total de termes dans } d}$$

$$\text{IDF}(t) = \log\left(\frac{\text{nombre total de documents}}{\text{nombre de documents contenant } t}\right)$$

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

References

Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232.

Erik Štrumbelj and Igor Kononenko. 2011. A general method for visualizing and explaining black-box regression models. In *Adaptive and Natural Computing Algorithms*, pages 21–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

Erik Štrumbelj and Igor Kononenko. 2014. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3).

Rich Caruana, Paul Koch, Yin Lou, Marc Sturm, Johannes Gehrke, and Noemie Elhadad. 2015. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730, Sydney, Australia.

Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin and. 2015. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA. Association for Computing Machinery.

Scott Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman and. 2018. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111.

Daniel W. Apley and Jingyu Zhu. 2020. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 82(4):1059–1086.

List of Figures

List of Tables