UNIVERSITÉ PARIS XIII – SORBONNE PARIS NORD

École doctorale Galilée [Laboratoire d'Informatique de Paris-Nord CNRS UMR 7030]

Adaptive Learning Applied to Fraud Detection

Apprentissage Adaptatif Appliqué à la Détection de la Fraude

Par Kodjo Mawuena AMEKOE

Thèse de doctorat d' Informatique

Dirigée par Hanane AZZAG

Et par Mustapha LEBBAH

Présentée et soutenue publiquement le 03/04/2025

Devant un jury composé de :

Nicolas LABROCHE, Prof.	Université de Tours, France	Rapporteur
Ndèye NIANG-KEITA, Prof.	CNAM Paris, France	Rapportrice
Thierry CHARNOIS, Prof.	Université Sorbonne Paris Nord, France	Examinateur
Haytham ELGHAZEL, Mcf	Université Lyon 1, France	Examinateur
Hanane AZZAG, Prof.	Université Sorbonne Paris Nord, France	Directrice de thèse
Mustapha LEBBAH, Prof.	Université Paris Saclay, France	Co-directeur
Zaineb CHELLY-DAGDIA, Mcf-Hdr	Université Paris Saclay, France	Co-encadrante
Grégoire JAFFRE, Ing.	Groupe BPCE	Co-encadrant

Résumé

Titre: Apprentissage Adaptatif Appliqué à la Détection de la Fraude

Mots clefs: Apprentissage adaptatif; apprentissage automatique; Interprétabilité; Mécanisme d'Attention; Ensemble d'Arbres de Décision; Détection de Fraude

Résumé: La fraude aux moyens de paiement, dont sont victimes les institutions bancaires, peut emprunter plusieurs canaux (par exemple : chèque, carte bancaire, virement) et entraîner d'importantes pertes financières ou des désagréments pour les clients, notamment en cas de fausses alertes. La lutte contre la fraude est donc une nécessité pour les banques et se traduit par un cycle conflictuel entre les équipes dédiées à la sécurité, qui mettent en place des algorithmes de détection et de blocage, et les fraudeurs, qui adaptent leurs stratégies pour les contourner. Malgré les efforts visant à utiliser des approches de *machine learning*, les pertes liées à la fraude restent significatives, se chiffrant à des milliards de dollars chaque année. Il devient alors crucial de questionner l'efficacité des modèles de machine learning (ML) classiques, qui sont par nature plus statiques et, par conséquent, peut-être moins adaptés à un environnement en constante évolution. Dans cette thèse, nous explorons le domaine du machine learning adaptatif, qui vise précisément à résoudre des problèmes comportant des relations de cause à effet dynamiques. À cet égard, nous avons étudié et proposé des solutions pour évaluer l'efficacité des modèles incrémentaux par batch ou par lot, comparés à ceux incrémentaux par instance, en tenant compte des défis réels de la détection de fraude tels que le déséquilibre des classes et le retard des étiquettes. Par ailleurs, avec la sophistication croissante des solutions ML, il devient courant pour les data scientists d'utiliser des outils d'interprétabilité (comme SHAP ou LIME), bien que la fiabilité de cette approche reste discutable. Nous avons donc exploré dans quelle mesure l'utilisation de modèles Intrinsèquement Interprétables (II) peut constituer une alternative pertinente pour renforcer la confiance dans les systèmes ML appliqués à la lutte contre la fraude. Dans cette optique, nous avons proposé un modèle II basé sur le mécanisme d'attention, qui offre l'avantage de produire des explications stables, en plus d'être à la fois interprétable et performant sur le plan prédictif. Enfin, notre étude sur des données réelles de virements bancaires a permis de confronter les méthodes proposées et existantes aux contraintes du monde industriel, aboutissant à la formulation de recommandations concrètes pour les data scientists et/ou chercheurs travaillant sur ce sujet.

Abstract

Title: Adaptive Learning Applied to Fraud Detection

Keywords: Adaptive learning; Machine learning; Interpretability; Attention Mechanism; Decision Trees Ensemble; Fraud detection

Abstract: Payment fraud, which affects banking institutions, can occur through various channels (e.g., checks, credit cards, bank transfers) and may result in significant financial losses or customer inconvenience, particularly in cases of false alerts. Combating fraud is therefore essential for banks, leading to a continuous cycle of conflict between security teams, who implement detection and blocking algorithms, and fraudsters, who evolve their strategies to bypass these measures. Despite efforts to employ machine learning approaches, the losses caused by fraud remain substantial, amounting to billions of dollars annually. This raises important questions about the suitability of traditional machine learning (ML) models, which are inherently more static and, therefore, potentially less effective in a dynamic environment. In this thesis, we explore the field of adaptive machine learning, which specifically addresses problems involving dynamic cause-and-effect relationships. In this context, we studied and proposed solutions to evaluate the efficiency of batch incremental models compared to instance incremental models, taking into account real-world challenges in fraud detection, such as class imbalance and label delay. Moreover, as ML solutions become increasingly sophisticated, it has become common for data scientists to use interpretability tools (such as SHAP and LIME), even though the reliability of this approach remains questionable. We have thus investigated to what extent Inherently Interpretable (II) models can serve as a viable alternative to enhance trust in ML systems applied to fraud detection. To this end, we proposed an II model based on the attention mechanism, which offers the advantage of producing stable explanations, in addition to being both interpretable and predictively efficient. Finally, our study on real-world bank transfer data allowed us to test the proposed and existing methods against the constraints of the industrial environment, resulting in concrete recommendations for data scientists and/or researchers working on this topic.

Que ton coeur ne soit pas vaniteux à cause de ce que tu connais, prends conseil auprès de l'ignorant comme auprès du savant, car on n'atteint pas les limites de l'art, et il n'existe pas d'artisan qui ait acquis la perfection. Une parole parfaite est plus cachée qu'une pierre verte, on la trouve pourtant auprès des servantes qui travaillent sur la meule.

PtahHotep

Contents

List of Figures

List of Tables

Liste des abréviations

ADASYN Adaptive Synthetic. 39

ADWIN ADaptive WINdowing. 44, 46–48, 84

AI Artificial Intelligence. 37

ARF Adaptive Random Forest. 47–50

AUC Area Under the Curve. 24, 45

AUCPR Area Under the PR Curve. 24, 25, 36, 46, 53, 66, 108, 111–113, 118, 119

AUCROC Area Under the ROC Curve. 24, 25, 36, 45, 46, 66, 92, 113

AWE Accuracy Weighted Ensemble. 47, 48

AXGB Adaptive eXtreme Gradient Boosting. 48

BRF Balanced Random Forest. 52

CART Classification and Regression Trees. 28, 47

CNN Condensed Nearest Neighbor. 39

DDM Drift Detection Method. 44, 119

DT Decision Tree. 28, 29, 40, 42, 47, 57, 70, 111, 112, 115

EBM Explainable Boosting Machine. 27, 38, 39, 57, 81, 90, 106, 108, 119

ENN Edited Nearest Neighbor. 39

ERM Empirical Risk Minimization. 22

FNN FeedForward Neural Network. 29, 34

GAM Generalized Additive Model. 26, 27, 38, 39, 57, 70, 81, 106, 117–119

GBM Gradient Boosting Machines. 29

HAT Hoeffding Adaptive Tree. 46

HD Hellinger Distance. 40, 43

HT Hoeffding Tree. 46, 47, 49

IBAN International Bank Account Number. 101, 104, 105

II Inherently Interpretable. 37, 82, 116

iid independently and identically distributed. 21

LB Leveraging Bagging. 47, 50

LFR Linear Four Rates. 44

LR Logistic Regression or Linear Regression. 26, 28, 57, 70, 111, 115

MHA Multi-Head Attention. 34

ML Machine Learning. 21, 26, 27, 29, 30, 35, 39, 41–43, 51, 52, 101–106, 109, 111, 114, 115, 117–120

MLP MultiLayer Perceptron. 29, 30, 34, 42, 58, 70

NN Neural Network. 29, 36, 37, 41, 58, 64, 68–70, 81, 109, 112, 118

OHE One Hot Encoding. 31, 34, 35

OOB Oversampling based Online Bagging. 49, 50

OOT Out Of Time. 108, 114, 115

Pbr Performance-based retraining. 107, 108, 114–116

PR Precision Recall. 24

RF Random Forest. 29, 41, 47, 58, 112, 115

RNN Recurrent Neural Network. 51

ROC Receiving Operating Characteristic. 23–25

ROSE Robust Online Self-Adjusting Ensemble. 50

SGBT Streaming Gradient Boosted Trees. 49

SHAP SHapley Additive exPlanations. 37, 38, 117

SMOTE Synthetic Minority Oversampling TEchnique. 39, 40, 50

SRA Self-Reinforcement Attention. 60–62

SRP Streaming Random Patches. 48

TabSRA Tabular Self-Reinforcement Attention. 59, 60, 106–108, 117, 118

UOB Undersampling based Online Bagging. 49

VFDT Very Fast Decision Tree. 46

XAI eXplainable Artificial Intelligence. 36, 37, 66

XGBoost eXtreme Gradient Boosting. 29, 48, 49, 52, 58, 106–108, 112, 118

Chapter 1

Introduction

1.1 Fraude aux moyens de paiement

Peut-être considéré comme une fraude tout acte malhonnête (par rapport aux lois ou règlements en vigueur) commis dans l'intention de tromper un individu (ou un groupe d'individus), l'objectif ultime étant de lui soutirer un bien ou des privilèges. Parmi les exemples les plus connus de l'histoire récente, on trouve les fraudes électorales, fiscales ou informatiques. L'adoption croissante des moyens de paiement scripturaux a donné naissance à de nouvelles formes de fraudes, communément appelées fraudes aux moyens de paiement. Celui qui commet l'acte est appelé fraudeur, tandis que celui qui subit l'acte est appelé la victime.

Les moyens de paiement les plus couramment utilisés incluent les cartes bancaires (paiement via des canaux électroniques, paiement sans contact, paiement mobile, ou sur internet), les virements bancaires (instantanés ou non), les chèques et les prélèvements.

Les techniques utilisées par les fraudeurs sont parfois d'une innovation imprévisible et peuvent varier en fonction des moyens de paiement concernés. Selon le *Rapport de l'Observatoire de la sécurité des moyens de paiement 2023 de la Banque de France*, les méthodes les plus fréquentes identifiées sont les suivantes:

- **Carte bancaire** : Usurpation du numéro de carte (via hameçonnage, vol ou perte), manipulation par téléphone ou en ligne pour amener la victime à valider une transaction non authentique.
- **Chèque** : La technique la plus courante consiste à utiliser des chèques volés ou perdus pour régler un paiement ou effectuer un encaissement direct.
- **Virements bancaires** : Tout comme pour les cartes bancaires, la manipulation est souvent utilisée pour inciter la victime à valider un virement frauduleux (par exemple, en se faisant passer pour un conseiller bancaire ou un fournisseur). Il existe également des techniques de détournement de virements, consistant à modifier un ordre

de virement pour récupérer les montants concernés.

Les fraudes aux moyens de paiement peuvent entraîner d'importantes pertes financières pour les clients et les banques (ou, plus généralement, pour les Prestataires de Services de Paiement). Elles peuvent également exposer les banques concernées à des sanctions des régulateurs et à une perte de réputation si des systèmes ou dispositions adéquates de gestion de la fraude ne sont pas mis en place.

Concernant les pertes financières, le *Rapport de l'Observatoire de la sécurité des moyens de paiement 2023 de la Banque de France* estime à environ 1,2 milliard d'euros le montant des fraudes aux moyens de paiement en France pour l'année 2023. Ce montant se répartit comme suit :

- Carte bancaire : 496 millions d'euros (correspondant à un taux de fraude de 0,053 %)
- Chèque : 364 millions d'euros (correspondant à un taux de fraude de 0,078 %)
- Virement : 312 millions d'euros (correspondant à un taux de fraude de 0,001 %)

1.2 Disposition de lutte contre la fraude aux moyens de paiments

Les dispositions généralement mises en oeuvre dans la lutte contre la fraude peuvent être catégorisées en deux étapes: la prévention et la détection de la fraude (Dal Pozzolo [1]). L'objectif de la prévention de la fraude est d'instaurer des lois ou des techniques visant à décourager les intentions ou l'appétence à commettre des fraudes, ou du moins à en limiter les conséquences le cas échéant. Les techniques de détection, quant à elles, interviennent pour bloquer, au bon moment, certaines transactions qui ne respectent pas certaines règles ou qui présentent une forte probabilité d'être frauduleuses.
Parmi les méthodes de prévention, on peut citer :

- **Les sensibilisations régulières des usagers des moyens de paiement** à certains types de fraudes, généralement effectuées par messagerie ou par l'affichage d'alertes lors (ou avant) la finalisation de certaines transactions ;
- **L'imposition par les législateurs aux prestataires de services de paiement (et aux banques)** d'utiliser des systèmes d'authentification forte pour approuver certaines transactions (par exemple, la DSP2 en Europe);
- **Le plafonnement du nombre ou du montant des transactions sur une période donnée.** Par exemple, le montant maximal de retrait peut être limité à 1 000 euros par jour. Dans certains cas, il revient au détenteur du moyen de paiement de fixer des seuils spécifiques ou d'ajouter des outils supplémentaires pour authentifier les transactions ;

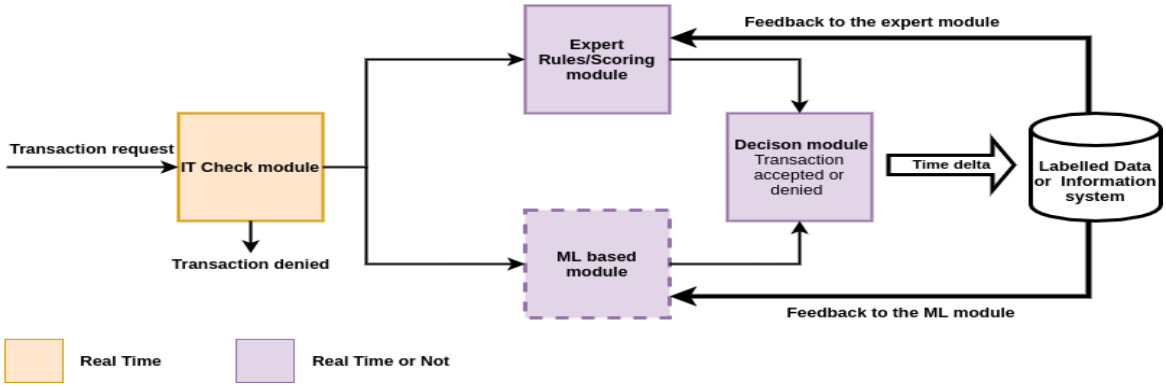


Figure 1.1: *Pipeline typique de gestion de la fraude (non exhaustif). La demande d'opération ou de transaction est d'abord traitée par un module de vérification (par exemple, vérification du code PIN pour les virements bancaires et paiements par carte, vérification d'identité pour les chèques) et refusée si la transaction ne passe pas cette étape. Sinon, un score de fraude est attribué à la demande de transaction à l'aide des modules de Machine Learning (ML) et des Systèmes Experts. Enfin, un module de décision est utilisé pour agréger les scores des modules ML et Expert, et pour accepter ou refuser la transaction en cours en fonction de seuils définis par les besoins métier. Après un certain délai (par exemple, 1 jour ou 1 semaine), la véritable nature de la transaction est supposée connue et utilisée pour améliorer à la fois le module ML et le module d'Expert.*

- **La temporisation de certaines opérations** . Par exemple, certaines banques retardent la validation de l'ajout de nouveaux bénéficiaires pour un virement. Un autre exemple concerne les chèques, dont l'encaissement peut être différé de quelques jours. Il est toutefois important de noter que les clients privilégient généralement des transactions fluides, ce qui peut rendre les institutions bancaires dotées d'un dispositif de temporisation important moins attrayantes.

Dans les dispositifs de détection de fraude, on peut lister les étapes qui ne sont pas forcément disjointes, comme l'indique la Figure 1.1:

- Vérification de la validité de certaines caractéristiques, notamment celles de prévention de la fraude. Par exemple, la vérification de la conformité du code d'authentification pour un paiement par carte bancaire en ligne ou du code PIN lors d'un paiement sur un terminal électronique; la vérification de la date de validité de la carte bancaire; ou encore la vérification de la suffisance des fonds sur le compte lors d'un virement.
- Utilisation d'un module d'expert : ce module repose principalement sur des règles établies par des experts en lutte contre la fraude, basées sur leur expérience.
- Utilisation d'un système basé sur le Machine Learning (ML) : ce système s'appuie sur les données stockées des transactions passées, dont la nature, frauduleuse ou non, est typiquement connue. Les informations extraites du module ML, lorsqu'elles sont intelligibles, servent généralement à enrichir les connaissances des experts et, par conséquent, à améliorer le module Expert.

1.3 Défis pour l'utilisation du Machine Learning dans la détection de la fraude

Il existe plusieurs défis liés à l'utilisation du machine learning pour la détection de la fraude dans le monde réel. Parmi ces défis, les plus connus sont :

1. Les changements dans la distribution des données au fil du temps
2. Le déséquilibre des classes (le nombre de transactions frauduleuses est généralement très faible)
3. La nécessité d'intelligibilité des solutions ML utilisées

Les modèles traditionnels de machine learning sont construits sur l'hypothèse théorique que la distribution des données d'entraînement est identique à celle des données de production ou de test. Malheureusement, la distribution des données peut changer au fil du temps, ce qui peut entraîner une baisse significative de la performance prédictive attendue si cette modification/évolution n'est pas correctement prise en compte. Cette situation est généralement appelée dérive conceptuelle (Gama et al. [2], Lu et al. [3], and Hinder, Vaquet, and Hammer [4]). Concernant la fraude, la dérive conceptuelle peut être due à : (i) un changement dans la stratégie des fraudeurs, qui développent continuellement de nouveaux schémas de fraude pour contourner le modèle de détection de la fraude en production; (ii) un changement dans le comportement transactionnel des détenteurs de moyens de paiement. Par exemple, pendant les périodes de vacances ou de fêtes, les gens peuvent dépenser ou utiliser plus fréquemment leurs moyens de paiement par rapport à leur comportement habituel. Cette situation peut entraîner un changement important dans la distribution des données d'entrée; (iii) un changement dans les législations (par exemple, une augmentation du plafond de paiement), ou encore le développement de nouvelles solutions de paiement sur les moyens de paiement existants (par exemple, Wero récemment proposé pour les virements bancaires en Europe).
Le délai pour obtenir le retour d'information ou connaître la véritable nature des transactions, comme montré dans la Figure 1.1 rend encore plus complexe le problème de changement de distribution. Par exemple, pour les cartes de paiement, la nature frauduleuse d'une transaction ne peut être constatée que si le titulaire de la carte la signale rapidement ou si un analyste de fraude contacte le titulaire de la carte pour vérifier une transaction inhabituelle. Dans le cas contraire, il est courant d'attendre jusqu'à la fin de la période de prescription (1 semaine, 1 mois), où toutes les transactions non déclarées comme frauduleuses sont généralement considérées comme légitimes. Ce délai d'obtention de retour peut donc permettre aux fraudeurs d'utiliser des approches ou des stratégies de fraude actuellement inconnues des systèmes de détection de fraude à plusieurs reprises, avant que les analystes ou experts ne s'en aperçoivent. Il est donc crucial pour un système de détection de fraude basé sur le machine learning de prendre en compte les changements au fil du temps.

Dans les contextes de paiement, le nombre de transactions frauduleuses est heureusement généralement bien plus faible que celui des transactions légitimes (par exemple, moins

de 1 transaction frauduleuse pour 1000 virements bancaires légitimes en France en 2023), une situation connue sous le nom de *déséquilibre des classes*. En général, les modèles de machine learning sont développés sous l'hypothèse d'une représentation égale des classes, de sorte que le déséquilibre des classes peut nuire à la performance ou, du moins, rendre plus complexe la procédure d'évaluation des modèles développés. L'impact du déséquilibre des classes est particulièrement significatif lorsqu'il est associé à un chevauchement des classes (Dal Pozzolo [1] and Vuttipittayamongkol, Elyan, and Petrovski [5]) et à du bruit sur les étiquettes (Krawczyk [6]). Le chevauchement des classes fait référence à la situation où les membres de deux ou plusieurs classes partagent les mêmes espaces ou zones de caractéristiques. Dans la détection de fraude, il s'agit d'une situation courante, car les fraudeurs tentent souvent de réaliser des transactions frauduleuses qui ressemblent autant que possible à des transactions légitimes.
Quant au bruit sur les étiquettes, il met en évidence des situations où certaines transactions réellement frauduleuses peuvent ne pas être signalées dans les systèmes de retour d'information ou dans les données. Cette situation peut affecter la capacité du modèle à détecter des fraudes similaires à l'avenir, d'autant plus que le nombre d'exemples frauduleux est déjà faible. Il est donc essentiel de considérer des modèles de machine learning qui soient relativement robustes face au déséquilibre des classes, au chevauchement des classes et au bruit.

Troisièmement, nous considérons la nécessité d'intelligibilité, c'est-à-dire que les modèles de machine learning ou leurs décisions doivent, dans une certaine mesure, être compréhensibles pour les Data Scientists, les experts en détection de fraude ou les régulateurs. Les Data Scientists peuvent vouloir s'assurer que les caractéristiques apprises par les modèles semblent cohérentes avec la logique métier et sont assez fiables pour passer en environnement de production. Dans ce sens, la simple performance prédictive sur un ensemble de test peut ne pas suffire, mais en addition, il doit être possible de comprendre certaines décisions des modèles: *Pourquoi le modèle a-t-il pris cette décision ? Quelles sont les caractéristiques déterminantes dans les décisions du modèle ?*
Les compréhensions acquises peuvent ensuite aider à enrichir les connaissances des experts en fraude, et ainsi augmenter la confiance dans la collaboration entre le module ML et le module d'expert. Comme la détection de fraude implique parfois des données personnelles, les régulateurs peuvent exiger des modèles non biaisés (par rapport à certains groupes de personnes) de même que la justification des décisions des systèmes de prise de décision automatique (par exemple, le RGPD : article 22 en Europe, AI ACT).
Un autre bénéfice de l'intelligibilité, c'est qu'il est souhaitable de comprendre les changements qui se produisent dans la distribution des données pour une adaptation appropriée au fil du temps.
C'est dans cette optique que la plupart des modèles de machine learning de pointe (en termes de performance prédictive, ce sont des boîtes noires) utilisés en détection de fraude se reposent sur des outils d'interprétabilité *post hoc* tels que SHAP (Lundberg and Lee [7]), LIME (Ribeiro, Singh, and Guestrin [8]) pour expliquer leurs décisions. Pourtant, la confiance accordée à ces méthodes d'explicabilité est parfois remise en question.

Un dernier défi, mais non des moindres, est le temps de réponse du module basé sur le ML. En effet, en fonction des moyens de paiement ou du pipeline de gestion de la fraude,

les modèles ML doivent produire leurs prédictions dans un délai limité (allant de quelques millisecondes à quelques minutes) et le coût de la maintenance de la production ne doit pas dépasser les pertes dues à la fraude. De plus, la plupart des institutions financières commencent à soutenir des exigences écologiques, et peuvent donc privilégier l'utilisation de solutions frugales (lorsque cela est possible). Ces contraintes poussent à trouver un compromis entre la performance du module ML et la consommation des ressources.

1.4 Contexte de la thèse

Cette thèse a été réalisée entre le Groupe BPCE, le Laboratoire d'Informatique de Paris Nord (LIPN) et le DAVID Lab de l'UVSQ-Université Paris Saclay, dans le cadre d'une collaboration CIFRE (Convention Industrielle de Formation par la Recherche) gérée par l'Association Nationale de la Recherche et de la Technologie (ANRT).

Né de la fusion en 2009 des réseaux Banque Populaire et Caisse d'Épargne, le Groupe BPCE figure parmi les principaux acteurs bancaires en France. Fort de son modèle coopératif décentralisé et de son portefeuille d'expertises (par exemple Natixis, Oney, Banque Palatine), il sert plus de 36 millions de clients et traite 20% du marché des paiements français. À ce titre, il fait face, comme ses pairs, à de nombreuses tentatives de fraude, qu'il convient de déjouer.

Au sein du Pôle Data & IA de la Direction Digital&Payments de BPCE, un ensemble de Data Scientists travaille sur des sujets réglementaires et sécuritaires tels que la Lutte Anti-Blanchissement de Capitaux (LAB), la Lutte Contre le Financement du Terrorisme (LCFT) et la Détection de Fraude aux moyens de paiements. Au-delà de l'aspect opérationnel, ces Data Scientists se consacrent également à des Travaux de Recherche et Développement (R&D) pour apporter plus d'innovations dans les dispositifs de BPCE.
Cette thèse, inscrite dans un cadre R&D, revêt un intérêt stratégique, tant pour apporter davantage d'intelligibilité aux dispositifs de détection de fraude que pour améliorer la performance du système, que ce soit pour réduire les pertes financières ou limiter les réclamations des clients à cause des blocages injustifiés (faux positifs).

1.5 Contributions

Dans cette thèse, nous avons abordé les défis discutés dans la Section 1.3, en partant d'un cadre statique où nous avons traité la question de l'intelligibilité. Par la suite, nous avons généralisé l'approche proposée à un environnement dynamique (évolutif) englobant les applications réelles de détection de fraude.

1.5.1 Modèles intrinsèquement interprétables et précis dans un environnement statique

La première contribution de cette thèse consiste à remettre en question la nécessité d'utiliser des outils *post hoc* pour tenter d'obtenir des solutions ML interprétables. Le principal problème des outils d'interprétabilité *post hoc* est la question de la fiabilité. Un nombre croissant d'articles comme Amoukou, Salaün, and Brunel [9], Kumar et al. [10], and Huang and Marques-Silva [11] alertent sur le fait que les outils *post hoc* ne sont pas suffisants, voire ne sont pas adaptés pour exprimer le véritable comportement (même local) d'un modèle, dépendant du type de modèle et de la distribution sous-jacente des données (par exemple, les interactions, les corrélations entre les variables).
Dans le **Chapitre** 3, nous avons exploré la possibilité de contourner le besoin d'interprétabilité *post hoc*, sans sacrifier de manière excessive la performance prédictive. Dans cette optique, nous avons proposé une nouvelle classe de modèles intrinsèquement interprétables, basés sur un mécanisme d'attention, appelés TabSRAs, qui, contrairement aux modèles linéaires ou aux modèles additifs généralisés, ne limitent pas l'ordre des interactions. Au contraire, les contraintes de stabilité locale sont intégrées dans la conception des TabSRAs dans le but de produire des explications similaires pour des points de données d'entrée similaires. Nous avons également proposé des évaluations empiriques approfondies pour mettre en évidence le compromis entre l'interprétabilité et la précision prédictive : les situations où il est avantageux d'utiliser des modèles intrinsèquement interprétables et d'autres cas où il peut être pertinent de considérer l'interprétabilité *post hoc* des modèles en boîte noire en complément des modèles intrinsèquement interprétables, afin de mieux comprendre le phénomène modélisé.

1.5.2 Efficacité des modèles dans des environnements évolutifs

La seconde contribution porte sur le choix du modèle de Machine Learning (ML) dans des situations où les données de production arrivent au fil du temps et peuvent être sujettes à des dérives conceptuelles. Dans ce contexte, nous avons pu constater que les solutions fonctionnant par lots semblent clairement plus adoptées en entreprise pour des problèmes comme la détection de fraude (ou problèmes connexes). Alors même que la quasi-totalité des récents travaux de la littérature se focalisent sur le développement ou l'amélioration des méthodes basées sur les algorithmes en ligne ou incrémentaux par instances (Section 2.2.4), qui de par leur formulation peuvent sembler plus adaptées aux environnements évolutifs. *Est-ce à cause d'une mauvaise connaissance de la littérature par Data Scientists travaillant sur ces genres de sujets? À l'opposé, est-ce parce que les modèles présentés dans la littérature sur l'apprentissage adaptatif, basés en partie sur des solutions incrémentales par instance, ne satisfont pas pleinement ou vraiment les exigences du secteur industriel concerné?*

Nous avons examiné dans le **Chapitre** 4, s'il serait pertinent d'adopter des solutions incrémentales par instance ou en ligne afin de permettre une adaptation rapide aux changements. Il s'avère que dans des cas où i) l'intelligibilité est nécessaire, ii) les étiquettes deviennent disponibles avec un **délai important** et iii) les classes sont fortement déséquilibrées,

comme dans la détection de fraude, les solutions incrémentales par lots demeurent l'option supérieure. De plus, cette dernière option facilite l'intégration de l'humain dans la boucle et l'inspection humaine des résultats produits par le ML, augmentant ainsi probablement la confiance dans les pipelines de détection de fraude basés sur le ML.

1.5.3 Apprentissage automatique dans la détection réaliste de fraude : Application à la fraude au virement bancaire

Étant donné que nous sommes dans le cadre d'un programme de doctorat appliqué (CIFRE), une contribution majeure a été de proposer une solution d'apprentissage automatique réaliste pour améliorer le pipeline de détection de fraude, à la fois en termes de fiabilité et de performance prédictive. Pour des raisons de confidentialité, les chercheurs purement académiques sont rarement informés des conditions de travail réelles (retards dans l'étiquetage, contraintes de validation, contraintes de déploiement...) en détection de fraude. Et même lorsqu'ils en sont conscients, la rareté des données réelles empêche de tester ces conditions. Comme conséquence, des données hypothétiques ou synthétiques sont généralement utilisées pour simuler l'apprentissage automatique dans la détection de fraude.
Grâce à notre observation du processus de validation des modèles ML en conditions réelles et aux données fournies par le Groupe BPCE, nous avons exploré le cas de la détection de fraude par virement bancaire. Nous reconnaissons qu'une seule étude de cas ne suffit pas pour couvrir tous les aspects et défis liés à l'utilisation de l'apprentissage automatique dans la détection de fraude aux paiements. Cependant, nous espérons que notre formalisation du problème (**Chapitre** 5) guidera les chercheurs à intégrer certaines conditions réalistes dans la conception de leurs solutions de détection de fraude et, inversement, aidera les praticiens en apprentissage automatique à améliorer leurs systèmes de détection de fraude.
Les méthodes et outils explorés incluent non seulement les solutions développées dans nos contributions précédemment mentionnées, mais aussi des pratiques déjà connues dans la communauté des Data Scientists.

1.6 Plan

Le reste du manuscrit est organisé comme suit : Dans le Chapitre 2, nous avons introduit l'état de l'art sur le machine learning dans un cadre statique (Section 2.1.2) avec des détails sur l'évaluation lorsque les classes sont très déséquilibrées, sur les modèles classiques ainsi que les modèles plus récents utilisant le mécanisme d'attention. Par la suite, nous avons présenté les approches existantes pour la classification dans un cadre dynamique (Section 2.2) où les données arrivent au fil du temps et peuvent être soumises aux dérives. De même, nous avons évoqué les travaux connus sur l'utilisation de l'apprentissage automatique adaptatif pour la détection de fraude (Section 2.3).

Le chapitre 3 contient notre première contribution sur l'interprétabilité dans un cadre statique où nous avons présenté dans un premier temps le modèle TabSRA, son fonction-

nement à l'aide des illustrations visuelles ainsi que les aspects théoriques motivant l'implémentation (Section 3.3). Dans la Section 3.4, nous avons proposé une étude empirique sur le compromis entre la performance prédictive et l'interprétabilité des modèles, y compris le TabSRA. L'évaluation de chaque modèle ou solution sur les critères comme la performance prédictive, la fiabilité des explications par rapport aux phénomènes modélisés, et la stabilité des explications a permis de formuler des recommandations pratiques pour quelques situations couramment rencontrées sur les données tabulaires.
Le Chapitre 4 présente notre étude sur le choix d'algorithme de machine learning dans un cadre dynamique, où nous avons mis en compétition les modèles incrémentaux par instance avec ceux par lots, avec une attention particulière sur la question du retard des étiquettes. Par la suite, le Chapitre 5 présente une étude de cas sur la fraude au virement bancaire avec la prise en compte des contraintes opérationnelles.

Enfin, le Chapitre 6 conclut le document avec un résumé des éléments abordés dans la Section 6.1. Quelques perspectives pour les travaux futurs du point de vue théorique et applicatif sont exposées dans la Section 6.2.

Introduction

Payment Fraud

Fraud can be defined as any dishonest act (contrary to laws or regulations in force) committed with the intent to deceive an individual (or a group of individuals), with the ultimate aim of extracting a good or privilege. Notable examples in recent history include electoral, tax, or cyber fraud. The growing adoption of scriptural means of payment has given rise to new forms of fraud, commonly referred to as payment fraud. The perpetrator of the act is called the fraudster, while the individual or entity targeted is referred to as the victim.

The most commonly used payment methods include bank cards (electronic channels, contactless payment, mobile payment, or online payment), bank transfers (instantaneous or otherwise), checks, and direct debits.

The techniques employed by fraudsters are sometimes unpredictably innovative and can vary depending on the payment method concerned. According to the *Rapport de l'Observatoire de la sécurité des moyens de paiement 2023 de la Banque de France*, the most frequently identified methods are as follows:

- **Credit cards**: Misappropriation of card numbers (via phishing, theft, or loss), manipulation over the phone or online to coerce the victim into validating an unauthorized transaction.
- **Checks**: The most common technique involves using stolen or lost checks to make payments or perform direct deposits.
- **Bank transfers**: As with bank cards, manipulation is often employed to trick the victim into authorizing a fraudulent transfer (e.g., by impersonating a bank advisor or supplier). There are also redirection techniques, where a transfer order is altered to divert the intended funds.

Payment fraud can lead to significant financial losses for customers and banks (or, more generally, Payment Service Providers). Moreover, it can expose the banks concerned to regulatory penalties and reputational damage if adequate fraud management systems or measures are not implemented.

Regarding financial losses, the *Rapport de l'Observatoire de la sécurité des moyens de paiement 2023 de la Banque de France* estimates the total amount of payment fraud in France for the year 2023 to be approximately 1.2 billion euros. This amount is distributed as follows:

- **Credit cards**: 496 million euros (corresponding to a fraud rate of 0.053%)
- **Checks**: 364 million euros (corresponding to a fraud rate of 0.078%)
- **Bank transfers**: 312 million euros (corresponding to a fraud rate of 0.001%)

Measures to Combat Payment Fraud

The measures generally implemented to combat fraud can be categorized into two steps: fraud prevention and fraud detection (Dal Pozzolo [1]). The objective of fraud prevention is to establish laws or techniques aimed at discouraging the intent or propensity to commit fraud, or at least to limit its consequences when it occurs. Detection techniques, on the other hand, intervene to block certain transactions at the right time that fail to meet specific rules or exhibit a high likelihood of being fraudulent.

Examples of prevention methods include:

- **Regular awareness campaigns for users of payment methods**, typically conducted via messaging or by displaying alerts during (or before) the completion of certain transactions;
- **Mandates from legislators requiring payment service providers (and banks)** to implement strong authentication systems for approving certain transactions (e.g., PSD2 in Europe);
- **Setting limits on the number or value of transactions within a given period.** For instance, the maximum withdrawal amount may be capped at 1,000 euros per day. In some cases, the holder of the payment method can set specific thresholds or add additional tools to authenticate transactions;
- **Introducing delays for certain operations.** For example, some banks delay the approval of new beneficiary additions for transfers. Another example is the delayed clearing of checks by a few days. However, it is important to note that customers typically prefer seamless transactions, which may make banks with significant delays less appealing.

In fraud detection systems, the steps, which are not necessarily distinct, can be listed (as illustrated in Figure 1.1):

- Verifying the validity of certain features, particularly those aimed at fraud prevention. For example, checking the compliance of authentication codes for online card

payments or PIN codes during electronic terminal payments; verifying the validity date of the card; or ensuring sufficient funds are available in the account for a transfer.

- Using an Expert module: This module relies primarily on rules established by fraud prevention experts, based on their experience.
- Using a Machine Learning (ML)-based system: This system leverages stored data from past transactions, where the nature (fraudulent or not) is typically known. The insights extracted from the ML module, when interpretable, are generally used to enrich expert knowledge and, consequently, improve the Expert module.

Challenges for Using Machine Learning in Fraud Detection

There are several challenges associated with using machine learning for fraud detection in real-world scenarios. Among these challenges, the most well-known are:

1. Changes in data distribution over time.
2. Class imbalance (the number of fraudulent transactions is usually very low).
3. The need for interpretability of the ML solutions used.

Traditional machine learning models are built on the theoretical assumption that the distribution of training data is identical to that of production or test data. Unfortunately, the data distribution may change over time, potentially causing a significant drop in predictive performance if this change is not adequately addressed. This situation is commonly referred to as concept drift (Gama et al. [2], Lu et al. [3], and Hinder, Vaquet, and Hammer [4]). In the context of fraud, concept drift may arise from: (i) changes in fraudsters' strategies, as they continuously develop new fraud schemes to bypass the current fraud detection model in production; (ii) changes in the transactional behavior of payment instrument holders. For instance, during holidays or festive periods, people may spend or use their payment instruments more frequently compared to their usual behavior, leading to significant shifts in the input data distribution; (iii) regulatory changes (e.g., increasing payment limits) or the development of new payment solutions for existing instruments (e.g., the recently proposed Wero feature for bank transfers in Europe).

The delay in receiving feedback or knowing the true nature of transactions, as illustrated in Figure 1.1, further complicates the issue of data distribution changes. For example, in the case of payment/credit cards, a transaction's fraudulent nature can only be confirmed if the cardholder quickly reports it or if a fraud analyst contacts the cardholder to verify an unusual transaction. Otherwise, it is common to wait until the end of the prescription period (1 week, 1 month), after which all transactions not reported as fraudulent are generally considered legitimate. This delay in feedback allows fraudsters to exploit currently unknown

fraud approaches or strategies repeatedly before analysts or experts become aware of them. It is therefore crucial for a machine learning-based fraud detection system to account for temporal changes.

In payment contexts, the number of fraudulent transactions is fortunately much lower than that of legitimate transactions (e.g., fewer than 1 fraudulent transaction per 1,000 legitimate bank transfers in France in 2023), a situation known as *class imbalance*. Generally, machine learning models are developed under the assumption of equal class representation, meaning that class imbalance can negatively impact performance or, at the very least, complicate the evaluation process of the developed models. The impact of class imbalance is particularly significant when combined with class overlap (Dal Pozzolo [1] and Vuttipittayamongkol, Elyan, and Petrovski [5]) and label noise (Krawczyk [6]). Class overlap refers to situations where members of two or more classes share the same feature space. In fraud detection, this is common, as fraudsters often attempt to make fraudulent transactions resemble legitimate ones as closely as possible.
Label noise highlights situations where some fraudulent transactions may not be flagged in feedback systems or data. This can affect the model's ability to detect similar frauds in the future, especially given the already limited number of fraudulent examples. It is therefore essential to consider machine learning models that are robust to class imbalance, class overlap, and noise.

Third, we consider the need for interpretability, meaning that machine learning models or their decisions must, to some extent, be understandable for Data Scientists, fraud detection experts, or regulators. Data Scientists may wish to ensure that the features learned by the models are consistent with business logic and reliable enough to be deployed in production environments. In this sense, predictive performance on a test set alone may not suffice; it must also be possible to understand certain model decisions: *Why did the model make this decision? What are the key features driving the model's decisions?*
The insights gained can then help enrich the knowledge of fraud experts, thereby increasing confidence in the collaboration between the ML module and the expert module. Since fraud detection sometimes involves personal data, regulators may require models to be unbiased (with respect to certain groups of people) and capable of justifying the decisions made by automated decision-making systems (e.g., GDPR: Article 22 in Europe, AI ACT).
Another benefit of interpretability is the ability to understand changes occurring in the data distribution for appropriate adaptation over time.
In this regard, most state-of-the-art machine learning models (in terms of predictive performance, which are often black boxes) used in fraud detection rely on *post hoc* interpretability tools such as SHAP (Lundberg and Lee [7]) or LIME (Ribeiro, Singh, and Guestrin [8]) to explain their decisions. However, the trustworthiness of these explanation methods is sometimes questioned.

A final challenge, but no less important, is the response time of the ML-based module. Depending on the payment methods or the fraud management pipeline, ML models must produce predictions within a limited time frame (ranging from a few milliseconds

to a few minutes), and the cost of maintaining production should not exceed the losses caused by fraud. Moreover, most financial institutions are beginning to support ecological requirements, which may encourage the adoption of frugal solutions (when feasible). These constraints necessitate a trade-off between the performance of the ML module and resource consumption.

Thesis Context

This thesis was conducted in collaboration with Groupe BPCE, the Laboratoire d'Informatique de Paris Nord (LIPN), and the DAVID Lab of UVSQ-Université Paris Saclay, as part of a CIFRE (*Convention Industrielle de Formation par la Recherche*) partnership managed by the *Association Nationale de la Recherche et de la Technologie* (ANRT).

Created in 2009 through the merger of the *Banque Populaire* and *Caisse d'Épargne* networks, Groupe BPCE is one of the leading banking players in France. With its decentralized cooperative model and portfolio of expertise (e.g., Natixis, Oney, Banque Palatine...), it serves over 36 million customers and processes 20% of the French payments market. As such, it faces numerous fraud attempts, like its peers, which must be effectively countered.

Within the Data & AI Division of BPCE's Digital & Payments Department, a team of Data Scientists works on regulatory and security-related topics such as Anti-Money Laundering (AML), Counter-Terrorism Financing (CTF), and Payment Fraud Detection. Beyond the operational aspects, these Data Scientists are also dedicated to Research and Development (R&D) efforts to bring more innovation to BPCE's systems.
This thesis, conducted within an R&D framework, has strategic importance, aiming both to enhance the interpretability of fraud detection systems and to improve system performance, whether by reducing financial losses or limiting customer complaints caused by unjustified blocks (false positives).

Contributions

In this thesis, we addressed the challenges discussed in Section 1.6, starting from a static framework where we tackled the issue of interpretability. Subsequently, we generalized the proposed approach to a dynamic (evolving) environment encompassing real-world fraud detection applications.

Inherently Interpretable and Accurate models in a Static Environment

The first contribution of this thesis questions the necessity of using *post hoc* tools to achieve interpretable ML solutions. The main issue with *post hoc* interpretability tools is their reliability. A growing number of studies, such as Amoukou, Salaün, and Brunel [9], Kumar et al. [10], and Huang and Marques-Silva [11], warn that *post hoc* tools are insufficient, or even inadequate, to capture the true (even local) behavior of a model, depending on the type of model and the underlying data distribution (e.g., interactions, correlations among variables).
In **Chapter** 3, we explored the possibility of bypassing the need for *post hoc* interpretability without excessively sacrificing predictive performance. To this end, we proposed a new class of inherently interpretable models based on an attention mechanism, called TabSRAs, which, unlike linear models or generalized additive models, do not constrain the order of interactions. Instead, local stability constraints are integrated into the design of TabSRAs to produce similar explanations for similar input data points. We also proposed extensive empirical evaluations to highlight the trade-offs between interpretability and predictive accuracy: cases where inherently interpretable models are advantageous and other situations where the complementarity of *post hoc* interpretability tools with black-box models may provide deeper insights into the modeled phenomena.

Model Efficiency in Evolving Environments

The second contribution focuses on selecting a Machine Learning (ML) model for scenarios where production data arrives over time and may be subject to concept drift. In this context, we observed that batch incremental solutions are clearly more widely adopted in the industry for problems such as fraud detection (or related issues). This is despite the fact that most recent research efforts focus on developing or improving methods based on online or instance-based incremental algorithms (Section 2.2.4), which, due to their formulation, may appear better suited to evolving environments. *Is this due to a lack of familiarity with the literature among Data Scientists working on these topics? Alternatively, is it because the adaptive learning models presented in the literature, partly based on instance incremental solutions, do not fully or adequately meet the requirements of the concerned industrial sector?*
In **Chapter** 4, we examined whether it would be appropriate to adopt instance incremental or online solutions to enable rapid adaptation to changes. It turns out that in cases where i) interpretability is required, ii) labels become available with a **significant delay**, and iii) classes are highly imbalanced, as in fraud detection, batch incremental solutions remain the superior option. Moreover, this approach facilitates human-in-the-loop integration and allows for human inspection of the results produced by ML, thereby likely increasing trust in ML-based fraud detection pipelines.

Machine Learning in Realistic Fraud Detection: Application to Bank Transfer Fraud

Given that this work is part of an applied doctoral program (CIFRE), a major contribution has been the development of a realistic machine learning solution to enhance the fraud detection pipeline in terms of both reliability and predictive performance. For confidentiality reasons, purely academic researchers are rarely informed about real-world conditions (labeling delays, validation constraints, deployment constraints, etc.) in fraud detection. Even when they are aware, the lack of access to real-world data prevents testing under these conditions. As a result, hypothetical or synthetic data is commonly used to simulate machine learning in fraud detection.
By observing the validation process of ML models in real-world conditions and leveraging data provided by Groupe BPCE, we explored the case of fraud detection in bank transfers. We acknowledge that a single case study cannot cover all aspects and challenges of applying machine learning to payment fraud detection. However, we hope that our problem formalization (**Chapter** 5) will guide researchers to incorporate realistic conditions into the design of their fraud detection solutions and, conversely, help ML practitioners enhance their fraud detection systems.
The methods and tools explored include not only the solutions developed in our previously mentioned contributions but also well-established practices in the Data Science community.

Outline

The remainder of the manuscript is organized as follows: In Chapter 2, we introduce the state of the art in machine learning in a static setting (Section 2.1.2), detailing evaluation methods for highly imbalanced classes, classical models, and more recent models using attention mechanisms. We then present existing approaches for classification in a dynamic environment (Section 2.2) where data arrives over time and may be subject to drift. Furthermore, we discuss known works on the use of adaptive machine learning for fraud detection (Section 2.3).

Chapter 3 contains our first contribution on interpretability in a static environnement, where we first introduce the TabSRA model, its functionality through visual illustrations, and the theoretical aspects motivating its implementation (Section 3.3). In Section 3.4, we present an empirical study of the trade-off between predictive performance and interpretability of models, including TabSRA. The evaluation of each model or solution based on criteria such as predictive performance, the reliability of explanations concerning the modeled phenomena, and the stability of explanations allowed us to formulate practical recommendations for common scenarios encountered in tabular data.

Chapter 4 presents our study on the choice of machine learning algorithms in a dynamic environnement, where we compare instance incremental models with batch-based ones, with particular attention to the issue of delayed labels. Subsequently, Chapter 5 presents a

case study on bank transfer fraud, taking into account operational constraints.

Finally, Chapter 6 concludes the document with a summary of the elements discussed (Section 6.1). Some perspectives for future work, both theoretical and applied, are outlined in Section 6.2.

1.7 Publications & Libraries

Publications

Papers Published in International Journals

- Amekoe, K.M., Azzag, H., Dagdia, Z.C., Jaffre, G. Exploring accuracy and interpretability trade-off in tabular learning with novel attention-based models. Neural Comput & Applic 36, 18583–18611 (2024). https://doi.org/10.1007/s00521-024-10163-9
- Amekoe, K. M., Lebbah, M., Jaffre, G., Azzag, H., & Dagdia, Z. C. (2024). Evaluating the Efficacy of Instance Incremental vs. Batch Learning in Delayed Label and Dynamic Environments. Submitted to Machine Learning journal ECML PKDD 2025. arXiv preprint arXiv:2409.10111.

Papers Published in International Conferences and Workshops

- Amekoe, K. M., Dilmi, M. D., Azzag, H., Dagdia, Z. C., Lebbah, M., & Jaffre, G. (2023, October). TabSRA: An Attention-based Self-Explainable Model for Tabular Learning. In ESANN 2023-European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (pp. 199-204).
- Amekoe, K.M., Azzag, H., Lebbah, M., Dagdia, Z.C., Jaffre, G. (2025). A New Class of Intelligible Models for Tabular Learning. In: Meo, R., Silvestri, F. (eds) Machine Learning and Principles and Practice of Knowledge Discovery in Databases. ECML PKDD 2023. Communications in Computer and Information Science, vol 2135. Springer, Cham. https://doi.org/10.1007/978-3-031-74633-8_16

Libraries and Code

All the codes and configurations used in our experiments (excluding certain configurations applied to confidential fraud data) are made available to the community on the Github[1] account or the Unsupervise[2] project.

[1] https://github.com/anselmeamekoe/
[2] https://github.com/unsupervise/

Additionally, we have proposed an open-source library implementing TabSRA, skorch-tabsra[3], which is the inherently interpretable model introduced in this thesis. This library is enhanced with the Skorch package (Tietz et al. [12]) to facilitate its use for Data Scientists and/or researchers already familiar with the sklearn library (Buitinck et al. [13]).

1.8 Notation

In this document, we will adhere to the notations listed in the Table 1.1.

Table 1.1: *Notations*

Symbol	Meaning
x	a multidimensional random variable defined over a domain $\mathcal{X} \subset \mathbb{R}^p$
y	a unidimensional random variable defined over a domain $\mathcal{Y}$
$\mathbf{x} = (x_1, ..., x_p)$	a realization of x
y	a realization of y
$\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$	a sequence of n realizations of x
$\{y_1, y_2, ..., y_n\}$	a sequence of n realizations of y
$\hat{y}$	an estimation of y
$\mathcal{P}(\mathsf{y})$	probability distribution of a random variable y
$\mathcal{P}(\mathsf{x}, \mathsf{y})$	the joint probability distribution of y and x
$\mathcal{P}(y)$	the probability mass function $\mathcal{P}(\mathsf{y} = y)$, y suppose discrete
$\mathcal{P}(y\|x)$	the conditional distribution function of the target variable y knowing that $\mathsf{x} = \mathbf{x}$
$\hat{\mathcal{P}}(y)$	an estimation of $\mathcal{P}(y)$

[3] https://github.com/anselmeamekoe/pytabsra/

Chapter 2

State Of The Art

In this Chapter, we provide a comprehensive background on machine learning techniques for classification tasks with a particular focus on the binary case. We present a review of the state-of-the-art in learning with tabular data, interpretability in a static environment encompassing the strengths and weaknesses of every approach. Subsequently, we present the same analysis in a dynamic environment where the data distribution may change over time. Finally, we highlight well-known works on the use of adaptive machine learning for fraud detection.

2.1 Classification in Standard Machine Learning

The general purpose of the Machine Learning (ML) is extracting knowledge from data using algorithms widely called *models*.
In the field of ML, classification tasks involve assigning an input (which is a realization of a random vector x) to an output value y (of a response variable y) taking its values in a finite set of categories or classes $\mathcal{Y} = \{c_1, ..., c_K\}$. The case $K = 2$ is called binary classification (e.g., the transaction is fraudulent or genuine, the email is spam or not), that is, $\mathcal{Y} = \{0, 1\}$. Sometimes, an example belonging to class 1 is referred to as *positive* (and negative otherwise).

The classification problem involves learning a model called a classifier (or learner) $h : \mathcal{X} \longrightarrow \mathcal{Y}$ where $\mathcal{X} = (\mathcal{X}_1 \times \mathcal{X}_2 \times .... \times \mathcal{X}_p)$ is the domain of inputs or *observations* or *instances*. In general $\mathcal{X} \subset \mathbb{R}^p$ with p the number of characteristics, attributes/features, or the dimension of the input space.
In classic ML, the model is trained using a finite number or sequence of instances supposed independently and identically distributed (iid) and called *training set*:

$$\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \in (\mathcal{X} \times \mathcal{Y})^n \tag{2.1}$$

Once trained, the model h model can be used to make *prediction* or *inference* on a new instance $\mathbf{x}$:

$$\hat{y} = h(\mathbf{x})$$

The success of the training process is usually measured using the *error*:

$$L_{\mathcal{D}_{train}}(h) = \frac{1}{n}\sum_{i=1}^{n} l(h(\mathbf{x}_i), y_i) \tag{2.2}$$

with l the loss function which is typically the 0-1 loss for the binary classification task defined by:

$$l(\hat{y}, y) = \mathbb{1}_{\hat{y}\neq y} = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

The quantity $L_{\mathcal{D}_{train}}(h)$ is called empirical risk and its minimization over a predefined function space $\mathcal{H}$ is called Empirical Risk Minimization (ERM) (Shalev-Shwartz and Ben-David [14]):

$$\hat{h} = \arg\min_{h\in\mathcal{H}} L_{\mathcal{D}_{train}}(h) \tag{2.4}$$

In general, the success of the learning process is not measured using only the training error but also using the error on an unseen set of observations usually referred to as *test set*. This quantity computed on the test set is generally referred to as the *generalization error* and is expected to represent an estimate of the performance of the learned model in real use (or production setting).

Without losing generality, we will focus for the rest of this Chapter on **the case of two classes** that illustrates well our problem of interest, which is fraud detection.

For many problems, it is convenient to not only predict the class given the input instance but also to get the confidence of the model in predicting the class. In this sense, classifiers are usually designed to estimate the posterior probability $s = \mathcal{P}(\mathsf{y} = 1|\mathsf{x} = \mathbf{x})$ in the case of two classes. One famous of such models is Logistic Regression. The obtained *score* or the probability s can be converted into the class label using a threshold function:

$$g_{tr}(s) = \begin{cases} 1 & \text{if } s > tr \\ 0 & \text{if } s \leq tr \end{cases} \tag{2.5}$$

The threshold is usually chosen to optimize a given evaluation metric or business metric, typically using a test data sometimes referred to as *backtest* data. For some real-world use cases, several thresholds can be used, resulting in several subclasses. The confidence score obtained can also be used to *rank* instances.

For models that can not directly produce estimates of the posterior probability $P(\mathsf{y} = 1|\mathsf{x} = \mathbf{x})$ (e.g., K-Nearest Neighbor classifier), the produced scores can be calibrated using statistical techniques.
In the rest of this Chapter, we will refer to *scoring classifier* or simply classifier, a model that can produce a score if there is no ambiguity.

2.1.1 Evaluation in classification

Classification models are usually evaluated based on the confusion matrix computed on a set of observations, as illustrated by Table 2.1 for the binary classification case.

Well-known metrics that can be computed from the confusion matrix are:

Table 2.1: *Confusion Matrix. TP=True Positives, FP = False Positives, TN= True Negatives, FN=False Negatives. $n = TP+FP+TN+FN$ represents the total number of observations in the evaluation set.*

		Actual class	
		1	0
Assigned	1	TP	FP
class	0	FN	TN

- Accuracy: $Accuracy = \frac{TP+TN}{n}$.
- Precision: $Precision = \frac{TP}{TP+FP}$.
- Recall: $Recall = \frac{TP}{TP+FN}$. The recall is also usually called the True Positive Rate (TPR) or Sensitivity.
- True Negative Rate: $TNR = \frac{TN}{FP+TN}$. This also refer to as $Specificity$.
- False Positive Rate: $FPR = \frac{FP}{FP+TN}$. $FPR = 1 - Specificity$
- F_β measures: $F_\beta = \frac{1+\beta^2}{\beta^2} \frac{Precision \times Recall}{Precision + Recall}$

The accuracy measures how often a machine learning model correctly predicts the actual class and is widely used as a base evaluation measure in classification. However, for imbalanced classification problems, the accuracy can be misleading. For example, for a problem with 1000 instances where 99% (990) are from the class 0 (negative instances), a classifier which always predicts 0 (no matter the input instance) will get a superior or almost perfect accuracy (0.99) compared to another model that perfectly classified all positive examples but misclassified 20 negative instances (0.98 of accuracy). However, TN examples and TP examples may not have the same importance/cost in such class imbalanced situations. To alleviate this problem, specific metrics accounting for the performance on the positive instances such as $Precision$, $Recall$, or F_β measures are usually used (the F_1 being the harmonic mean between the $Precision$ and the $Recall$). The *cost* of correct and incorrect classifications can also be used (in the evaluation) in situations where it is possible to predefine (specify) them (Elkan [15]).

For problems where estimating the confidence score or the posterior probability is important/provided, a threshold function (Equation 2.5) is used before computing the confusion matrix or measures listed above. However, as highlighted previously, in many real-world situations, setting this threshold may involve not only the model Developer or Data Scientist's abilities but also the intervention of business owners. In this situation, Data Scientists may prefer using evaluation measures that do not rely too much on a predefined threshold.
One of the widely used measures for such requirements is the Receiving Operating Characteristic (ROC) (Fawcett [16]). As shown in Figure 2.1, in the ROC space, the TPR is

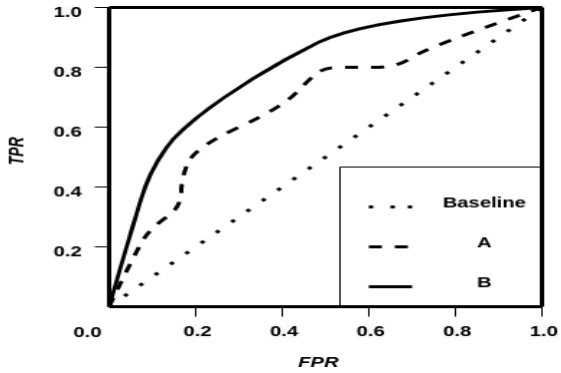


Figure 2.1: *ROC curve*

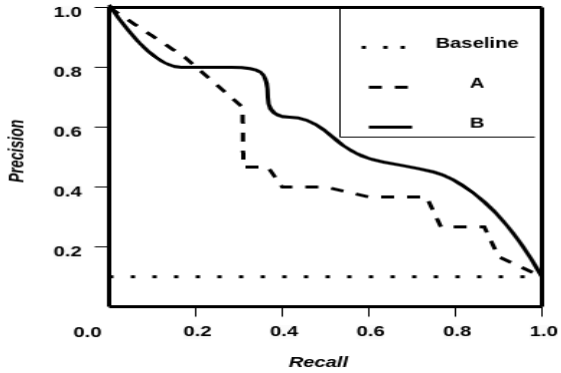


Figure 2.2: *PR curve*

plotted versus the FPR for different thresholds (or equivalently the number of ranked instances). In this space, (0.0, 1.0) represents a perfect achievable classification while (0.0,0.0) and (1.0, 1.0) correspond respectively to setting the threshold such that the model always assigns class 0 and class 1, respectively, regardless of the observations. The diagonal line (i.e., $TPR = FPR$) represents the performance of a random guessing classifier referred to as Baseline in Figure 2.1 and the classifier with the curve B is better than the one with the curve A (the curve B dominates completely the curve A in the ROC space).

In the attempt to compare classifiers, it is more convenient to summarize the ROC curve with a single scalar. The Area Under the Curve (AUC) is one of such scalars ranging from 0.5 (for a random guessing classifier) to 1 (a perfect classifier) in the ROC space.
Given a set of labels and assigned scores by a classifier, the AUCROC simply highlights the probability of ranking a randomly chosen positive instance higher than a randomly chosen negative example. The AUCROC is usually estimated using a trapezoidal rule or the U statistic from Mann-Whitney U test (Fawcett [16]).

Another measure used to assess scoring classifiers is the Area Under the PR Curve (AUCPR) which considers the Precision Recall (PR) curve instead of the ROC. In the PR space, the $Precision$ is plotted versus the $Recall$ as shown in Figure 2.2. The AUCPR is deemed to be more efficient for comparing classifiers than AUCROC for highly imbalanced problems (Davis and Goadrich [17] and Saito and Rehmsmeier [18]).

The problem with the ROC curve is that in situations where the dataset is highly imbalanced, an important change in the false positive rate FP may lead to a small change

in False Positive Rate (FPR) due to the large number of True Negatives (TN). Therefore, comparing in the ROC space may be less discriminatory, especially when the TN predictions/instances have no special added value to the modeling process. On the contrary, when using PR space, the $Precision$ is computed helping to account for both True Positives retrieved (which probably have important value compared to True Negatives) as well as False Positive examples. As a practical example, let's consider an imbalance problem with 100 observations of class 1 (positive examples) and 99,900 negative examples. We want to compare two models A and B for a given threshold α:

- Model A predicts 100 observations to belonging to class 1 when 90 actually do
- Model B predicts 1000 observations to belonging to class 1 when 90 actually do

Hence the:

- $Recall_A(\alpha) = TPR_A(\alpha) = TPR_B(\alpha) = \frac{90}{100} = 0.9$
- $FPR_A(\alpha) = \frac{10}{99900} = 0.0001$; $FPR_B(\alpha) = \frac{910}{99900} = 0.009$
- $Precision_A(\alpha) = \frac{90}{100} = 0.9$; $Precision_B(\alpha) = \frac{90}{1000} = 0.09$

Using the ROC space, the $Recall$ gap is 0 and the FPR gap is = 0.009-0.00001=**0.0089**. Using the PR space, the $Precision$ gap is 0.9-0.009=**0.81** and the $Recall$ gap between is 0. For this example, we can notice that the $Precision$ (hence the PR space) is more discriminatory between the models A and B than the FPR (the ROC space).

Regarding the estimation of the AUCPR, in contrast to the ROC space, the trapezoidal method (the linear interpolation between points) can result in misleading or overly optimistic estimates of performance (Davis and Goadrich [17] and Saito and Rehmsmeier [18]). This is due to the fact that for a fixed $Recall$, the $Precision$ may not be constant (for some problems, the precision may continue to decrease as we increase the threshold even if the recall remains the same). As a consequence, non-linear interpolation solutions are adopted for PR (Saito and Rehmsmeier [18]) and one of the widely used resulting estimators is *Average Precision*:

$$AP = \sum_{i=1}^{n} Precision(i).[Recall(i) - Recall(i-1)] \tag{2.6}$$

where

$$Precision(i) = \frac{TP_i}{i} \tag{2.7}$$

is the fraction representing the number of True Positives among the first i ranked instances and $Recall(i)$ is the corresponding recall.

In practice, both the AUCROC and AUCPR can be computed or used together whenever possible, where the advantage of AUCROC is being insensitive to changes in class distribution (for example, when the proportion of positive examples changes from the train to test data or data sampling is used) and the one of AUCPR is being more discriminatory, therefore useful for model selection in highly imbalanced situations.

2.1.2 Classical Machine learning models

We consider a class of models h which, given the input data point $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$ can be formulated as

$$h(\mathbf{x}) = \sigma^{-1}(f(\mathbf{x})) \tag{2.8}$$

where σ represents the link function (e.g., $\sigma(\mu) = \log(\frac{\mu}{1-\mu})$ for binary classification and $\sigma = Identity$ for regression tasks).

Linear Models

A Logistic Regression or Linear Regression (LR) is a transparent model obtained by a simple linear combination of input features as follows:

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x} \\ &= \beta_0 + \beta_1 x_1 + ... + \beta_i x_i + ... + \beta_p x_p \end{aligned} \tag{2.9}$$

$\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_p)$ is the regression coefficients vector and β_0 is the intercept or bias term. β_i is a global importance measure of the i-th feature while $\beta_i x_i$ is the effect used in the explanation of individual output. LRs have been used for a while as statistical models with well-known open-source implementations such as *Statsmodels* (Seabold and Perktold [19]) for Python users and *glm* (Lüdecke et al. [20]) for R users. Another famous tool is *Scikit-learn* (Pedregosa et al. [21]), the Python module that offers various optimization algorithms and regularization means for fitting linear models in ML paradigms. Regularization methods such as LASSO (Tibshirani [22]) can be very useful in the presence of multicollinearity situations that may impair the fitting of classic statistical LR models as well as their interpretability. Recently, deep learning libraries such as Pytorch (Paszke et al. [23]) and Tensorflow (Abadi et al. [24]) provide Linear or Dense layer options helping to optimize linear models using gradient descent-like methods.
Due to its simplicity, a LR cannot efficiently model problems with non-linear effects (unless the convenient non-linear term is identified and added to the model as a new feature) and feature interactions.

Generalized Additive Models

The general formulation of a Generalized Additive Model (GAM) (Hastie and Tibshirani [25]) is as follows:

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} f_i(x_i) \tag{2.10}$$

where f_i is a shape function that, given the i-th input feature x_i produces the output $f_i(x_i) \in \mathbb{R}$. Analyzing the unidimensional shape functions can help to understand the local and global behavior of GAMs. Linear models are a particular case of GAMs in which

every shape function is linear, that is, $f_i(x_i) = \beta_i x_i$.
Classically, GAMs were used as statistical models where typically spline functions (Wahba [26]) are considered for fitting the shape functions. Recently, some ML implementations are proposed to ease the optimization of the shape functions of GAMs. One of such implementations is the Explainable Boosting Machine (EBM) (Nori et al. [27]), which is arguably one of the most established implementations. EBM uses piecewise constant functions obtained from Decision Trees (that use only one feature per tree) and a boosting mechanism to fit the shape functions. Although the fact that GAMs can learn non-linear patterns, they are inefficient in handling data with interactions. To alleviate this problem of interactions, Lou et al. [28] proposed the use of pairwise interactions in the formulation of GAMs.

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) \tag{2.11}$$

This latter version is sometimes referred to as *Generalized Additive Models plus Interactions* (GA^{2}Ms). In the EBM implementation ([27]), the maximum number of interaction terms f_{ij} is controllable, helping to preserve the intelligible aspect of the overall model.

Adding pairwise terms generally raises a problem of effect identification, that is, should the contribution/importance of a given feature be attributed to the main term or the interaction terms? To solve this ambiguity, techniques such as interaction purification (Lengerich et al. [29]), consisting of using the functional ANOVA to push interaction effects into main effects whenever possible, or heredity constraints (Yang, Zhang, and Sudjianto [30]) are typically used. The heredity constraints suggest that a pairwise interaction term can only be included in the final model if at least one of its parent main effects is "important."
The principal difficulty of effect identification when using GAMs comes from the concurvity problem (Ramsay, Burnett, and Krewski [31] and Siems et al. [32]). The concurvity refers to the situation when the shape functions might be correlated, that is, there exist $c_0 \in \mathbb{R}$ and one-dimensional functions $g_1, ..., g_p$ not all zero such that:

$$c_0 + \sum_{i=1}^{p} g_i(x_i) \approx 0$$

The multicollinearity is therefore a particular case of concurvity when $g_i(x_i) = \alpha_i x_i$. As for linear models, the concurvity impairs seriously the interpretability of GAMs. To alleviate this issue, recent works such Kovács [33] and Siems et al. [32] propose to use feature selection or to penalize the correlation between shape functions during the training.

Due to their formulation, using GAMs with more than 2 orders of interactions can be very tricky or difficult for humans to understand or interpret (for pairwise interactions, for example, it is still possible to visualize 2-way or pairwise interactions heatmaps, but what is doable for 3-way interactions? visualize interactions cubes? What about effects identification?). However, for some problems, it might be convenient to model more than pairwise interactions. As an alternative to GAMs, tree-based or attention mechanism-based methods can be used in such situations.

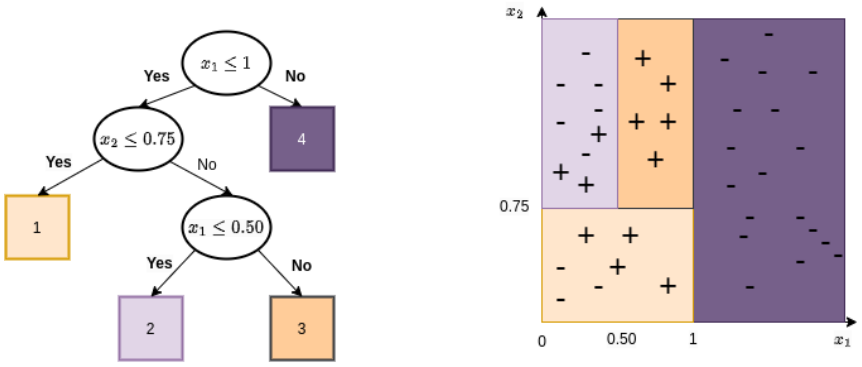


Figure 2.3: *Illustration of binary Decision Tree of depth 3 using two features x_1, x_2 to partition the data space. The total number of leaf nodes is $T = 4$ (ranging from 1 to 4).*

Decision Trees

A Decision Tree (DT) is a model that is obtained by partitioning the training data (space) recursively, imitating a tree-like structure (i.e., if-else rules applied in a root-child node fashion) such that similar instances (with respect to their target values) are grouped together. The splitting is generally stopped when it remains only one observation in a given node or the termination criteria are met as shown by Figure 2.3. This terminal node is referred to as a leaf node. DTs are generally interpretable by design and can produce transparent decisions, especially when the depth of the tree is reasonable. Similar to LRs, the *Scikit-learn* library offers a fast and optimized implementation of the Classification and Regression Trees (CART) (Breiman [34]) which is one of the famous DTs.
Given the input instance $\mathbf{x}$, the output value is:

$$h(\mathbf{x}) = w_{q(\mathbf{x})}, \quad q : \mathbb{R}^p \longrightarrow \mathcal{T} \tag{2.12}$$

where q is the function that maps the input $\mathbf{x}$ to the corresponding leaf node index in $\mathcal{T} = \{1, 2, ..., T\}$ with T being the total number of leaf nodes. $w = (w_1, w_2, ..., w_T)$ represents the corresponding *weights* vector. Once the DT is fitted, the weight w_t is estimated as the proportion of training instances of class 1 falling into the leaf node t for classification problems (and the average target value for regression problems). Contrary to LRs, DTs are able to fit naturally some non-linear relations or interactions among features. However, DTs may quickly overfit the train data and can be unstable to a small shift in the data (especially the overly complex trees); therefore, pruning or regularization (for example, by setting conditions on the minimal number of instances in the leaf nodes during the training) is usually necessary for better generalization. These options are for instance available in the *Scikit-learn* package (Pedregosa et al. [21]). In general, tree ensembles are used to improve the generalization and the predictive performance over a single DT.

Decision Tree Ensemble

Model ensembling (or ensemble methods) consists in aggregating the decision or prediction from different individual models (called base or weak learners) to obtain better predictive

performance or expressiveness. In the decision tree ensemble, the base learner is a DT model. In what follows, we list some well-known decision tree ensembles:

Random Forest. Random Forest (RF) (Breiman [35]) is one of the most popular decision tree ensembles which fits a number of independent (parallel) DTs on various subsamples of the dataset and uses aggregation (known as bagging or bootstrap aggregating) to improve the predictive accuracy and control overfitting. Feature or column subsampling is also used in Random Forest helping to empower the diversity among learned trees. Among the open source implementations is the one provided by in *Scikit-learn* (Pedregosa et al. [21]). Given the input instance $\mathbf{x}$, the output value is:

$$h(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} h_m(\mathbf{x}) \tag{2.13}$$

where M is the total number of trees in the ensemble and $h_m(\mathbf{x})$ is the output of the m^{th} DT classifier as shown by the Equation (2.12).

XGBoost. In eXtreme Gradient Boosting (XGBoost) (Chen and Guestrin [36]), DTs are sequentially combined using a boosting mechanism, that is, the current DT regressor considered as *weak learner* is added to the ensemble to reduce the error (computed using the gradient and hessian) done by the previous or the baseline classifier resulting in an overly *strong learner* (with small bias) at the end of the training. XGBoost is one of the first Gradient Boosting Machines (GBM) that offers a scalable implementation with a regularization possibility making it arguably the most used in real-life use cases and one of the leading predictive performance state-of-the-art in tabular ML competitions. The final output of the XGBoost model given the input $\mathbf{x}$ is:

$$f(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x}) \tag{2.14}$$

where the $f_m(\mathbf{x}) \in \mathbb{R}$ is the output or corresponding weight of the m^{th} DT regressor. Note that for XGBoost, the final weight is converted to a score (for classification tasks) using a *Sigmoid* function as shown by the Equation (2.8).
In general, a decision tree ensemble with an important number/size of learners requires *post hoc* tools for explaining their decisions (they are not interpretable by design).

MultiLayer Perceptron

Also known as FeedForward Neural Network (FNN), MultiLayer Perceptrons (MLPs) are a part of Neural Network (NN) models. Neurons of MLPs are fully connected as shown by Figure 2.4 and are equipped with non-linear functions also referred to as *activation functions*. MLPs are simply a sequence or composition of linear functions followed by non-linear transformations.
That is, given the input $\mathbf{x} \in \mathbb{R}^p$, the final output is:

$$f(\mathbf{x}) = Linear(MLPLayer(....MLPLayer(\mathbf{x}))) \tag{2.15}$$

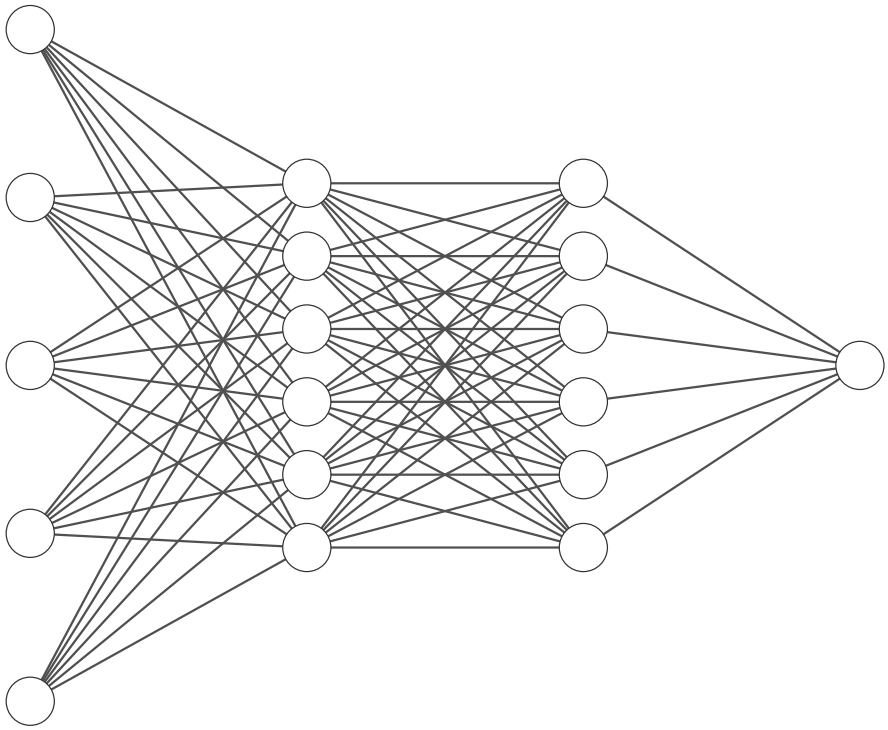


Figure 2.4: *Illustration of two (2) hidden layers MLP of width six (6). The input layer contains $p = 5$ neurons/weights representing the number of features. The output layer is made up of one neuron meaning that the final output is a real number.*

where $MLPLayer(\mathbf{x}) = Activation(Linear(\mathbf{x}))$, $Linear$ is a linear transformation (Equation 2.9) and $Activation$ is the activation function (e.g., $ReLU(\mathbf{x}) = \max\{0, \mathbf{x}\}$).

Thanks to its differentiable property, MLPs are usually optimized using gradient backpropagation (Rumelhart, Hinton, and Williams [37]). Sometimes weight decay (or L_2 penalization) is used for regularization during the training step, and dropout (Srivastava et al. [38]) is used to empower diversity among neurons. Numerical discretization (Gorishniy, Rubachev, and Babenko [39]) can also be used to ease the modeling of long tail distributions with MLPs.

2.1.3 Attention mechanism and tabular data

Attention (for humans) is a cognitive capacity or function to selectively concentrate when and where needed on a specific part of available whole information as highlighted in the review (Niu, Zhong, and Yu [40]). This biological capacity inspired the world of ML and impacted significantly the Deep Learning field by giving birth to the so-called *attention mechanism*. The success in Machine Translation of the self-attention implementation used in the Transformer paper (Vaswani et al. [41]) motivates several researchers in using it for tabular or heterogeneous data (Huang et al. [42], Gorishniy et al. [43], and Somepalli et al. [44]) to compete with tree ensemble models.

In what follows, we will use the machine translation task to provide a general background on the attention mechanism.

Given the input sequence S of symbol representations $[\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_p]$, the goal of machine translation is to generate (one element at a time using a *decoder*) the output sequence $U = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_m]$ typically using an intermediate continuous representation of S noted as $[\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_p]$ and obtained from an *encoder* (Vaswani et al. [41]).
The sequence U can be further converted into a natural language using predefined vocabulary.

For example, S can be the representation of the sentence "I love you" using a OHE in predefined vocabulary or corpus of d worlds or tokens, i.e., $\mathbf{s}_{\text{I}} \in \mathbb{R}^d$, $\mathbf{s}_{\text{love}} \in \mathbb{R}^d$ and $\mathbf{s}_{\text{you}} \in \mathbb{R}^d$. Let's assume that the translation task is to convert this sentence to French.

Table 2.2: *Example of the attention matrix for machine translation*

	I	love	you
Je	0.94	0.04	0.02
t'	0.03	0.05	0.92
aime	0.06	0.86	0.08

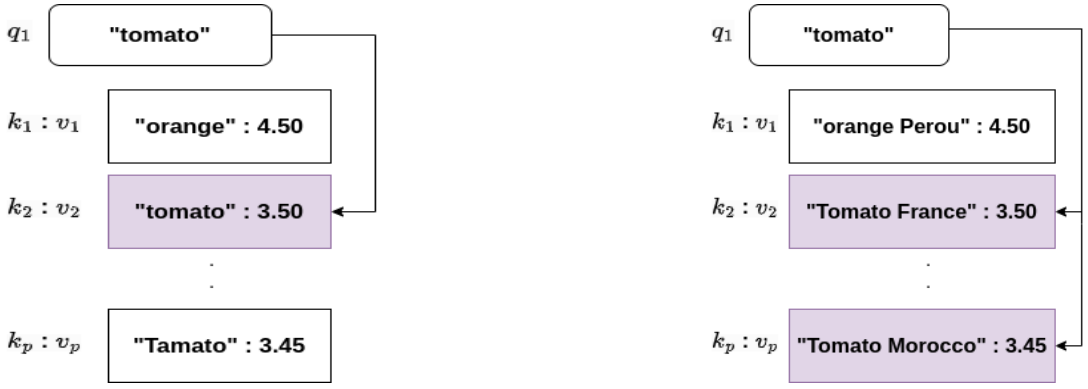
An example of attention visualization is proposed in the Table 2.2. We can see from this example that the model generates the word "Je" when paying attention to the word "I". Similarly, the model focuses more on "you" which is the third input word, to produce "t'" (which is actually the second in the translated sentence). This example shows that the attention mechanism can help the translation model or machine to see beyond the simple order in the input sequence.

In machine translation, the attention mechanism can be between the encoder and decoder, i.e., between the input sequence (or its continuous representations) and the output sequence, as illustrated with Table 2.2. It can also be used to learn continuous representations in both encoders and decoders (Vaswani et al. [41]). The attention matrix is also referred to as the alignment matrix (Bahdanau [45]).

Beyond machine translation or language processing, the attention mechanism was used successfully in computer vision (Dosovitskiy [46]). This is done by first splitting the image into a finite number of patches and considering each patch as a word or token analogously to what is done in language processing.

On the Self-Attention. With further details, the encoder-decoder example illustrated above works as follows: the decoder sends a ***query*** and obtains a response/result also called ***context representation*** (Bahdanau [45]) which is the weighted sum of ***values*** (representations obtained from the encoder). The weights are typically obtained by computing a similarity or alignment between the query and memory ***keys*** which are also provided by the encoder.

The *query*, *keys*, *values* terminology can be explained using the analogy of information retrieval in a dictionary data typically disposed in **key** : **value** structure.
For example, on a supermarket machine, a user searches the word **"tomato"** to get its price



(a) *Hard retrieval* **(b)** *Soft retrieval*

Figure 2.5: *Illustrative analogy between the query, key, value terminology in attention mechanism and information retrieval in dictionaries.*

in (euros) as shown in Figure 2.5.
With the hard retrieval (Figure 2.5a), 3.50 is returned directly as the *value* corresponding to title or *key* (noted k_2) that matches exactly the user's search or *query* (referred to as q_1). For the situation where there is no title that matches exactly the query, the returned result or value can be proportional to the *similarity* between *query* and all known titles (in memory). With this retrieval approach called *soft* (Figure 2.5b), the returned result is $0.5 \times 3.50 + 0.5 \times 3.45 = 3.475$ where 0.5 corresponds to the match or *similarity* score between q_1 and k_2, q_1 and k_p respectively.
In both cases, the result can be formulated as follows :

$$c_1 = \sum_{i=1}^{p} a(q_1, k_i) \times v_i \tag{2.16}$$

where $a(.,.)$ is the function that helps to compare the user's request to the titles available in memory. $a(.,.)$ produces a one hot vector for hard retrieval and a real-valued but **normalized** vector for the soft retrieval.
Generally speaking, the *value* can be a vector (not only a real number) representing a word, text, image, video... Also, people may want to produce results or *representation* for several *queries* simultaneously. In this situation, the vectorization can be very useful.

The particular case where both the queries, keys, and values are obtained from the same input (e.g., sentence, image) is known as *Self-Attention* (Vaswani et al. [41]).
The self-attention was found to be very useful for learning continuous representations over input sequences without suffering too much from the problem of long-range dependencies. For example, the self-Attention can be used for the vectorial representation of each word in the sentence "I love you" which accounts for the similarity between these words or the semantic.

Technically, given the sequence $S = [\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_p] \in \mathbb{R}^{p \times d}$ i.e., $\mathbf{s}_i = (s_i^1, s_i^2, ..., s_i^d)$, a matrix of queries ($Q \in \mathbb{R}^{p \times d_k}$), keys ($K \in \mathbb{R}^{p \times d_k}$) and values ($V \in \mathbb{R}^{p \times d_v}$) are produced

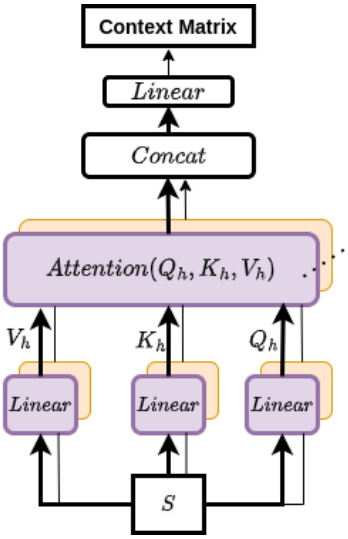


Figure 2.6: *Illustration of the Multi-Head Attention with $H = 2$ heads.*

typically using linear projection functions. These matrices are thereafter used to compute the attention matrix and context representation as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.17}$$

where $A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{p\times p}$ is the attention matrix and $C = Attention(Q, K, V) \in \mathbb{R}^{p\times d_v}$ is the context representation. The dot product between the Q and K is scaled by $\sqrt{d_k}$ to reduce the influence of larger values of the projection dimension d_k. Furthermore, a $softmax$ function is applied to convert the attention weights over each query into probabilities distribution (that sum to one). This attention computation is referred to as *Scaled Dot-Product Attention* (Vaswani et al. [41]).

Multi-Head Attention. Instead of using a single attention computation, the authors in [41] proposed the use of H parallel computation of the attention mechanism of Equation (2.17) as illustrated by Figure 2.6. Each single computation of the context matrix is called *head* and the whole computation mechanism is called *Multi-Head Attention*:

$$MHA(S, H) = Concat(head_1(S), ..., head_H(S))W^O \tag{2.18}$$

where $head_h(S) = Attention(Q_h, K_h, V_h) \in \mathbb{R}^{p\times d_v}$ is the $h-$th context representation obtained from the input sequence S and $W^O \in \mathbb{R}^{Hd_v\times d}$ is typically a linear matrix used to project the new context matrix into the initial dimension i.e., $\mathbb{R}^{p\times d}$.
$Concat$ represents here the concatenation of H matrix of the same shape.
In practice, using several or H attention heads doesn't increase the computation cost as d_k, d_v are chosen such that $d_k = d_v = d//H$ (when using a single head, $d_k = d$). Instead, the use of the Multi-Head Attention eases the parallelization of computations.

Transformer for tabular data. The Multi-Head Attention initially proposed for language processing was seriously investigated for tabular data. In this attempt, given an

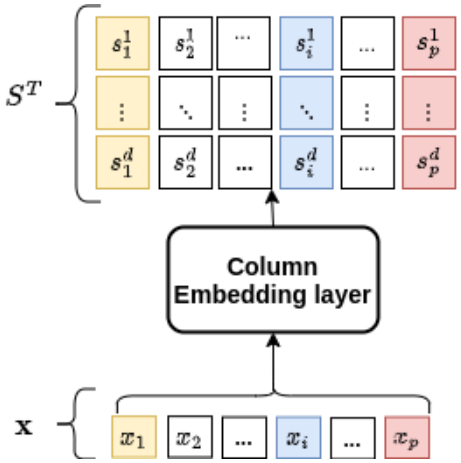


Figure 2.7: *Illustration of a standard process for column embedding or tokenization within the Transformer architecture designed for tabular data. The raw input* $\mathbf{x}$ *generally consists of both numerical and categorical features. The numerical representation* S *is outputted by the column embedding layer. The transformation is usually performed column-wise.*

input observation $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$, each component x_i is **considered as a word or token** and therefore **embedded** to a d dimensional space similarly to text as in Vaswani et al. [41] (An illustration is shown in Figure 2.7).

The obtained embedding S of shape $\mathbb{R}^{p \times d}$ is fed to the Multi-Head Attention layer for learning a contextual representation, as shown by Figure 2.8.
Aside from the MHA sublayer, the Transformer block used in Vaswani et al. [41], Huang et al. [42], Gorishniy et al. [43], and Somepalli et al. [44] contains also a **Position-wise** FeedForward Neural Network (FNN) sublayer and, **normalization and residual connections** are applied to the output of each sublayer before the next sublayer. That is, given the embedding S of the initial input $\mathbf{x}$, the input of the next FNN sublayer is:

$$Z = LayerNorm(S + MHA(S, H)) \tag{2.19}$$

In general, several Transformer blocks (typically $N = 6$ blocks) are used sequentially to learn high level concepts. The final context representation matrix $C^T \in \mathbb{R}^{d \times p}$ (Figure 2.8) outputted by the Transformer blocks is flattened into one dimensional vector and fed to a final Model or classifier (typically a MLP) producing the final output $\hat{y}$.
Sometimes, instead of the flattened version of the context matrix C, a one-dimensional slice (vector) corresponding to a *CLS or classification token* (appended to the output of the column embedding, i.e., S) is used as input for the final classifier as in **FT-Transformer** (Gorishniy et al. [43]).

Regarding the column embedding (Figure 2.7,2.8), the simplest way to encode a categorical feature is to use a One Hot Encoding (OHE). However, in TabTransformer, SAINT, FT-Transformer for example, a more sophisticated embedding is used consisting of projecting the previously obtained One Hot Encoding (OHE) vector into a continuous vector of d

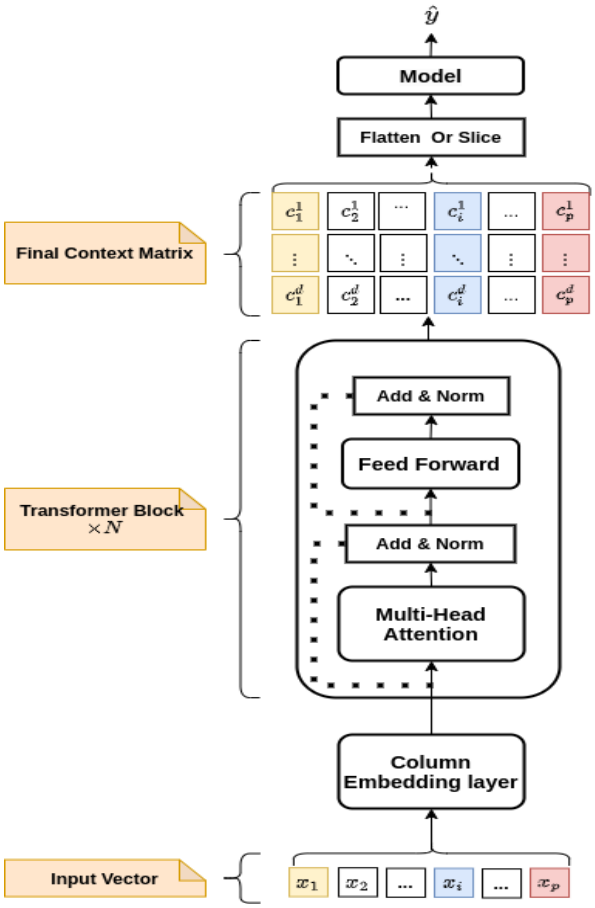


Figure 2.8: *Generic Transformer architecture for tabular data*

elements typically using a linear layer (such implementation is available on Pytorch[1]). For FT-Transformer, numerical features are also embedded into d dimensional vector using a linear layer. In SAINT, a ReLU is used in addition to a linear layer for the embedding of numerical features.

Although treating categorical features as tokens is intuitive (OHE is typically used to transform one-dimensional features into d binary features in traditional ML), it becomes less straightforward when applied to continuous features within tabular data or environments. Consequently, first Transformer implementations for tabular data such as Huang et al. [42] used feature embedding and the transformer block for only categorical features. However, using the tokenization and learning the context representation of numerical (continuous) features in addition to categorical ones tends to improve the expressiveness of the overall model (Somepalli et al. [44] and Gorishniy et al. [43]) even if at the cost of an increase in the number of learnable parameters.

While the attention matrix within the encoder-decoder architecture (Vaswani et al. [41] and Bahdanau [45]), as depicted in Table 2.2, can be utilized for interpretability in machine translation and vision tasks (Dosovitskiy [46]), its application for tabular data remains con-

[1] https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

siderably less direct. That is, for tabular data, what matters is to understand potential interactions and to identify important input features (with respect to target prediction/feature) which are usually one-dimensional real numbers. Unfortunately, visualizing an attention matrix of shape (p, p) or the context representation $C^T \in \mathbb{R}^{d \times p}$, p being the number of features, is not enough or directly informative as an explanation measure especially when the number of Transformer blocks (N) is greater than one (1).

In addition, the use of attention matrix or weights as an explanation measure was questioned even in language processing and vision tasks since they might be uncorrelated to well-known feature importance measures such as gradient-based, saliency measures (Jain and Wallace [47] and Bastings and Filippova [48]). Overall, the use of classical attention weights for explanation purposes requires a clear definition of what an *explanation* is (Wiegreffe and Pinter [49] and Bastings and Filippova [48]).

2.1.4 Predictive performance based state-of-the-art

The debate on the best performing between decision tree ensemble such as Random Forest (Breiman [35]), XGBoost (Chen and Guestrin [36]), LightGBM (Ke et al. [50]), CatBoost (Prokhorenkova et al. [51]) and NNs such as Trompt (Chen et al. [52]), FT-Transformer (Gorishniy et al. [43]) or SAINT (Somepalli et al. [44]), on tabular data is not settled. However, certain meta-features guide the choice of one over the other depending on the problem (Grinsztajn, Oyallon, and Varoquaux [53], Borisov et al. [54], Gorishniy et al. [43], and McElfresh et al. [55]). Regarding NNs, they are usually differentiable, making them easy to use, for example in multimodal settings (e.g., encoding tabular information with text, images, etc.), multitask settings (Agarwal et al. [56]) or pretraining and finetuning context (Arik and Pfister [57], Huang et al. [42], and Somepalli et al. [44]). Tree-based models, on the other hand, are capable of applying abrupt thresholding to features, making them more suitable for handling skewed feature distributions and other forms of dataset irregularities (Grinsztajn, Oyallon, and Varoquaux [53] and McElfresh et al. [55]). Moreover, in contrast to NNs, which generally require a long training time owing to the gradient back-propagation mechanism, decision tree based models are known to be relatively fast to train.

Instead of contrasting NNs and decision tree ensembles, authors such Shwartz-Ziv and Armon [58] suggest combining the strengths of these two classes of models (using, for example, weighting schema or blending). Such a combination helps to unify the diverse concepts learned by models and is usually the winning option in predictive performance-based data challenges or competitions.

2.1.5 On the Interpretability and Trustworthy question

Predictive performance-based state-of-the-art models are typically flexible enough to extract complex patterns or associations (correlations) between features.
Unfortunately, even with *good* $Accuracy$, AUCPR, AUCROC, it might still be difficult for humans to trust the relevancy of learned correlations, the so-called *trustworthy question* which involves asking, for example, if the model is fair (avoids discrimination against certain minority groups) and if it is stable. The eXplainable Artificial Intelligence (XAI) is an

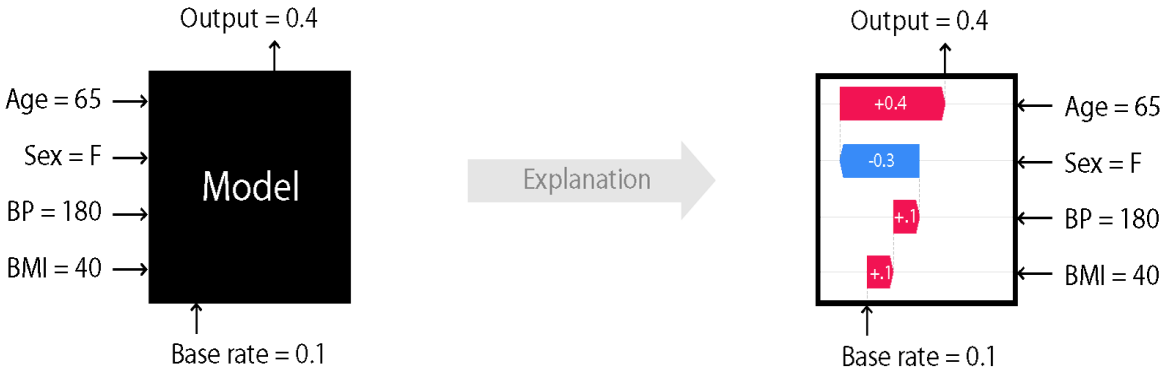


Figure 2.9: *Illustration of SHAP feature attribution. Left : A backbox model outputted a score of 0.4 to an instance having Age=65, Sex=F, BP=180, BMI=40. Right : SHAP is used to explain this decision of the model. According to SHAP, starting from the base score/rate* $\phi_0 = 0.1$*, Age=65 pushes it by* $+0.4$ *while Sex=F has a negative contribution of* -0.3*. The two remaining features have each a contribution of* $+0.1$

emerging field with the goal to bring back trustworthiness in the use of Artificial Intelligence (AI) in decision-making by obtaining human-interpretable models (Ali et al. [59]). The XAI can be categorized into two classes: (i) use *post hoc* interpretability methods/tools to explain or approximate the decision of full-complexity black box models (deep NNs, decision tree ensemble) at least locally; (ii) use directly Inherently Interpretable (II).

Post hoc explanation of full-complexity models: focus on SHAP

The *post hoc* explanation of full-complexity models can be organized into two categories: feature selection and feature attribution (Huang and Marques-Silva [60]). Feature selection methods involve identifying a set (or subset) of features relevant for target prediction among which we have Anchors (Ribeiro, Singh, and Guestrin [61]) and Logic-based abduction (Marques-Silva and Ignatiev [62]). Feature attribution methods remain the most used eXplainable AI (XAI) techniques and involve assigning relative importance to each input feature. SHapley Additive exPlanations (SHAP) (Lundberg and Lee [7]) is a well-known unifying framework for feature attribution, based on the Shapley values.

Given the input $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$, $P = \{1, ..., p\}$ the set of all features, the SHAP[2] feature attribution of the $i-$th feature is:

$$\phi_i = \sum_{E \subseteq P \setminus \{i\}} \frac{|E|!(|P| - |E| - 1)!}{|P|!} [f_{E \cup \{i\}}(\mathbf{x}_{E \cup \{i\}}) - f_E(\mathbf{x}_E)] \tag{2.20}$$

where $\mathbf{x}_E$ is the restriction (values) of the input features in E. $f_{E \cup \{i\}}$ and $f_E(\mathbf{x}_E)$ are two distinct models trained respectively with/without the $i-$th feature. One interesting property of SHAP is additivity; that is, the marginal contribution of each feature sums to the output of the model as illustrated with the Figure 2.9.

[2] https://shap.readthedocs.io/en/latest/

That is:

$$f(x) - \phi_0 = \phi_1 + \phi_2 + ... + \phi_p \tag{2.21}$$

where ϕ_0 is a constant baseline value.

In practical situations, it is infeasible to train a model for all possible subsets $E \subseteq P \backslash \{i\}$; therefore, to compute the exact SHAP value. As a result, an approximation of this quantity is usually estimated. Such an approximation is based on simulating the **absence of features**, a process that, depending on the nature of the data (correlation, interactions between features) and the underlying model, can result in biased approximation or huge computational costs (Chen et al. [63]).

To alleviate this problem, model-specific SHAP has been developed, among which the well-known TreeSHAP solution (Lundberg et al. [64]) was designed for decision tree-based models. However, even the exact SHAP computation was found misleading, for example, in abductive explanations (Huang and Marques-Silva [60]), and non-additive learned functions (Kumar et al. [10]). Overall, the question of the reliability of *post hoc* explanations for instance SHAP with respect to the underlying model remains in the literature (Kumar et al. [10], Huang and Marques-Silva [60], Amoukou, Salaün, and Brunel [9], and Ali et al. [59]).

In general, practitioners are not aware of these aspects of the *post hoc* interpretability tools (probably due to the absence of ground truth explanations for real-world problems), resulting in wrong explanations or promises to the end user. It is therefore worth noting to recall that *post hoc* methods are, in general, heuristics or approximations of the underlying model's decision-making (which is also an approximation of a true phenomenon) and the expectation from them should be *what else can the model tell me?* as stated by Lipton [65].

Inherently interpretable solutions

Although the term "inherently interpretable" may seem more general or vague, an inherently interpretable model is not one whose *explanation* derives from a heuristic (e.g., attention weight aggregation across several layers or heads, SHAP attribution...) but from an explicit hypothesis and/or formulation resulting in transparent (*white-box* or *glass-box*) or understandable (to humans) model decision making (Ali et al. [59]). A classical example of such models is a statistical model in which the data are assumed to have some properties (e.g., for linear models, the features should be independent, have a Gaussian distribution, and have a linear relation with the target feature). In modern machine learning based inherently interpretable models, instead of making assumptions about the distribution of data that are not necessarily verified, some intelligible constraints are put in the model formulations resulting in partially transparent decision making. These models are sometimes referred to as *gray-box* models and are expected to provide a good trade-off between the interpretability and the modeling expressiveness (Ali et al. [59]).

Among these models, we mention Neural Additive Models (NAMs) (Agarwal et al. [56]) which present a neural implementation of the classic GAMs. NODE-G^{2}AM (Chang, Caruana, and Goldenberg [66]) and GAMI-Net (Yang, Zhang, and Sudjianto [30]) improve NAMs by considering pairwise interactions among the features. Another well-known example is the EBM (Section 2.1.2).

To the best of our knowledge, it is difficult to handle higher-order interactions using

GAMs based solutions. To alleviate this problem, we propose TabSRAs (Section 3.3), an attention-based solution that does not require an interaction identification step.

It is important to point out that EBM can capture abrupt changes. The same flexibility is possible for NAMs (using ExU activation) or NODE-G^{2}AM by imitating the discrete thresholding of decision trees (Popov, Morozov, and Babenko [67]). Although this property can help increase the expressiveness or help detect bias in data (Chen et al. [68]), it may result in discontinuities that are sometimes difficult to justify. That is, similar explanations are expected for similar input data, the so-called stability or robustness property (Alvarez-Melis and Jaakkola [69]). With TabSRAs, the stability can be easily controlled using the range of attention coefficients.

2.1.6 On Imbalanced data classification

Using ML for imbalanced data classification is a challenging task as several theoretical training properties or guarantees (which typically consist in maximizing the accuracy or minimizing the error rate as highlighted with Equation 2.1, 2.3) are not obtained under such conditions. Even algorithm evaluation is affected as explained in Section 2.1.1.

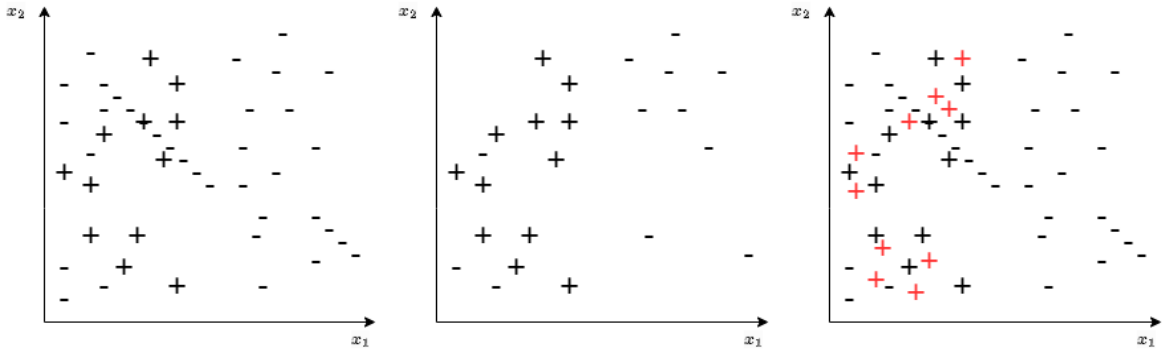
Techniques used to alleviate the impact of class imbalance (during the model learning) are generally categorized in *data level approaches*, *algorithms level approaches* or *hybrid approaches* (Chen et al. [70], Vuttipittayamongkol, Elyan, and Petrovski [5], and Dal Pozzolo [1]).

Data level approaches

These approaches are used to reduce the bias in class distribution or rebalance the classes using *undersampling* (reducing the proportion of negative instances) and/or *oversampling* (increasing the proportion of positive instances in the data distribution). The simplest way to achieve this is to use random sampling.

Among the well-established data rebalancing methods, we note Synthetic Minority Oversampling TEchnique (SMOTE) (Chawla et al. [71]), which consists in using linear interpolation to create new synthetic positive examples. Several variants of SMOTE such as Borderline-SMOTE (Han, Wang, and Mao [72]), DBSMOTE (Bunkhumpornpat, Sinapiromsaran, and Lursinsap [73]), GSMOTE (Douzas and Bacao [74]) were proposed to improve the classic SMOTE using, for instance, geometric or density information. Adaptive Synthetic (ADASYN) is another well-known oversampling technique that exploits the minority class density.

Regarding the undersampling, the most established methods are based on finding Nearest Neighbors of observations. Among such techniques, we cite the Condensed Nearest Neighbor (CNN) (Hart [75]), Edited Nearest Neighbor (ENN) (Wilson [76]) that suggest deleting from the training data negative instances that have different labels from two out of three of their nearest neighbors. In the same sense, Tomek [77] proposes to remove Tomek's link negative instances. Tomek's link exists when two instances from different classes are closest neighbors to each other. A practical implementation of all listed above



(a) *Initial distribution* **(b)** *Undersampled* **(c)** *Oversampled*

Figure 2.10: *Illustration of the resampling. Figure 2.10a shows the initial data distribution, which is imbalanced. In 2.10b, random undersampling is used to reduce some negative instances, resulting in an equal class representation. Oversampling is illustrated with 2.10c. Instead of creating new identical copies of some positive instances as done when using random oversampling, new synthetic positive instances are generated (in red) as typically done in SMOTE.*

techniques is available on the imbalanced-learn[3].

Although these sampling techniques can help to reduce the bias in the data distribution, they also have drawbacks. Oversampling methods usually increase the training computational cost (when producing new positive instances), the potential of overfitting (by biasing the model decision toward the new minority examples that may not be relevant in production or test time or that are noise) as highlighted in (Chen et al. [70]).

Undersampling techniques typically have the advantage of reducing the computational cost (by reducing the training data size) but could lead to information loss, especially with respect to the majority class (Vuttipittayamongkol, Elyan, and Petrovski [5]).

Algorithms level approaches

In this category, the used approach depends intrinsically on the type of algorithm. For decision trees, for example, Cieslak and Chawla [78] used the Hellinger Distance (HD) (which is a categorical and skew insensitive distance) as splitting criteria in the DT instead of the Information Gain. Another technique consists in modifying the misclassification cost in order to favor the minority class, the so-called *cost sensitive learning* (Elkan [15]).

Hybrid approaches

These approaches are usually used with model ensembles, where the sampled version of the initial training data is given to each base learner. Some examples such SMOTE Bagging is focused on the diversity of the ensemble. In EasyEnsemble (Liu, Wu, and Zhou

[3] https://imbalanced-learn.org/stable/under_sampling.html

[79]), a boosting model typically AdaBoost (Freund, Schapire, and Abe [80]) is built on a balanced subsample of initial training data helping to learn diverse aspects of the majority class (therefore to reduce the information loss risk that may occur with the classic under-sampling).

Although several authors emphasized the challenge of class imbalance for classification tasks, others such as Japkowicz and Stephen [81], Garcı–a, Mollineda, and Sánchez [82], Stefanowski [83], and Dal Pozzolo et al. [84] demonstrated that class overlap situations can add more complexity to this class imbalance challenge.

Class overlap and imbalance in classifcation

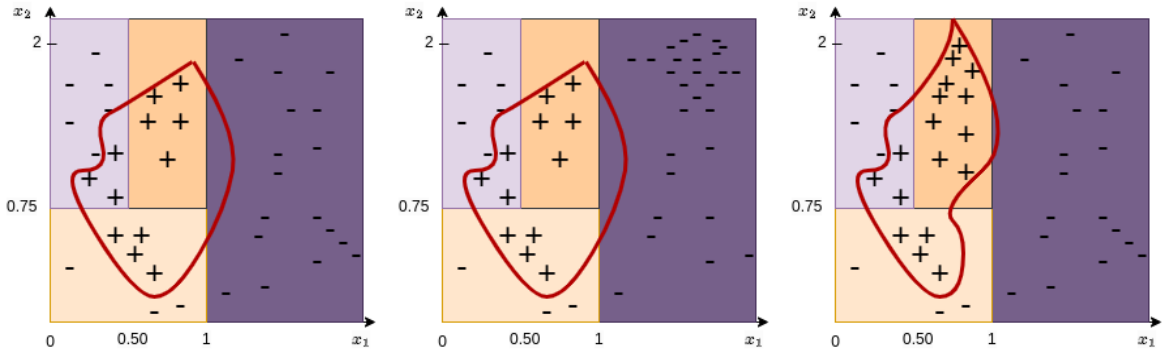
Class overlap refers to the situation where observations from different classes share a common region in the data space (Vuttipittayamongkol, Elyan, and Petrovski [5]). Works such as Garcı–a, Mollineda, and Sánchez [82], Stefanowski [83], and Santos et al. [85] pointed out that class overlap has a more harmful impact on the classifier's performance than class imbalance. That is, when there is no class overlap (Vuttipittayamongkol, Elyan, and Petrovski [5]) or classes are linearly separable (Batista, Prati, and Monard [86]), standard algorithms, even a simple linear model, can be used for imbalanced data classification regardless of the imbalance degree. Moreover, Vuttipittayamongkol, Elyan, and Petrovski [5] demonstrated empirically that the (negative) effect of the class imbalance on the Random Forest (RF) classifier's performance depends on the class overlap as well as the number of positive observations in the data.
Therefore, with sufficient training examples, flexible and non-linear classifiers such as tree-based ensembles or NNs are expected to provide a *good* performance in highly imbalanced and overlapping settings. In addition, using overlap-based sampling techniques, as suggested by Vuttipittayamongkol, Elyan, and Petrovski [5] may help to improve the performance whenever it is computationally feasible.

2.2 Classification in dynamic Environments

In standard ML, the test/production and train data are assumed to be generated by the same process. However, in practice, the test data may *shift* for the train data, a situation usually referred to as ***drift*** (Figure 2.11). In general, the changes in data are observed over time, when typically the production data may arrive continuously and should be processed conveniently. This situation is sometimes called *evolving environments*.

2.2.1 Concept drift and Monitoring in Evolving Environments

In ML, a *concept* refers to a rule that can be deduced or inferred from observed data (from a practical point of view) using a model or, more generally, the true rule or law under the data generating process. In supervised classification, the concept is usually used to highlight



(a) *Initial concept* **(b)** *Virtual drift* **(c)** *Concept drift*

Figure 2.11: *The Figure 2.11a shows an initial observed concept $\mathcal{P}_{t_0}(\mathbf{x}, y)$. Two models are used to estimate decision boundaries $\mathcal{P}_{t_0}(y|\mathbf{x})$ where the red one is a MLP like model and the background is DT like partition. In the Figure 2.11b there is a virtual drift, that is, $\mathcal{P}_{t_0}(\mathbf{x})$ changes but doesn't affect the learned decision boundaries. Figure 2.11c, $\mathcal{P}_{t_0}(\mathbf{x})$ changes requiring the adaptation of the red boundary: this is a concept drift. However, this change doesn't affect the tree-like learn model.*

a link between a set of predictors (or independent features) and a target (or dependent) variable.

There exists a *concept drift* (Gama et al. [87]) between two time points t_0 and t_1 if:

$$\exists \mathsf{x}, \mathcal{P}_{t_0}(\mathsf{x}, \mathsf{y}) \neq \mathcal{P}_{t_1}(\mathsf{x}, \mathsf{y}) \tag{2.22}$$

where $\mathcal{P}_{t_0}(\mathsf{x}, \mathsf{y})$ is the joint probability distribution (between the feature vector and the target variable) at the time t_0. It is important to point out that in practice, this definition depends on the observed or selected samples and not the underlying data generating process as highlighted in Hinder, Vaquet, and Hammer [4].

Using the Bayes rule, the joint probability distribution can be decomposed as follows:

$$\mathcal{P}_{t_0}(\mathsf{x}, \mathsf{y}) = \mathcal{P}_{t_0}(\mathsf{y}|\mathsf{x})\mathcal{P}_{t_0}(\mathsf{x}) = \mathcal{P}_{t_0}(\mathsf{x}|\mathsf{y})\mathcal{P}_{t_0}(\mathsf{y}) \tag{2.23}$$

The Equation (2.23) shows that the change in the joint probability distribution can come from (1) a change in $\mathcal{P}_{t_0}(\mathsf{y}|\mathsf{x})$, (2) a change in $\mathcal{P}_{t_0}(\mathsf{x})$, (3) a change in $\mathcal{P}_{t_0}(\mathsf{x}|\mathsf{y})$, (4) a change in $\mathcal{P}_{t_0}(\mathsf{y})$ or some combinations of the previous.

In supervised classification, the changes that affect the predictive performance or at least the decision boundary are, in general, of interest and require adaptation:

- $\mathcal{P}_{t_0}(\mathsf{y}|\mathsf{x}) \neq \mathcal{P}_{t_1}(\mathsf{y}|\mathsf{x})$. This drift implies a change in the decision boundary and potentially the performance of a learned ML model. In the literature (Žliobaite [88], Gama et al. [2], Lu et al. [3], and Hinder, Vaquet, and Hammer [4]), this drift is refereed to as *real drift*, *concept shift* or simply *concept drift*. The real drift can also be accompanied by a change in $\mathcal{P}_{t_0}(\mathsf{x})$.

- $\mathcal{P}_{t_0}(\mathsf{x}) \neq \mathcal{P}_{t_1}(\mathsf{x})$ and $\mathcal{P}_{t_0}(\mathsf{y}|\mathsf{x}) = \mathcal{P}_{t_1}(\mathsf{y}|\mathsf{x})$. This change in called *virtual shift*, *data shift*, *input shift* or *feature shift*.

The change that may occur in the observed data distribution with respect to one concept can be *sudden/abrupt* (by switching from one concept to another within a short time), *incremental* (consisting of many intermediate concepts before converging to a new relatively stable concept), *gradual* (a new concept gradually replaces an old one over time) (Gama et al. [2] and Lu et al. [3]). Moreover, all known concepts can also *reoccur* after some time of absence.

It is important to point out that observed data in classification, for example, can exhibit several sub-concepts (e.g., a conjunction of several decision rules can result in a positive label) and each concept can be subject to one of the listed types of drift, resulting potentially in multiple drifts in one dataset (Giobergia et al. [89]).

The use of the ML in drifting environments usually follows three steps (Lu et al. [3]): (1) *drift detection* for checking whether drift occurs or not; (2) *drift understanding* to understand instance when (the time), how (the severity) and where (the region affected) the drift occurs; (3) *drift adaptation*.

Drift detection

Drift detection refers to techniques or algorithms called *drift detector* used to check whether a drift occurs at a particular time point or interval (equivalently between two data windows). These drift detectors usually raise alerts to inform a system or humans of changes in monitoring settings.
The drift detection can be split into four stages (Lu et al. [3] and Hinder, Vaquet, and Hammer [90]):

1. Data acquisition/retrieval/selection. This step corresponds to splitting the data streams into old (reference) window and new window (of recent observations). Depending on the goal behind the drift detection, the reference window can be *fixed* (for example the training set of a static algorithm), *sliding* (e.g., data collected in the last two months), growing (e.g., the fixed reference data is completed at each timestamp with identified non-drifted new samples).

2. Computing statistics/descriptors. For example, the mean (or cumulative mean), standard deviation are computed to summarize the distribution. Sometimes, dimension reduction or feature embedding techniques (using kernels) are used in this step (Rabanser, Günnemann, and Lipton [91]).

3. Computing dissimilarity. Distance measures such L_p, Hellinger Distance (HD), Jensen-Shannon, Kullback-Leibler divergence are computed to measure dissimilarity between the descriptors of the old and the new distribution. Under some assumptions, the dissimilarity may have a statistical property, and typically, an important dissimilarity value is a synonym of drift.

4. Normalization. This step is in general optional but may be necessary. For example, the p-value must be corrected or normalized in multiple test situations.

Depending on the used data, descriptors, dissimilarity measure, the drift detection methods can be further categorized in *performance or error based*, *data distribution based* or *multiple hypothesis based* drift detection (Lu et al. [3] and Bayram, Ahmed, and Kassler [92]).
Error based drift detector assumes the label availability and a *significant* change in the model performance typically in the 0-1 loss (Equation 2.3) is a synonym of drift. Among such methods we note Drift Detection Method (DDM) (Gama et al. [93]) and its variants EDDM (Baena-Garcıa et al. [94]), HDDM (Frias-Blanco et al. [95]). Another well-known of such techniques is ADaptive WINdowing (ADWIN) (Bifet and Gavalda [96]).

Although error-based methods remain the most investigated in the literature and the building block of several adaptive algorithms (Gomes et al. [97], Bifet, Holmes, and Pfahringer [98], Domingos and Hulten [99], and Bifet and Gavalda [96]), it has been shown recently that the correlation between the model loss and drift is invalid (Hinder et al. [100]). In addition, these methods may fail in detecting drift affecting subgroups of observations as shown empirically by Giobergia et al. [89].

Data distribution based methods are generally focused on the drift in the input data, that is $\mathcal{P}_{t_0}(\mathbf{x})$. These methods may be computationally more expensive than error based methods, especially for high dimension feature situations. Recent approaches in this category are based on building *virtual binary classifier* where observations are labeled 0 in the old (historical) window, and 1 in the new window. A discriminatory power significantly greater than that of a random classifier is a synonym of a data drift (Hinder, Vaquet, and Hammer [90]). It is important to point out that a change in input data may not imply a decrease in model performance. Inversely, some real drifts might occur in $\mathcal{P}_{t_0}(\mathbf{y}|\mathbf{x})$ affecting potentially the model's performance but not detectable using only data distribution based methods as highlighted by Žliobaite [88].

Multiple hypothesis techniques are based on two previous but use multiple hypotheses testing in parallel or hierarchically for the drift detection. One of such methods is the parallel Linear Four Rates (LFR) (Wang and Abraham [101]) and its hierarchical variant (Yu and Abraham [102]) which consist in monitoring the True Positive, False Positive, True Negative, False Negative rates (Table 2.1).

Recently, the change in the feature attributions provided by the model or the *post hoc* interpretability tools is used for drift detection purposes Haug et al. [103] and Mougan et al. [104]. Using feature attributions (instead of input data) for drift detection can help account for dependencies among features learned by the model and avoid triggering alerts that occur in features that are informative for the models. However, the attribution-based drift detection assumes that the explanations obtained using feature attributions are trustworthy, even the ones coming from *post hoc* tools.

Overall, as highlighted by Hu, Kantardzic, and Sethi [105], there is no free lunch in drift detection; there is no universal best drift detector able to detect all types of drift, even the one using model's error Hinder et al. [100].

Drift understanding

The *when question* in drift understanding makes sense when data streams are handled and monitored using time information. It involves identifying the time point or the time interval where the drift occurs.

Regarding the where question, it helps to know or identify the region in the feature (data) space affected by the drift. This process is sometimes referred to as drift localization (Hinder, Vaquet, and Hammer [90]) and is very helpful when the drift doesn't affect all the data but some subgroups.

The evaluation of the severity of the drift (the *how* question) helps to judge the need for adaptation after a drift. As explained in Lu et al. [3], the quantification of the severity of the drift is less straightforward with error-based methods, as the monitored error might not directly be correlated to the concept drift (Hinder et al. [100]).
The drift understanding is useful for convenient and trustful adaptation of the learning models in evolving environments. However, this topic is still less explored in the literature, arguably due to the fact that state-of-the-art algorithms in learning with data streams are auto-adaptable or instance incremental (Gomes et al. [97], Bifet, Holmes, and Pfahringer [98], Domingos and Hulten [99], Bifet and Gavalda [96], and Gunasekara et al. [106]), and don't necessarily need to explain their numerous detected drifts or changes to human users/operators.

Drift adaptation

Approaches used to adapt learning models to distribution changes can be categorized into *active* and *passive* adaptation (Bayram, Ahmed, and Kassler [92] and Dal Pozzolo [1]).

The active adaptation, also called informed adaptation, refers to the situation where the model adaptation is preceded by thorough drift detection. This supposes that the drift detection is trustworthy. With the passive adaptation, also called *blind* adaptation, the model is constantly updated as soon as new supervised samples (labels) become available, regardless of the fact that a drift is detected or not.

The model adaptation can consist in retraining a new model (this is the case when the drift affects the global data distribution) or retraining partially or adjusting an existing model.

2.2.2 Performance evaluation strategy in dynamic environments

The most common strategy used to assess the predictive performance in dynamic environments is the **prequential** evaluation, also known as the test-then-train strategy. In this approach, each observation is first used for testing (updating the evaluation metric) and then for updating the learning model. In general, prequential evaluation is often utilized alongside a forgetting mechanism (Gama, Sebastiao, and Rodrigues [107, 108]), like a sliding window or a fading factor, to emphasize recent performance. Prequential evaluation is usually applied in settings where labels are available immediately after the observation or after a fixed number of instances, and it is typically applied for additive metrics such as the $Accaracy$.

Regarding imbalanced scoring problems, the Prequential AUC proposed by Brzezinski and Stefanowski [109] uses a sliding window to compute AUCROC for drift detection or the predictive performance evaluation. However, this AUC approximation typically requires a large window size before achieving confident estimation (compared to additive metrics such

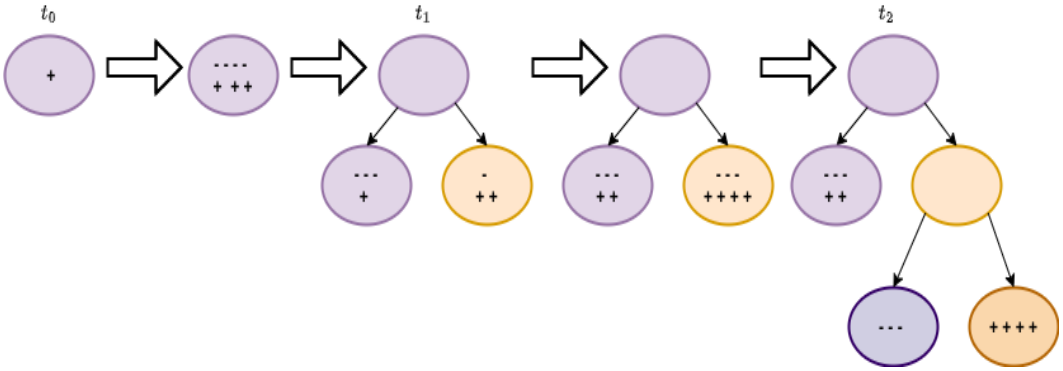


Figure 2.12: *Illustration of the Hoeffding Tree. The tree starts with a single root node (at t_0). When a new instance arrives, statistics are updated and when there are sufficient accumulated instances in a given parent node with respect to the Hoeffding inequality, there is an attempt to split it. In this example, more than 5 instances are required for splitting a parent node and t_1, t_2 highlights the split time. Each arriving instance is used only one time for updating the statistics.*

as error rate) especially for imbalanced data (Agarwal et al. [110]).

For supervised problems, observations may be stored in a buffer or chunk with varying and significant label delays. Once the labels become available, they can be used to test and/or update/retrain the learning model. This strategy is sometimes called **interleaved chunks**. The interleaved chunks approach allows the use of common batch evaluation metrics, such as AUCROC (Bradley [111]) and AUCPR (Boyd, Eng, and Page [112]) as well as commonly additive metrics.

2.2.3 Examples of models in learning in dynamic environments

The learning in dynamic environments is dominated by two classes of models: *instance incremental* and *batch incremental* models (Read et al. [113]). The former models learn from an observation or sample as it arrives incrementally. Regarding batch incremental algorithms or systems, they store observations in batches to update an existing model or retrain from scratch a new one.

Hoeffding Adaptive Tree (HAT)

The Hoeffding Adaptive Tree (HAT) (Bifet and Gavalda [96]) is one of the instance incremental models built on top of the Very Fast Decision Tree (VFDT) or Hoeffding Tree (HT) (Domingos and Hulten [99]). HAT uses the ADWIN (Bifet and Gavalda [96]) detector to monitor every single node of the tree and remove drifted ones.

The HT (Figure 2.12) uses the Hoeffding bound or inequality to choose the number of instances required for splitting nodes. The use of the Hoeffding bound helps to derive some

theoretical guarantees, especially the well-known asymptotic convergence to the classical DTs in a stationary environment (without drift). However, authors such as Rutkowski et al. [114] highlighted that this guarantee does not hold for non-numerical features and splitting measures such as information gain and Gini index, typically used for building classic DTs. The main advantage of the HT is that every instance is used/processed only once; therefore, it is fast enough to handle high-speed data streams.
Python packages such as *River* (Montiel et al. [115]), CapyMOA[4], offer a fast implementation of these algorithms.

Hoeffding tree based ensemble

Similarly to classic Decision Tree (DT), the HT has been used successfully in model ensembling.

Adaptive Random Forest. Adaptive Random Forest (ARF) is an incremental version of the standard RF built on top of the HT. Each tree is equipped with a ADWIN detector with two possible drift thresholds, one for warming and the second for replacing the worst-performing trees. To empower the diversity in the ensemble, the online Bagging (Oza and Russell [116]) is used where typically each new observation (instance) is sampled using the Poisson distribution of parameter 6, similarly to Leveraging Bagging (LB) (Bifet, Holmes, and Pfahringer [98]).
Technically, once a new instance $\mathbf{x}_t$ arrives, each tree h_m in the ARF receives $k_m(\mathbf{x}_t)$ copies of this instance for training, $k_m(\mathbf{x}_t)$ generated randomly according to $Poisson(\lambda = 6)$.
ARF remains one of the famous and best performing among instance incremental algorithms (Gomes et al. [97], Montiel et al. [117], and Gunasekara et al. [106]).

Batch incremental models

Ensemble learning and a performance/error based tracking of base learners dominate the literature of batch incremental algorithms for data streams. One of such models is the Accuracy Weighted Ensemble (AWE) (Wang et al. [118]) in which base classifiers (typically C4.5, CART, SVM, Linear models) are weighted based on their expected test accuracy (estimated using the current training data) over time. The ensemble is made up of the M best performing model on the current data.
Some variants such as Accuracy Updated Ensemble (AUE, AUE2) (Brzeziński and Stefanowski [119] and Brzezinski and Stefanowski [120]) were proposed to improve the AWE using conditional updating and an instance incremental model (e.g., HT) is used as a base learner instead of a batch learner.

Another well-known batch incremental model is Learn++.NSE (incremental learning for Nonstationary Environments) (Elwell and Polikar [121]). In Learn++.NSE, an age weighing schema of learners (using a sinusoidal function) combined with a performance-based weighing on the current training data is used to determine the importance of each learner

[4] https://github.com/adaptive-machine-learning/CapyMOA

in the ensemble.

There is also one notable approach called Adaptive eXtreme Gradient Boosting (AXGB) (Montiel et al. [117]), where new decision tree models are created and appended to the boosted ensemble. More precisely, once the ensemble (the XGBoost model) is full, the oldest member is removed before appending a new one (the push strategy), or older members are directly replaced with newer ones (the replacement strategy).
Another variant of AXGB called $AXGB_A$ where the ensemble is equipped with an ADWIN drift detector was proposed by the same author but was found less efficient compared to the passive counterpart (where the drift or the model error was not explicitly tracked).

2.2.4 Batch incremental versus Instance incremental learning

The utility of rigorously comparing batch incremental and instance incremental approaches is underestimated in the learning literature on evolving data streams. This is arguably due to the preference for automatically updating the model over time and/or avoiding the storage of observations in memory whenever possible (which favors instance incremental solutions). Consequently, attempts to use batch incremental solutions are often biased towards using *small* predefined chunks/batches/windows of instances to retrain or update an old model automatically. Among these approaches, we cite the AWE and AXGB described in the previous Section.

Dynamic or continuous offline optimization of batch learners (using data collected over time, e.g., days, months) alternate with online inference (real-time prediction) is often neglected in the literature when comparing batch and instance incremental learning; however, this strategy remains a widely used approach in real-world production/deployment or *Human in Loop* Machine Learning Operations (MLOps) because it makes human inspection and validation (bias correction, understanding of changes) easier, thereby increasing trustworthiness. Among the studies that compare batch incremental and instance incremental learning in a supervised setting, we note the work by Read et al. [113], from which we summarize the main disadvantages of these two approaches in Table 2.3.
Their experimental result reveals the superior predictive performance (but longer running time) of the instance incremental ensemble namely Leveraging Bagging with Hoeffding Tree as a base learner (LB-HT) (Bifet, Holmes, and Pfahringer [98]) over a AWE using a window of 500 observations and an ensemble of 10 learners maximum. In Montiel et al. [117], the window size was increased to 1,000 and the number of learners to 30 for batch incremental models; however, conclusions remain the same, i.e., the state-of-the-art instance incremental solution ARF (Gomes et al. [97]) showed an overall best predictive accuracy over both the batch solution AXGB proposed by the authors and the AWE strategy. An important point missed in the comparison was hyperparameter optimization. Only a fixed reference configuration of hyperparameters was used when comparing models for all benchmark datasets. Can hyperparameter optimization change the final conclusion? The authors in [117] demonstrated the influence of optimization steps on the performance of the ensemble of XGBoost (called BXGBoost in the paper), which implies an increase of 14% in the average accuracy, becoming the best batch incremental of their benchmarking.
Overall, to the best of our knowledge, the intra-comparison of instance incremental models reveals that ARF (Gomes et al. [97]), Streaming Random Patches (SRP) (Gomes, Read,

Table 2.3: *Comparative Disadvantages of Batch and Instance Incremental Learning*

Learning Approach	Main disadvantages
Batch	• Require deciding the batch/windows size for retraining/updating the model; • Cannot learn the most recent examples until a new batch is complete.
Instance	• Only learns a concept correctly from a large number of examples (e.g., the convergence of HT to a batch-trained Decision Tree is asymptotic and guaranteed only in a stationary environment) [99, 122]; • Has fewer established results than batch learning (e.g., evaluation, bias correction, interpretability).

and Bifet [123]) which is similar to ARF and Streaming Gradient Boosted Trees (SGBT) (Gunasekara et al. [106]) which is the instance incremental version of XGBoost are the predictive performance-based leading state-of-the-art models for learning in dynamic environments.

We decided to reconsider the debate of the winning approach between the batch and instance incremental solutions. In this sense, we designed an evaluation in accordance with real-world (class imbalance, label delay) production scenarios and showed that batch incremental solutions such as XGBoost can have a superior predictive ability with a thorough optimization step (Chapter 4).

2.2.5 Class imbalance in dynamic environments

The class imbalance is an additional challenge for the learning in dynamic environments. The imbalance ratio (between the negative and positive classes) can be fixed or dynamic over time (Aguiar, Krawczyk, and Cano [124]) i.e., $\mathcal{P}_{t_0}(y) \neq \mathcal{P}_{t_1}(y)$. The later situation refers to as *prior (probability) shift* can cause a problem of miscalibration of an already calibrated model (Dal Pozzolo et al. [84]).

The techniques used to handle imbalanced data in dynamic environments are, in general, based on the ones used for static settings (described in Section 2.1.6) (Aguiar, Krawczyk, and Cano [124]). The resampling techniques Ferreira et al. [125] or the cost-sensitive learning Loezer et al. [126] were successfully applied to the ARF model for imbalanced data streams (Aguiar, Krawczyk, and Cano [124]). The resampling typically consists in modifying the generated number by the $Poisson$ distribution (used in the Online Bagging Oza and Russell [116]) by taking into account the imbalance ratio. Another well-known such approach is Undersampling based Online Bagging (UOB) and Oversampling based Online

Bagging (OOB) (Wang, Minku, and Yao [127, 128]). More recently Robust Online Self-Adjusting Ensemble (ROSE) was proposed by Cano and Krawczyk [129] and used a self-adjustment of the Online Bagging parameter. ROSE was found empirically competitive in situations where the imbalance ratio is dynamic, especially for multi-class classification tasks.
Overall, for a recent survey on the imbalanced learning with online or instance incremental algorithms, we kindly invite the interested reader to Aguiar, Krawczyk, and Cano [124] from which we note that there is no single best way to handle class imbalance in dynamic environments.

Regarding batch incremental algorithms, one of the well-known solutions is the modification of the Learn++.NSE (Elwell and Polikar [121]) using SMOTE (the resulting model is called Learn++.CDS) or using a class-specific weighted error metric instead of the classic error metric which is not suited for imbalanced datasets (Learn++.NIE) (Ditzler and Polikar [130]).

The propagation of old batches/chunks was found useful by Dal Pozzolo et al. [131] in an empirical study on a fraud detection dataset. However, this option may increase the need for data storage and the training time.

2.2.6 Label delay in learning in dynamic environments

When learning from data streams, there are three main possibilities regarding label availability (Gomes et al. [97]): (i) immediate, (ii) delayed with a finite time interval, and (iii) delayed indefinitely. There are also settings where all these possibilities can occur simultaneously (Gomes et al. [132]).

Most studies in the supervised paradigm assume immediate availability; however, in many real-world tasks, the label may arrive with an important delay. The (finite) delay mechanism can be *deterministic*, meaning fixed for every value of the feature vector, or *(semi) stochastic*, meaning it follows an unknown probability distribution (Plasse and Adams [133]).
An example of a semi-stochastic case is fraud detection, in which the labels of fraudulent transactions are quickly revealed, whereas those of genuine transactions are mostly revealed after a fixed prescription period. Plasse and Adams [133] proposed an adaptive Linear Discriminant Analysis classifier combined with a weighing scheme to handle the labeling delay using a real-world credit scoring dataset, but their analysis was limited to only linear models, and no specific evaluation procedure was proposed. In Gomes et al. [97], the influence of label delay was investigated using a fixed delay mechanism of 1,000 instances, and the authors concluded that this resulted in an important performance drop for the ARF and LB models. Unfortunately, the comparison did not include batch incremental algorithms, and while stochastic delay appears to be the most common delay mechanism in real-world problems, it was not investigated.

Grzenda, Gomes, and Bifet [134] designed a delayed evaluation framework, called *continuous re-evaluation*, where the goal is to assess the system's or algorithm's capability to refine its predictions over time before the delayed label arrives. Although this framework is quite interesting for flight data, it is infeasible for many problems (e.g., fraud detection and online credit scoring) to request the model to make predictions several times for the

same instance.

In Chapter 4, we propose a framework focused on the first time (the first request for) prediction evaluation. Additionally, the evaluation includes the comparison of instance and batch incremental learning in delayed dynamic environments, which, to the best of our knowledge, is still missing in the literature.

2.2.7 Interpretability in learning in dynamic environments

Although interpretability in dynamic environments is still in its early stages, the two approaches used in static settings (described in Section 2.1.5) can also be adapted to stream or online settings. Regarding *post hoc* solutions, we note for example iSAGE (Muschalik et al. [135]) which offers an incremental version of the Shapley Additive Global Importance (SAGE) (Covert, Lundberg, and Lee [136]) tool.
Unlike static settings, interpretability also requires understanding changes that occur in the model behaviors or, at least, in its *post hoc* explanations caused by incremental model updates (Haug, Tramountani, and Kasneci [137]). Unfortunately, these approaches have to contend with the reliability issue that *post hoc* methods suffer from in the static settings (as highlighted in section 2.1.5). Therefore, authors such as Haug, Tramountani, and Kasneci [137] favor the use of inherently interpretable models for evolving data stream problems, where the more interpretable to humans is likely to change less over time (Haug, Broelemann, and Kasneci [122]).

Our experimental results (Chapter 4) reveal that state-of-the-art batch incremental inherently interpretable models, besides being more interpretable (their architecture or weights change less frequently compared to instance incremental counterparts, and therefore easier for human tracking and understanding), can have competitive or even superior predictive performance compared to their instance incremental counterparts.

2.3 Adaptive Machine Learning for fraud detection

The use of Machine Learning (ML) for payment fraud detection requires the availability and access to transactional information, that is, the historical transactions for the initial modeling and the current ones for the scoring. Depending on the type of transaction and legislation, the requester profile or identification, localization, and time information can also be used. These information, which typically are sequences, can be exploited by the means of *sequence modeling* using, for example, Recurrent Neural Network (RNN) Zheng [138] and Branco et al. [139] or by applying feature aggregations in conjunction with classical ML models (Dal Pozzolo et al. [131] and Lucas and Jurgovsky [140]).

The usual aggregation functions are the $average$, min, max over a time sliding window, and the time difference information. One main advantage of the feature aggregation solution (over the sequence modeling) is the possibility of having interpretable decision-making (especially when the features are carefully computed). The methods and results presented in this thesis are based on the aggregation of raw information into an observation of p features.

The modeling approaches with aggregated features can be supervised i.e., exploiting the label (fraud or not) during the training or unsupervised.

2.3.1 Supervised approaches and adaptive fraud detection

The supervised learning applied to the Credit Card Fraud remains the most investigated approach in the literature (Dal Pozzolo et al. [131], Yeşilkanat et al. [141], Bayram, Köroğlu, and Gönen [142], Adebayo et al. [143], and Cherif et al. [144]). In the supervised settings, some past fraudulent transactions are collected (observations are labeled) and the goal is to prevent from these frauds or similar types. Although most of the works are based on proposing a solution (for example, for the class imbalance, overlapping) with a static train-test data splitting, authors such Yeşilkanat et al. [141] and Bayram, Köroğlu, and Gönen [142] made a step forward by accounting for a potential change/drift that may occur in the data distribution over time. In Yeşilkanat et al. [141], equal number of fraudulent and genuine cards are used over a sliding window to train the supervised model which was nothing else than XGBoost. Bayram, Köroğlu, and Gönen [142] investigated the use of an incremental version of XGBoost, where a new base learner (tree) is added to the initial ensemble every day using all transactions coming from fraudulent cards. However, the label delay is not taken into account or at least the authors didn't explain how they handle it.
One remarkable work regarding the adaptive ML is the Ph.D. thesis by Dal Pozzolo [1]. More importantly, the authors proposed an alert feedback framework for handling the delay in the label availability. Their solution was based on an ensemble of Balanced Random Forest (BRF) (Chen, Liaw, Breiman, et al. [145]) built on a batch of observations collected over days.

Our work aims to enrich the literature by examining seriously the efficiency of famous online or instance incremental methods that came after Dal Pozzolo [1] (which was based on the use of batch incremental solutions). In addition, we take into account the trustworthy constraints in the use of ML by examining the possible trade-off between the predictive accuracy and the interpretability. Finally, we propose an applicative example on Bank Transfer Fraud (including the modeling, the working conditions, and requirements) which has some similarities with Credit Card Fraud but also some particularities.

2.3.2 Unsupervised approaches and adaptive fraud detection

The unsupervised learning models are used in fraud detection by assuming that fraudulent activities are outliers or anomalies (Ingole et al. [146], Cui, Yan, and Wang [147], and Lucas et al. [148]). For example, an important deviation in the user's behavior with respect to himself or the rest of the users is considered as a potential fraudulent activity.

An important advantage of unsupervised learning is that it does not require information about past fraud (labels); therefore, it is relatively easy to implement and adapt, especially in dynamic environments. However, for some types of fraud, fraudsters may mimic the behavior of the authentic user, resulting in an important difficulty for anomaly detection methods. As a consequence, the supervised approach usually reveals significant superiority over unsupervised ones in fraud detection when labeled information is available (Thimonier et al.

[149]). In addition,Thimonier et al. [149] found that the well-tuned tree ensemble model also captures the majority of frauds detected by the unsupervised methods, challenging, therefore, the benefit of using a combination of anomaly detection and supervised models as suggested by Li et al. [150]. Carcillo et al. [151] investigated the *best of both words* where the features extracted from an unsupervised model are used in addition to initial features as input to the supervised model. The added value was not significant unless with the k-means feature which improved the AUCPR metric (but was detrimental for the top precision metric Equation 2.7).

Overall, an unsupervised approach can be very useful for monitoring the model and drift detection, especially when labels become available with an important delay (Pinto, Sampaio, and Bizarro [152]).

Chapter 3

Exploring Accuracy and Interpretability trade-off in Tabular Learning with Novel Attention-Based Models

Apart from high accuracy, what interests many researchers and practitioners in real-life tabular learning problems (e.g., fraud detection, credit scoring) is uncovering hidden patterns in the data and/or providing meaningful justification of decisions made by machine learning models. In this concern, an important question arises: *should one use inherently interpretable models or explain full-complexity models such as XGBoost, Random Forest with post hoc tools?* Opting for the second choice is typically supported by the accuracy metric, but it is not always evident that the performance gap is sufficiently significant, especially considering the current trend of accurate and inherently interpretable models, as well as accounting for other real-life evaluation metrics such as faithfulness, stability, and computational cost of explanations. In this Chapter, we show through benchmarking on 45 datasets that the relative accuracy loss is less than 4% on average when using intelligible models such as Explainable Boosting Machine (EBM). Furthermore, we propose TabSRAs, a new attention-based class of accurate tabular learning models with inherent intelligibility and demonstrate both theoretically and empirically that the instantiation called TabSRALinear is a viable option for (1) generating stable or robust explanations, and (2) incorporating human knowledge during the training phase.

3.1 Introduction

Since the promising results of the Transformer architecture on machine translation tasks (Vaswani et al. [41]), deep learning models continue to provide impressive performance, for example, in language modeling or computer vision. Convinced by the utility of the attention mechanism (used in the Transformer architecture), especially when modeling contextual

information, many efforts have been made to use it in order to match or compete with the accuracy of boosted tree models such as XGBoost (Chen and Guestrin [36]) in tabular modeling (Somepalli et al. [44], Kossen et al. [153], Huang et al. [42], and Gorishniy et al. [43]).

However, the models listed above typically use complex computation mechanisms or a large number of trees, making direct human inspections difficult. On the other hand, interpretability is usually (i) required by regulators in real-world applications (e.g., GDPR: Article 22 in Europe) and (ii) desired if the goal is to discover hidden patterns in the data (e.g., in fraud detection) or to ensure that the model does not learn a bias that may lead to significant drift in production. Therefore, recent studies, such as Lundberg and Lee [7], Ribeiro, Singh, and Guestrin [8], and Lundberg et al. [64], have focused on developing *post hoc* methods to explain, at least locally, the predictions of full-complexity or black box models. Unfortunately, although these methods provide interesting properties, they are sometimes based on some computational mechanisms (e.g., exact Shapley value computation) or hypotheses (e.g., independence between features) that are difficult to achieve in practice, leading to biased explanations (Amoukou, Salaün, and Brunel [9] and Kumar et al. [10]).

Still discussing interpretability, Rudin [154] provides a technical reason why an interpretable model might exist among the set of accurate models in any domain and encourages researchers to move toward finding this solution, especially for high-risk domains. As a result, a bench of inherently interpretable models (Nori et al. [27], Agarwal et al. [56], and Chang, Caruana, and Goldenberg [66]) have been recently implemented and provide superior accuracy compared to classical statistical models (e.g., linear models). Some natural questions arise: *How much accuracy are we actually sacrificing when using an inherently interpretable model instead of a fully complex one?* In light of this question, we first show that the relative performance gap is less than 4%, considering the recent tabular learning benchmark of 45 datasets (59 tasks) introduced by Grinsztajn, Oyallon, and Varoquaux [53]. Moreover, we show that accounting for some real-life metrics such as faithfulness, stability, and computational cost of explanations tends to favor the choice of inherently interpretable models. Second, we propose TabSRAs, an attention-based inherently interpretable solution, and demonstrate its superiority in terms of the stability of explanations and flexibility of incorporating human knowledge compared to existing solutions. Overall, the key contribution of this Chapter is to examine the trade-off between accuracy and interpretability in tabular learning by questioning the value of using *post hoc* tools to explain blackbox models in an era where accurate, inherently interpretable models are available. To achieve this:

1. We propose a simple, intuitive attention based inherently interpretable models called TabSRAs (Section 3.3). We show how to incorporate some human knowledge during its training and demonstrate through real-world datasets its superiority in terms of stability and flexibility for human knowledge incorporation (Section 3.4.4).

2. We thoroughly evaluate and compare state-of-the-art inherently interpretable models and their full-complexity counterparts in terms of their relative predictive performance (Section 3.4.2).

3. We further compare baseline models based on some practical metrics (faithfulness and stability) and demonstrate the usefulness of inherently interpretable models (Section 3.4.2 and 3.4.2).

4. We provide some recommendations on the choice of a tabular learning model in settings where interpretability is a key consideration (Section 3.5).

3.2 Existing interpretable solutions for tabular data problems

In this Section, we list existing interpretable solutions for tabular data investigated in this study.

3.2.1 Inherently interpretable models

Among inherently interpretable models, we consider the well-known solutions already described in Section 2.1.2:

- **Linear models.** (LR);
- **EBM_S**: Explainable Boosting Machine machine (EBM) without interaction terms as part of GAMs;
- **EBM**: Explainable Boosting Machine machine with pairwise interaction terms;
- **Decision Trees** (DT).

3.2.2 Full-complexity models combined with Post hoc tools

We consider in this study the *post hoc* interpretability for accuracy-based state-of-the-art models (Grinsztajn, Oyallon, and Varoquaux [53], Borisov et al. [54], Gorishniy et al. [43], and McElfresh et al. [55]). We split these models into two categories: Decision tree ensemble and Neural Nets models .

Decision tree ensemble

Generally, a decision tree ensemble with an important number/size of learners requires *post hoc* tools when explaining their decisions.
Owing to the node-path structure, decision tree ensembles can be easily combined with some model-specific *post hoc* interpretability tools. For example, TreeSHAP (Lundberg et al. [64]) is a well-suited and computationally efficient SHAP explanation framework for tree-based models. Another framework, Xreason (Ignatiev et al. [155]), aims at reasoning (formally) regarding explanations for tree ensembles with acceptable scalability.
In our evaluation, we consider:

- **Random Forest** (RF);
- **XGBoost** (XGBoost);
- **CatBoost** (Prokhorenkova et al. [51]). Similarly to XGBoost, CatBoost is also based on the boosting mechanism. In CatBoost, the *ordered boosting* strategy is utilized to address the target leakage issue that can arise in the original implementation of XGBoost or LightGBM. Additionally, CatBoost's implementation incorporates a built-in method for handling categorical and text features. In recent studies, such as Chen et al. [52] and McElfresh et al. [55], CatBoost has been shown to provide slightly better predictive performance compared to XGBoost. Therefore, we have included it in our predictive performance evaluation

Neural Nets.

NNs (especially deep NNs) usually require *post hoc* techniques for explaining their decisions. Among these tools, LIME (Ribeiro, Singh, and Guestrin [8]) and KernelSHAP (Lundberg and Lee [7]) are model-agnostic.
DeepSHAP is a Shapley value based specific tool for NNs, inspired by DeepLIFT (Shrikumar, Greenside, and Kundaje [156]). In the theoretical design of these tools, the features are assumed to be independent, and the learned functions are assumed to be linear, at least locally. These assumptions are generally difficult to meet in real-life tabular settings; consequently, these solutions are generally less reliable than their tree-based counterparts.

Based on the results from previous studies (Grinsztajn, Oyallon, and Varoquaux [53], Borisov et al. [54], Gorishniy et al. [43], and McElfresh et al. [55]), we consider in our study the following NNs designed specifically for tabular data (described in Section 2.1.2 and 2.1.3):

- **MultiLayer Perceptron** (MLP). It is a full complexity architecture that can model nonlinear effects and interactions. It somehow provides us the achievable accuracy by shallow and differentiable architectures.
- **ResNet**. ResNet is similar to MLP but includes skip connections (Gorishniy et al. [43] and Grinsztajn, Oyallon, and Varoquaux [53]).
- **FT-Transformer** (Gorishniy et al. [43]). FT-Transformer combines a Feature Tokenizer module and the classical Transformer block (Vaswani et al. [41]) resulting in superior accuracy over the classical MLP (Gorishniy, Rubachev, Khrulkov, and Babenko [43] and Grinsztajn, Oyallon, and Varoquaux [53]).
- **SAINT** (Somepalli et al. [44]). In SAINT, the attention mechanism (Vaswani et al. [41]) is used in the contextual embedding of features. In addition, batch inter-sample attention is used to get better representations.

In the following Section, we will introduce the proposed attention-based solution as a part of inherently interpretable models.

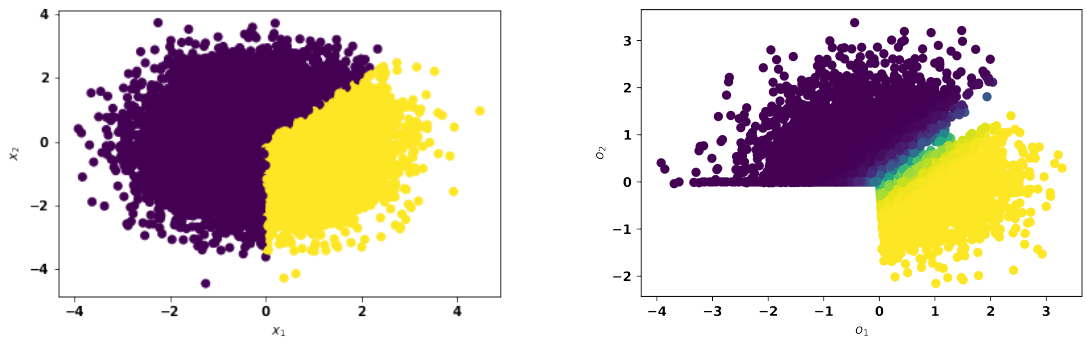
3.3 TabSRAs

In this Section, we describe TabSRAs, our proposed inherently interpretable solution.

Motivational example.
We consider the following function:

$$F_1(\mathbf{x}) = 5x_1 - 5x_2\mathbb{1}_{x_1>0} \quad \text{and} \quad y = \{\mathbb{1}_{p>0.5} \quad \text{with} \quad p = 1/(1+e^{-F_1(\mathbf{x})})\} \tag{3.1}$$

As simple as it may seem, this function cannot be directly modeled with a linear model due to the term $\mathbb{1}_{x_1>0}$, which forces x_2 to have no effect on the output when $x_1 < 0$. Shallow decision trees also are not suitable for this problem due to the linear term $5x_1 - 5x_2$ as shown by the Figure 3.1a which highlights the original data distribution, with the yellow color indicating the class of interest.
Is it possible to produce an intelligible intermediate representation which takes into account interactions among features and ease the use of simple models such as linear models or decision trees?



(a) *Raw data* **(b)** *Intermediate representation*

Figure 3.1: *Motivational example for TabSRAs. Illustration of the reinforcement process on 7500 synthetic data points with 0 mean, unit variance Gaussian distribution. The yellow color is used for the class of interest.*

The Figure 3.1b shows a possible such representation obtained by multiplying the raw inputs with learned coefficients (the green color there represents a linear decision boundary to separate the two classes). That is, through multiplication, values of x_2 can be significantly reduced, for instance, to 0 when needed (i.e., $o_2 \sim 0$ when $x_1 < 0$, $x_2 < 0$), which makes the classes easy to separate with the downstream model (e.g. a simple linear or logistic function). Moreover, this representation can help to understand the global behavior of the overall model, where it is confident in predicting class 1 (in yellow color) and where it is less confident, as highlighted by the green color (Figure 3.1b). That is what Tabular Self-Reinforcement Attention (TabSRA) models are about.

3.3.1 TabSRAs Architecture

TabSRAs (Figure 3.2) are based on the Self-Reinforcement Attention (SRA) mechanism. In a nutshell, given the raw input $\mathbf{x} \in \mathbb{R}^p$, the SRA block (Figure 3.3) denoted as a function $a(.) : \mathbb{R}^p \longrightarrow \mathbb{R}^p$ produces an attention vector $\mathbf{a} = (a_1, ..., a_i, ...a_p)$ which is further used to produce a reinforced input $\mathbf{o} = (o_1, ..., o_i, ..., o_p)$ as follows:

$$\mathbf{o} = \mathbf{a} \odot \mathbf{x} \tag{3.2}$$

where $\odot$ is the element-wise multiplication.

The learned reinforced vector $\mathbf{o}$ represents a new feature basis, where each component is guided by the raw input, that is, $o_i = a_i x_i$, helping to maintain the semantics of each dimension. In this space, some components may be shrunk, for instance, to zero using the attention weights $a_i \geq 0$. In other words, the attention vector acts like a soft instance-wise feature selector or mask. We provide some visual illustrations of how raw data are reinforced in Section 3.3.2.

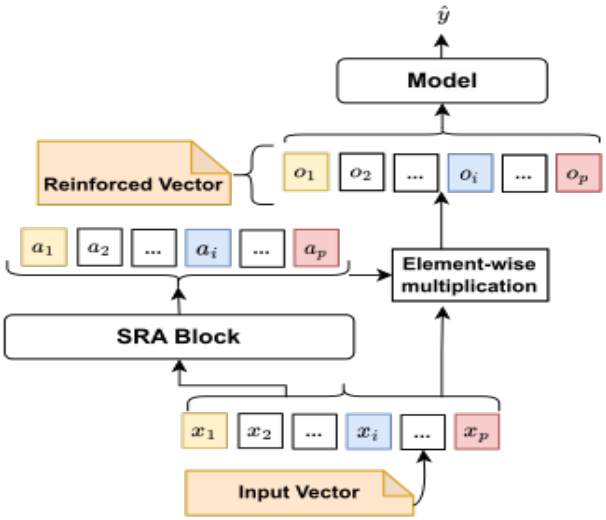


Figure 3.2: *TabSRAs architecture. The attention vector* $\mathbf{a} = (a_1, ..., a_p) \in \mathbb{R}^p$ *provided by the SRA block is used to produce a reinforced vector* $\mathbf{o} = (o_1, ..., o_p) \in \mathbb{R}^p$.

Finally, this reinforced vector is aggregated using a highly **transparent** downstream model (e.g., linear models, decision trees, or rules) to produce the final output. Because we are interested in end-to-end training and differentiable architecture, in this work, we consider a linear downstream model resulting in the so-called TabSRALinear architecture (Section 3.3.3).

3.3.2 SRA block

In the SRA block, the input vector $\mathbf{x} = (x_1, ...x_i, ..., x_p) \in \mathbb{R}^p$, is encoded into p keys in $K = [\mathbf{k}_1, \mathbf{k}_2, ..., \mathbf{k}_i, ..., \mathbf{k}_p]^T$ with $\mathbf{k}_i = (k_i^1, ..., k_i^{d_k}) \in \mathbb{R}^{d_k}$ using a key encoder and queries matrix $Q = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_i, ..., \mathbf{q}_p]^T$ with $\mathbf{q}_i = (q_i^1, ..., q_i^{d_k}) \in \mathbb{R}^{d_k}$ using a query encoder (Figure 3.3). Furthermore, with a Sigmoid activation function, all elements k_i^j of K (resp.

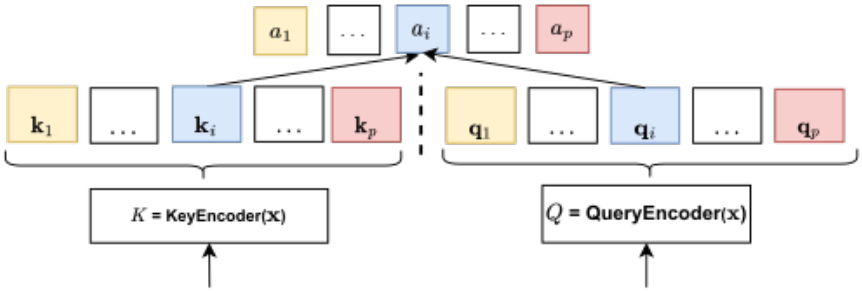


Figure 3.3: *SRA Block. The $KeyEncoder$ (resp. $QueryEncoder$) produces directly p keys (resp. queries)*

q_i^j of Q) are scalar numbers bounded in $[0, 1]$.

The keys in K are compared to the queries Q component by component using the scalar product as in Vaswani et al. [41]. This allows quantifying the alignment of different transformations of the same input calculating the attention weights $\mathbf{a} = (a_1, .., a_i, ..., a_p)$ as follows :

$$a_i = \frac{\mathbf{q}_i \cdot \mathbf{k}_i}{d_k} \quad \text{for} \quad i \in 1, \cdots, p \tag{3.3}$$

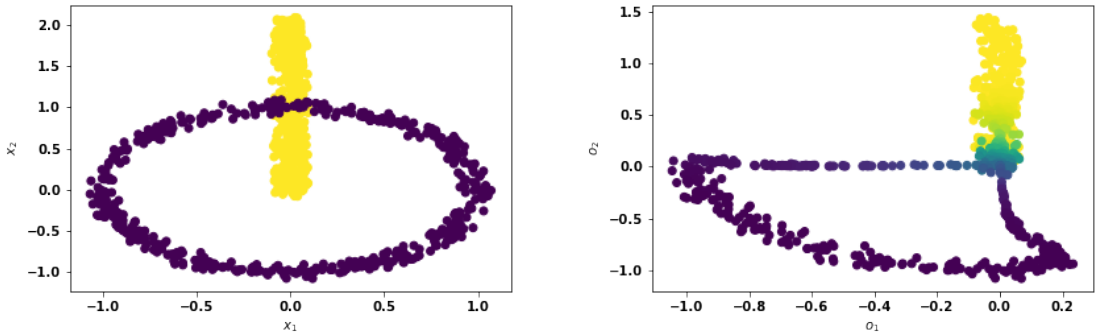
We further use the scaling by d_k in order to reduce the magnitude of the dot-product and to obtain dimension-free attention coefficients $a_i \in [0, 1]$. Some important differences between the proposed SRA block and the vanilla Transformer block (Somepalli et al. [44], Kossen et al. [153], Huang et al. [42], Gorishniy et al. [43], and Vaswani et al. [41]) as described in Section 2.1.3 are summarized in Table 3.1.

Table 3.1: *Some differences between the SRA block and the classical Transformer block. $K \in \mathbb{R}^{p \times d_k}$ the matrix of keys and $Q \in \mathbb{R}^{p \times d_k}$ the matrix of queries*

Point	Classical Transformer block	SRA block
Attention weight	$A = softmax(\frac{QK^T}{\sqrt{d_k}}) \in \mathbb{R}^{p \times p}$	$\mathbf{a} = \frac{\sum_{d_k} Q \odot K}{d_k} \in \mathbb{R}^p$
Value encoding	Yes	No
Additional processing (residual connection, LayerNorm)	Yes	No

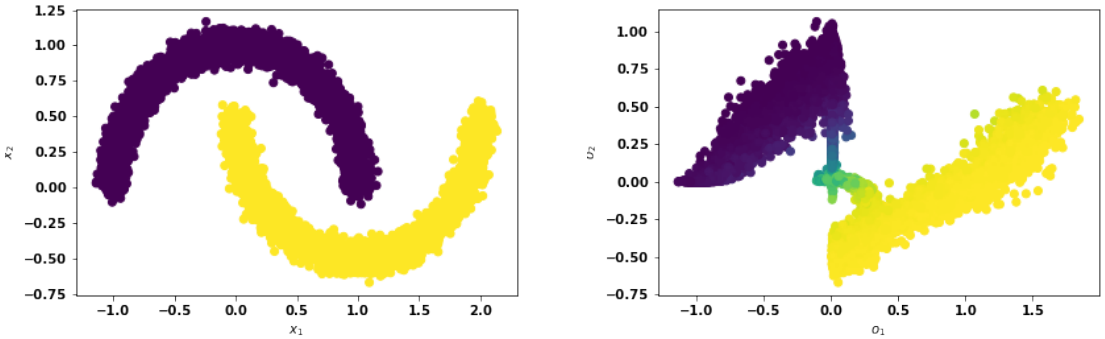
How raw data are reinforced using the SRA block.
To illustrate how the raw data is reinforced in practice using the SRA block, we use a 2D toy dataset with the objective of facilitating the visualization.

We consider the 2D chainLink (Ultsch [157]), the Noisy two moon as depicted in Figure 3.4 and Figure 3.5.



(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure 3.4: *Illustration of the reinforcement process with the ChainLink 2D: 1000 data points.*



(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure 3.5: *Illustration of the reinforcement process with the Noisy two moons: 10000 data points.*

By applying SRA coefficients to this dataset, we acquired a new data representation that enables the easy separation of classes, as shown in Figure 3.4b. Even without knowledge of the true data-generating process, it is apparent that all observations have been moved strategically so that a simple rule can effectively isolate nearly all yellow observations of interest. Please refer to Appendix A.2.2 for additional visualizations.

3.3.3 TabSRALinear: SRA Block and Linear downstream model

We investigate in this work a linear combination of the reinforced features (Equation 3.2) resulting in the additive and differentiable TabSRA model (Figure 3.2). This instantiation

called TabSRALinear can be formalized as follows:

$$\begin{aligned}\sigma(\hat{y}) &= \beta_0 + \boldsymbol{\beta} \cdot \mathbf{o} \\ &= \beta_0 + \beta_1 o_1 + ... + \beta_i o_i + ... + \beta_p o_p \\ &= \beta_0 + \beta_1 a_1 x_1 + ... + \beta_i a_i x_i + ... + \beta_p a_p x_p\end{aligned} \tag{3.4}$$

$\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_p)$ is the linear regression coefficient vector, β_0 the bias term, and $\mathbf{a} = (a_1, .., a_i, ..., a_p)$ the attention weights.

Interpretability with TabSRALinear. Using linear models to model data or phenomena that exhibit feature interactions results in internal conflicts between input components, poor accuracy, or even misleading interpretations. Considering these conflicts, the attention weight vector $\mathbf{a}$ may enhance or reduce some components (of the input vector) at strategic and specific positions depending on the context (or the whole information in $\mathbf{x}$), resulting in an internal equilibrium. Therefore, it is natural to interpret:

- a_i as the correction that the feature x_i received from other features or itself (due to the interactions) before influencing the output. On the instance level, $a_i = 0$ corresponds to the particular case where x_i has no effect on the output.
- $\beta_i a_i x_i$ as the contribution (the prediction importance) of the feature x_i to the output.

Furthermore, visualizing or computing the gradient $\beta_i a_i x_i$ vs. x_j will help to understand the global contribution of the feature x_i as well as interactions (for $i \neq j$).
Considering the important question of simulating the absence of features that arises with some well-known XAI tools (Chen et al. [63]) as highlighted in Section 2.1.5, it is worth nothing to point out that for TabSRALinear, the absence of the $i-$th feature corresponds to the case where $x_i = 0$. In addition, owing to interactions, an input feature can take a zero value (i.e., $x_i \approx 0$) but still influence the output through other features.

Human knowledge incorporation in TabSRALinear. For some problems (e.g., credit scoring), it is crucial to incorporate domain expert knowledge such as positive effects or monotonic constraints with respect to some features during or after training.

In TabSRALinear, the regression coefficient β_i (Equation 3.4) controls the overall "sense" (positive or not) of the effect of the $i-$th feature on the output. Therefore, a positive (resp. negative) sense constraint (knowledge) is added by setting $\beta_i \geq 0$ (resp. $\beta_i \leq 0$) during the training. Furthermore, a monotonic increasing (resp. decreasing) constraint on the $i-$th feature is added by setting at the same time $\beta_i \geq 0$ (resp. $\beta_i \leq 0$) and the attention weight a_i to a constant value, typically one. The latter constraint is equivalent to assuming that the effect of feature i on the output is not influenced by other features, i.e., linear. A practical use case is demonstrated in Section 3.4.4.

3.3.4 On the robustness of TabSRALinear's explanations

In this section, we provide a theoretical analysis and justify why TabSRALinear is a viable solution for robust self-explainability in tabular learning settings.

Robustness remains an important topic for feature attribution based explanation systems, with the goal of designing a convenient metric to assess the similarity of explanations provided for similar inputs (Alvarez Melis and Jaakkola [158], Alvarez-Melis and Jaakkola [69], and Agarwal et al. [159]). This is mainly because many state-of-the-art interpretability tools (Lundberg and Lee [7], Ribeiro, Singh, and Guestrin [8], and Lundberg, Erion, Chen, DeGrave, Prutkin, Nair, Katz, Himmelfarb, Bansal, and Lee [64]) operate on a single data point and the use of point-wise explanations to understand complex models is perhaps too optimistic or can lead to a false sense of understanding (Alvarez-Melis and Jaakkola [69]). To address this limitation, one might want to go beyond individual points and examine the behavior of the models in the neighborhood of certain target points. Therefore, interpretability methods or inherently interpretable models must produce explanations that are stable or robust to local perturbations (Alvarez-Melis and Jaakkola [69]).

The TabSRALinear model is designed to produce relatively robust explanations considering the following theorem:

Theorem 1. *The feature attributions produced by TabSRALinear (Equation 3.4) are locally stable in the sense of Lipschitz, that is, for every* $\mathbf{x} \in \mathbb{R}^p$*, there exist* $\delta > 0$ *and* $L_{\mathbf{x}} \geq 0$ *finite such that:*

$$\|\mathbf{x} - \mathbf{x}'\|_1 < \delta \implies \|\boldsymbol{\beta} \odot a(\mathbf{x}) \odot \mathbf{x} - \boldsymbol{\beta} \odot a(\mathbf{x}') \odot \mathbf{x}'\|_1 \leq L_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}'\|_1 \tag{3.5}$$

With $L_{\mathbf{x}} = \|\boldsymbol{\beta}\|_\infty [\|\mathbf{a}\|_\infty + L_{\mathbf{a}}(\|\mathbf{x}\|_\infty + \delta)]$ *and* $L_{\mathbf{a}} \geq 0$ *the Lipschitz constant of the SRA block.*

The objective of Theorem 1 is not to provide the tightest bound of the Lipschitz constant, but to provide some justification for the attention computation.

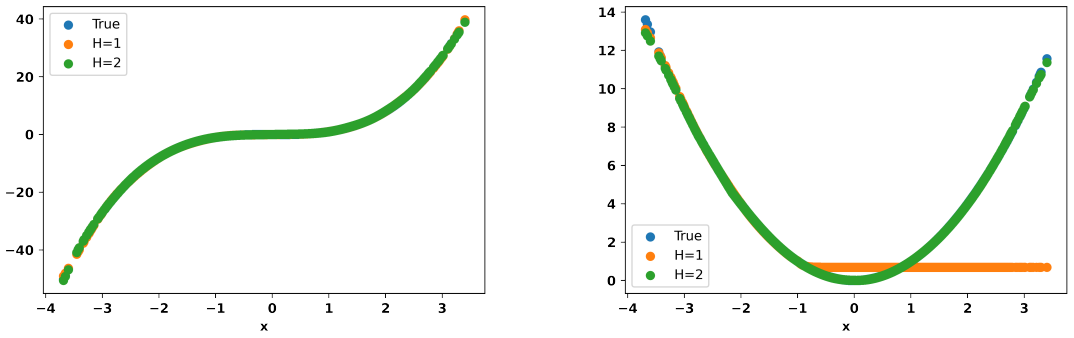
First, we notice the quantity $\|\mathbf{x}\|_\infty$ in the expression of $L_{\mathbf{x}}$, which shows that the raw input data should be **bounded**. This is a common situation when using NNs. That is, the data scaling technique (using the minimum and maximum or the mean and standard deviation) or quantile transformation is used to speed up the convergence.

We also have the term $\|\mathbf{a}\|_\infty$ which proves that the smaller the attention weights, the more stable the explanations. Using the scaling of Equation 3.3, we have $\|\mathbf{a}\|_\infty = 1$ and we can identify in the first term of $L_{\mathbf{x}}$ the Lipschitz constant of linear models, which is simply $\|\boldsymbol{\beta}\|_\infty$. Moreover, in situations where there are no interactions (and nonlinear effects) between features, almost all attention weights are expected to be constant (i.e., $L_{\mathbf{a}} \approx 0$), and TabSRALinear is reduced to classical linear models.

We present the empirical evidence of the stability of TabSRALinear's feature attribution compared with that of state-of-the-art interpretability tools such as (Lundberg et al. [64] and Lundberg and Lee [7]) in Section 3.4.2.

We kindly invite the reader to refer to Section A.1.2 for a complete proof of Theorem 1.

3.3.5 Improving TabSRALinear using model ensemble



(a) $f(\mathbf{x}) = \mathbf{x}^3$ **(b)** $f(\mathbf{x}) = \mathbf{x}^2$

Figure 3.6: *Illustration of the importance of the TabSRALinear ensemble. A single TabSRALinear (H=1) perfectly fits the monotonic function shown in Figure 3.6a. However, it struggles to fit the quadratic example (orange in Figure 3.6b). Using an ensemble of two (H=2) TabSRALinear models alleviates this problem: one TabSRALinear is specialized in fitting the first branch of the parabolic, while the second model of the ensemble fits the other branch.*

With the formulation of TabSRALinear (Equation 3.4), we assume that the learned function is monotonic or linear around the origin, that is, $\mathbf{x} \approx 0$. In other words, the Lipschitz constant (Theorem 1) is $L_{\mathbf{x}} = \|\boldsymbol{\beta}\|_\infty(1+\epsilon)$ where $\epsilon = L_{\mathbf{a}}\delta \longrightarrow 0$.
However, for some problems, the learned phenomena might not be monotonic around the origin (or around a given reference value), as illustrated by the quadratic function in Figure 3.6b. For such a problem, the TabSRALinear may provide poor modeling performance.

To mitigate this problem, we propose the use of model ensembling (Section 2.1.2), in which TabSRALinear is the base learner. To preserve the intelligibility of the overall architecture/ensemble, we found it more convenient to use the sum aggregation of the decisions of individual TabSRALinear (Equation 3.4) as follows:

$$\begin{aligned}\sigma(\hat{y}) &= \sigma(\hat{y}_1) + \cdots + \sigma(\hat{y}_h) + \cdots + \sigma(\hat{y}_H) \\ &= \beta_0^1 + \boldsymbol{\beta}^1 \cdot (\mathbf{a}^1 \odot \mathbf{x}) + \cdots + \beta_0^h + \boldsymbol{\beta}^h \cdot (\mathbf{a}^h \odot \mathbf{x}) + \cdots + \beta_0^H + \boldsymbol{\beta}^H \cdot (\mathbf{a}^H \odot \mathbf{x}) \\ &= \sum_{h=1}^{H} \beta_0^h + \left(\sum_{h=1}^{H} \beta_1^h a_1^h\right) x_1 + \cdots + \left(\sum_{h=1}^{H} \beta_i^h a_i^h\right) x_i + \cdots + \left(\sum_{h=1}^{H} \beta_p^h a_p^h\right) x_p\end{aligned} \tag{3.6}$$

where $\boldsymbol{\beta}^h$ (resp. $\mathbf{a}^h$) is the h-th linear model's coefficients (resp. the attention vector from the SRA block h) and β_0^h is the corresponding bias term. H represents the size or the number of learners in the ensemble. The TabSRALinear ensemble is equivalent to the Multi-head attention idea (described in Section 2.1.3) where context representations provided by different attention heads are first concatenated and thereafter transformed by a linear layer producing a new context representation or the final output.
In general, ensembling increases the number of parameters (compared to one single base

learner), consequently affects transparency. However, due to the additive nature, TabSRALinear ensemble (Equation 3.6) preserves the interpretable aspect of TabSRALinear that is, $\left(\sum_{h=1}^{H} \beta_i^h a_i^h\right) x_i$ represents the contribution or the effect of the feature x_i to the output. Nonetheless, a high number of H may result in less robust explanations (please refer to Section A.1.1 for the theoretical proof). Therefore, we recommend considering the ensemble size H from the set {1,2}, and for all results presented in this Chapter, including the case study, we optimized the ensemble size $H \in \{1, 2\}$. Furthermore, our ablation study (Table 3.6, Section 3.4.3) indicates that increasing H beyond 2 does not significantly enhance performance but instead increases running time.

We will refer to the TabSRALinear ensemble as TabSRALinear, and we will use the terms ensemble or head interchangeably.

3.4 Empirical study

In this Section, we report the key findings of the empirical comparison of inherently interpretable models to each other, as well as to full-complexity counterparts. Please note that the supplementary materials contain (1) a complete results analysis regarding the stability of TabSRALinear's explanations (Section A.2.4); (2) additional visualizations and results (Section A.2).

3.4.1 Experimental setup

Evaluation metrics

Our goal is to provide concrete numerical results to help practitioners and researchers choose between inherently interpretable solutions and explanations for full-complexity models. Given that the main justification for explaining blackbox models is to retain predictive performance while achieving interpretability, we believe it is crucial to compare both approaches (inherently interpretable models versus black-box models with XAI tools) based on the following criteria:

- **Predictive performance** (Section 3.4.2). To assess predictive performance, we utilize the coefficient of determination (R^2) for regression tasks and accuracy for classification tasks in the *middle-scale benchmark*. For datasets in the *Default benchmark* that are imbalanced, we evaluate the Area Under the ROC Curve (AUCROC), and particularly for highly imbalanced datasets such as the Credit Card Fraud dataset (Table 3.2), we measure the Area Under the Area Under the PR Curve (AUCPR) instead of AUCROC (as discussed in Section 2.1.1).

- **Faithfulness** (Section 3.4.2). Given that the faithfulness of explanations provided by blackbox models with XAI tools is often questioned in the literature (Amoukou, Salaün, and Brunel [9] and Huang and Marques-Silva [60, 11]), we also compare the

two approaches based on this criterion. Specifically, we measure the precision in identifying the truly relevant features.

- **Stability/Robustness** (Section 3.4.2). Which option can provide more robust/coherent explanations, meaning similar explanations for similar inputs?

We further illustrate the unique capabilities of our proposed solution, TabSRALinear, and knowledge incorporation through two real-world applications (Section 3.4.4).

Datasets

To evaluate the models based on the criteria listed above, we considered three types of datasets:

- *Middle-scale benchmark*: Recently introduced in Grinsztajn, Oyallon, and Varoquaux [53], it contains 45 real-world datasets (59 tasks precisely) of both numerical and heterogeneous binary classification or regression problems. We mainly use these datasets to assess the predictive performance of benchmark models. Note that the authors truncated the training set to 10,000 and the test set to 50,000 in order to ease the assessment of inductive biases of models (have homogeneous benchmarks). Also, cross-validation is used with a number of folds ranging from 1 to 5 depending on the test data size. Finally, in each fold, a Train/Validation/Test split is used (please refer to Section A.2.1 for more details).

- *Default benchmark*: we also consider four widely used datasets in tabular learning settings (we provide the summary in Table 3.2). We used two of these datasets to evaluate the robustness of explanations (Credit Card Fraud, Heloc Fico) and the remaining for the applicative case study.

- *Synthetic benchmark*: we finally consider synthetic datasets to assess explanations' faithfulness. As the ground truth for real-world datasets are generally unavailable for this purpose, we generate three synthetic datasets based on (1) additivity: we consider only additive functions. Recall that most feature attribution methods are used to explain the additive structure in modeled phenomena (Kumar et al. [10]) and (2) unicity: there is only one optimal explanation or feature attribution for every observation. Therefore, models that perfectly fit the data in terms of accuracy should converge with this explanation. In this concern, we consider datasets with five features $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ of size 30,000 based on the Gaussian distribution (of mean 0 and variance 1) as follows:

$$\textit{Synthetic 1}: \quad y = 5x_1 - 5x_2 \tag{3.7}$$

$$\textit{Synthetic 2}: \quad y = x_1^2 \tag{3.8}$$

$$\textit{Synthetic 3}: \quad y = (5x_1 - 5x_2)\mathbb{1}_{x_5 \leq 0} + (5x_3 - 5x_4)\mathbb{1}_{x_5 > 0} \tag{3.9}$$

Table 3.2: *Benchmark datasets. CF: Categorical features*

Datasets	# Datapoints	# features	# CF	Positive Class (%)
Bank Churn	10,000	9	2	20.37
Credit Default	30,000	22	3	22.16
Credit Card Fraud	284,807	29	0	0.17
Heloc Fico	10,459	23	0	47.81

Experimental details on the predictive performance evalution

Tuning. For the *Middle-scale benchmark*, we conducted random hyperparameter tuning for each inherently interpretable algorithm (described in Section 3.2.1, including TabSRALinear) and CatBoost (Section 3.2.2) over a period of 9 days (216 hours) on a 64-core processor CPU machine.

For the remaining full-complexity models (Section 3.2.2), the tuning results are reported from the previous benchmarking (Grinsztajn, Oyallon, and Varoquaux [53]) where a 64-Core Processor CPU machine is used for tree-based models and GPUs for NNs (please refer to [53] for more details). In this study, we make our best effort to ensure the convergence of algorithms, particularly NNs that have an important number of hyperparameters to tune and, at the same time, have the fairest comparison possible. Therefore, we consider $\approx$ 2x iterations per dataset for MLP, RF and $\approx$ 3x for XGBoost, CatBoost where x is the number of iterations for TabSRALinear (≈ 95 per dataset). For ResNet, FT-Transformer and SAINT which are more resource-consuming (see the training time Table 3.3), we used the same number of iterations as TabSRALinear.

Moreover, in order to have a bootstrap-like estimate of the test score (Grinsztajn, Oyallon, and Varoquaux [53]), for each algorithm and dataset we sample 50% of iterations and repeat this process 10 times with different random seeds. Note that using all iterations once does not change our conclusions (Section A.2.6).

Results aggregation accross datasets. We consider the test accuracy for binary classification tasks and R^2 score for regressions. To aggregate the results across datasets, we use a metric similar to the Average Distance to the Minimum (ADTM) (Wistuba, Schilling, and Schmidt-Thieme [160]). More precisely, for each algorithm, we divide the achieved score by the one of the best performing for a given dataset. In this way, we get a unitless score in $[0, 1]$ which is further aggregated to produce the relative average score to best. We also consider the rank and the running time (training + testing).

Data processing. Unless otherwise specified, categorical inputs are one-hot encoded for

Table 3.3: *Predictive performance of models across 59 tasks (45 datasets). We report the rank over all tasks, the relative test score (Accuracy/R^2) and running time (training+inference) in seconds. MRT: Mean Running Time*

Model	Rank				Mean Test Score			MRT [3]	
	min	max	mean	median	mean	median	std	mean	median
DT	2	12	10.476	11	0.868	0.907	0.163	0.294	0.032
EBM_S	1	11	7.692	8	0.931	0.955	0.087	23.997	5.144
EBM	1	10	5.477	5	0.959	0.982	0.067	97.837	19.737
LR	7	12	11.701	12	0.760	0.839	0.232	21.124	19.716
TabSRALinear	1	12	8.225	9	0.901	0.971	0.197	47.576	38.073
MLP	1	12	6.992	8	0.924	0.973	0.159	24.165	19.256
ResNet	1	12	7.120	8	0.909	0.975	0.195	95.123	53.212
SAINT	1	12	5.625	6	0.946	0.982	0.093	216.053	126.841
FT-Transformer	1	11	5.203	5	0.944	0.984	0.109	126.589	77.465
Random Forest	1	10	4.214	4	0.985	0.992	0.021	39.030	8.252
XGBoost	1	11	2.728	2	0.988	0.998	0.029	18.254	12.561
CatBoost	1	10	2.545	2	0.991	0.999	0.021	12.176	4.025

models which do not handle them natively[1], and numerical inputs are scaled[2] using mean and standard deviation to accelerate the convergence of NNs models.

3.4.2 Benchmark results

Prediction performance

In Table 3.3, we report the predictive performance results of inherently interpretable and full-complexity models.
The main findings are:

- The relative average predictive performance gap between the overall best-performing inherently interpretable (EBM) and the full-complexity counterpart (CatBoost) is less than 4%.

[1]The Neural Networks NNs described in Section 3.2.2 include an embedding layer designed to manage categorical features, as outlined in the work by Gorishniy et al. [43]. NNs (Section 3.2.2) are equipped with an embedding layer for handling categorical feature (Gorishniy et al. [43]) and CatBoost also offers a native approach for handling categorical features through a combination of target encoding (referred to as 'Borders') and Frequency Encoding (referred to as 'Counter')

[2]Gaussian quantile transformation was used in Grinsztajn, Oyallon, and Varoquaux [53]; however, as this transformation is not bijective/linear, we only use scaling to preserve interpretability in the initial feature space.

- Regarding NNs models, the inherently interpretable solution TabSRALinear provides a very competitive performance compared with MLP-like architectures (MLP, ResNet) with a gap of less than 2.5%.

For tasks where predictive performance is the only consideration, decision tree ensemble models (such as CatBoost, XGBoost, Random Forest) are clearly preferable solutions, especially when run time is considered. Compared with tree-based models, NNs have a longer general training time when using gradient descent optimization. This fact is highlighted by the running time of Linear models (LR) optimized by gradient descent compared to that of Decision Trees (DT); both are models known to be very shallow.

We can notice (Table 3.3) that the average relative test score between CatBoost and TabSRALinear is up to 9% while the median (i.e., on 50% of datasets) is less than 3%. This is due to the poor performance of the latter on some decision tree friendly datasets (see the comparison results on delays_zurich_transport Table A.14 and yprop_4_1 Table A.15). On such datasets, a simple decision tree may have superior predictive performance over MLP-like architectures or shallow NNs. FT-Transformer and SAINT, owing to their feature tokenization process, can still provide acceptable performance in such situations (see analysis on when the FT-Transformer is better than ResNet [43]). However, this comes with the cost of an important number of learnable parameters. We believe that combining numerical discretization (Gorishniy, Rubachev, and Babenko [39]) and TabSRALinear or simply using piecewise constant approximators such as EBM could be a good predictive inherently interpretable solution in such situations.

For this *middle-scale benchmark*, CatBoost significantly outperformed EBM (with a gap above 5%) for 9/59 tasks. We argue that this is because of the difficulty of GAM based inherently interpretable solutions to conveniently handle higher-order interactions (above 3). For such cases, TabSRALinear appears to be a good solution, particularly when the dataset does not exhibit strong discontinuities or irregularities (see the results for covertype Table A.20, pol, and sulfur Table A.15).

Faithfulness of explanations

In this part, we try to answer the following question: *"Is it true that inherently interpretable models produce more reliable explanations?"* For this purpose, we consider the XAI solution, XGBoost (one of the best predictive models in Table 3.3), explained by the tree path-dependent SHAP algorithm (TreeSHAP [64]). As justified in Section 3.2.2, this approach, i.e., XGBoost+TreeSHAP, is arguably one of the most used solutions by practitioners. As inherently interpretable solutions, we consider all the listed models in Section 3.2.1 except Decision Tree (DT) as it does not produce direct feature attributions. For the datasets, we consider the additive *Synthetic benchmark* (Section 3.4.1) with 80/20% Train/Test split.

The example called *Synthetic 1* is linear regression friendly and only x_1 and x_2 are relevant. The example *Synthetic 2* represents a parabolic function; therefore linear regression cannot accurately model it unless identifying and adding the convenient quadratic terms of the raw input features. Only x_1 should have non-zero importance/attribution for this example. Finally, the *Synthetic 3* borrowed from (Amoukou, Salaün, and Brunel [9]) high-

Table 3.4: *Relevant feature discovery capacity. Precision is used as a metric. R^2 (the higher, the better) is used to evaluate the test's predictive performance. The bold numbers denote the best for each dataset and metric.*

Datasets	Models	Test performance	Precision
Synthetic 1	LR	1.00	1.00
	TabSRALinear	1.00	1.00
	EBM_S	1.00	1.00
	EBM	1.00	1.00
	XGBoost+TreeSHAP	1.00	1.00
Synthetic 2	LR	0.00	0.00
	TabSRALinear	**1.00**	1.00
	EBM_S	0.99	0.99
	EBM	0.99	0.99
	XGBoost+TreeSHAP	0.99	1.00
Synthetic 3	LR	0.50	0.52
	TabSRALinear	1.00	**0.99**
	EBM_S	0.49	0.51
	EBM	0.98	0.97
	XGBoost+TreeSHAP	1.00	0.78

lights interactions between the features. For this example, a perfect modeling algorithm should use, depending on the sign of x_5, the features x_1 and x_2 only, or alternatively, the features x_3 and x_4. We restrict our analysis to those data points with $x_5 \leq 0$, which comprise $\approx 3,300$ instances. Therefore, only x_1 and x_2 are relevant among (x_1, x_2, x_3, x_4). We use precision in finding the most relevant features as an evaluation metric, while the R^2 is used to assess the test's predictive performance.

Here are the main takeaways from Table 3.4:

- An inherently interpretable model with poor predictive performance or inductive bias can produce incorrect feature attributions or misleading discoveries. This is highlighted by the results of LR on *Synthetic 2* and EBM_S (EBM without interaction terms) on *Synthetic 3*.
- Using *post hoc* interpretability tools can lead to incorrect feature attribution, although the perfect predictive performance of the underlining model. It is the case of XGBoost+TreeSHAP on the *Synthetic 3*. This dataset highlights the feature interactions that are well-known situations in which most *post hoc* explanation tools struggle to find 'truly' relevant features of the underlying model (Amoukou, Salaün, and Brunel [9], Kumar et al. [10], Huang and Marques-Silva [11], and Chen et al. [63]).

Overall, we can notice from these synthetic examples that an inherently interpretable model with "good" predictive performance/inductive bias usually produces reliable feature attribution. This is the case of TabSRALinear or EBM, which, with a predictive performance 2%

lower than that of XGBoost on *Synthetic 3*, manages to discover the most important features with a precision of 97%, compared with 78% for XGBoost.
It is important to point out that for EBM, we move the value of pairwise interaction terms containing x_5 to the remaining main effect. In other words, we add for example the value of the interaction $f_{15}(x_1, x_5)$ to $f_1(x_1)$, producing the contribution of x_1. This is possible because we know the ground truth; otherwise, heuristic methods such as interaction purification (Lengerich et al. [29]) are used to solve identification ambiguity problems. Recall that TabSRALinear, thanks to its formulation, does not require any interaction purification.

Stability of explanations

We also evaluate the interpretable solutions (used in Section 3.4.2) based on the robustness/stability of explanations, which involves answering the following question: *"Are the produced explanations similar for similar inputs?"* For this purpose, we use two real-world well-known numerical[4] datasets: Credit Card Fraud and Heloc Fico (Table 3.2). We consider the continuous notion of stability (Alvarez-Melis and Jaakkola [69]) and we estimate the local Lipschitz constant as follows:

$$\hat{L}(\mathbf{x}) = \underset{\mathbf{x}' \in \mathcal{N}_\epsilon(\mathbf{x})}{\arg\max} \|f_{expl}(\mathbf{x}) - f_{expl}(\mathbf{x}')\|_2 / \|\mathbf{x} - \mathbf{x}'\|_2 \tag{3.10}$$

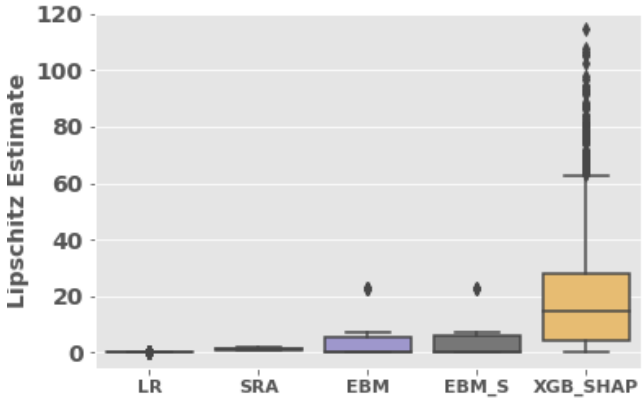
where the vector $f_{expl}(\mathbf{x})$ is the feature attribution (the explanation) for the given observation $\mathbf{x}$. For linear models, $\hat{L}(\mathbf{x})$ is equivalent to $\|\boldsymbol{\beta}\|_\infty$ and for TabSRALinear it is proportional to $L_{\mathbf{x}}$ (Theorem 1).

We use 80/20% Train/Test split, and we generate 100 neighbors $\mathcal{N}_\epsilon(\mathbf{x})$ by adding random Gaussian noise $\mathcal{N}(0, \epsilon \times I)$ to every target point $\mathbf{x}$ on the test data. The used perturbation is small enough to avoid excessively changing the predictions (Section A.8). Thus, we use $\epsilon = 0.001$ for the Credit Card data and $\epsilon = 0.01$ for the Heloc Fico dataset.

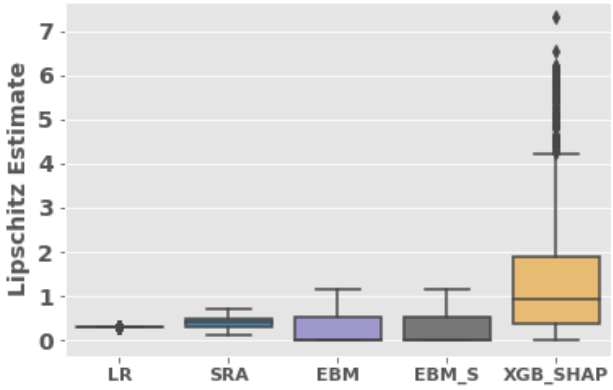
We can see from Figure 3.7 that the considered inherently interpretable models can produce more robust explanations than XGBoost+TreeSHAP. Due to their piecewise constant structure, Explainable Boosting Machines (EBMs) tend to produce usually similar explanations for similar inputs; however, they may also produce some outliers (see the example of the Credit Card Fraud dataset Figure 3.7a) that arguably are due to some abrupt change points in the learned shape functions. Regarding XGBoost+TreeSHAP, the important variability in explanations can be either explained by XGBoost's flexibility to capture local discontinuities or the incapacity of the interpretability tool to reproduce the correct local variability (Alvarez-Melis and Jaakkola [69]).

Apart from the Linear models (LR), which are known to be robust and conservative, TabSRALinear seems to be an encouraging trade-off for robust interpretability, especially when adding predictive performance: test AUCROC for Heloc Fico (LR = 0.781; TabSRALinear= 0.795; EBM_S=0.795; EBM=0.798; XGboost= 0.798) and test AUCPR for Credit Card

[4]For the sake of brevity, we used datasets containing only numerical features, as the notion of neighborhood and Lipschtitz estimate is not trivial for categorical and discrete features. For more details, see Alvarez-Melis and Jaakkola [69]



(a) *Credit Card Fraud dataset: 2533 random test points*



(b) *Heloc Fico dataset: 1076 random test points*

Figure 3.7: *Estimation of Lipschitz constant on real word datasets (the lower the better). LR = Logistic Regression, SRA=TabSRALinear, XGB_SHAP=XGBoost+TreeSHAP*

Fraud[5] (LR = 0.734; TabSRALinear= 0.844; EBM_S=0.831; EBM=0.834; XGboost=0.848).

3.4.3 Ablation study for TabSRALinear

To have more robust and intelligible explanations, we mentioned in Section 3.3.5 and Section A.1.1 that it is recommended to consider the size or number of ensemble/head $H \in \{1, 2\}$. In this Section, we empirically show how varying some hyperparameters, such as the number of the ensemble, can impact the predictive performance of the TabSRALinear model. First, we provide some details on the implementation.

We use the same architecture for the key and query encoders (Section 3.3.2) which is

[5]This dataset is highly imbalanced therefore AUCPR is a more appropriate than AUCROC [17]

Table 3.5: *Influence of the dimension of the query/key encoder d_k using the Middle-scale benchmark. H is set to 1. Each value of d_k is tuned with 17 random iterations.*

d_k	Rank				Mean Test Score			Mean Running Time	
	min	max	mean	median	mean	median	std	mean	median
4	1	3	2.203	2	0.933	0.994	0.183	35.946	29.875
8	1	3	1.992	2	0.961	0.997	0.143	39.429	37.529
12	1	3	1.805	2	0.958	0.999	0.168	40.719	34.767

Table 3.6: *Influence of the number of ensemble H using the Middle-scale benchmark. Each value of H is tuned with 30 random iterations.*

H	Rank				Mean Test Score			Mean Running Time	
	min	max	mean	median	mean	median	std	mean	median
1	1	6	4.169	5	0.954	0.994	0.142	40.758	36.330
2	1	6	3.542	4	0.961	0.996	0.154	48.419	38.817
3	1	6	3.458	3	0.953	0.995	0.162	63.703	57.671
4	1	6	3.127	3	0.963	0.996	0.160	61.878	53.618
5	1	6	3.203	3	0.977	0.997	0.103	91.683	62.889
6	1	6	3.500	4	0.969	0.996	0.116	99.672	73.969

M hidden layers fully connected neural network with ReLU activation. The dimension of $i-$th hidden layer is $m_i = pd_k//2^{(M-i)}$ for $i = 0, ..., M-1$ where $M \geq 1$, $d_k//2^M \geq 1$, p is the number of features and d_k the dimension of the query/key vectors (Section 3.3.2). With such implementation, the total number of learnable parameters in TabSRALinear is:

$$N = H\left[(p+1) + 2(2 - (\frac{1}{2})^M)pd_k + \frac{1}{2}(M-1)p^2d_k + \frac{4}{3}(1 - (\frac{1}{4})^M)p^2d_k^2\right] \tag{3.11}$$

with H the number or size of the ensemble/head.

Dimension of the query/key encoder d_k. In Equation 3.11, we can notice that the number of parameters is quadratic in d_k. Therefore, we recommend not setting the value d_k too high (typically consider $d_k \in [4, 8, 12]$). In our experiments, $d_k = 8$ is a good default value (as shown in Table 3.5). Overall, it remains a tunable parameter depending on the case study.

Number of ensemble H. Increasing the size of the ensemble/head H leads to a rise in the number of parameters; consequently, extending the processing time, as demonstrated in Table 3.6. However, the change in the predictive performance is not significant for the *Middle-scale benchmark*. As for the trade-off between the running time and intelligibility, we recommend optimizing $H \in \{1, 2\}$.

The number of hidden layers in query/key encoder M. So far, we have given the

results of TabSRALinear with one hidden encoder $M = 1$, in order to avoid learning suspicious interactions.

We explore the predictive performance for $M = 2$ utilizing the *Middle-scale benchmark*. TabSRALinear, with $M = 1$, emerges victorious in 31 out of 59 tasks. This outcome is not surprising given our focus on middle-scale datasets. However, we anticipate that employing $M = 2$ could yield comparable or potentially superior performance for larger datasets characterized by potentially strong interactions.

3.4.4 Real world application of TabSRALinear

In this Section, we show the value of the TabSRALinear inherently interpretable solution considering two real-world applications: bank churn modeling and credit default prediction (80/20% Train/Test split).

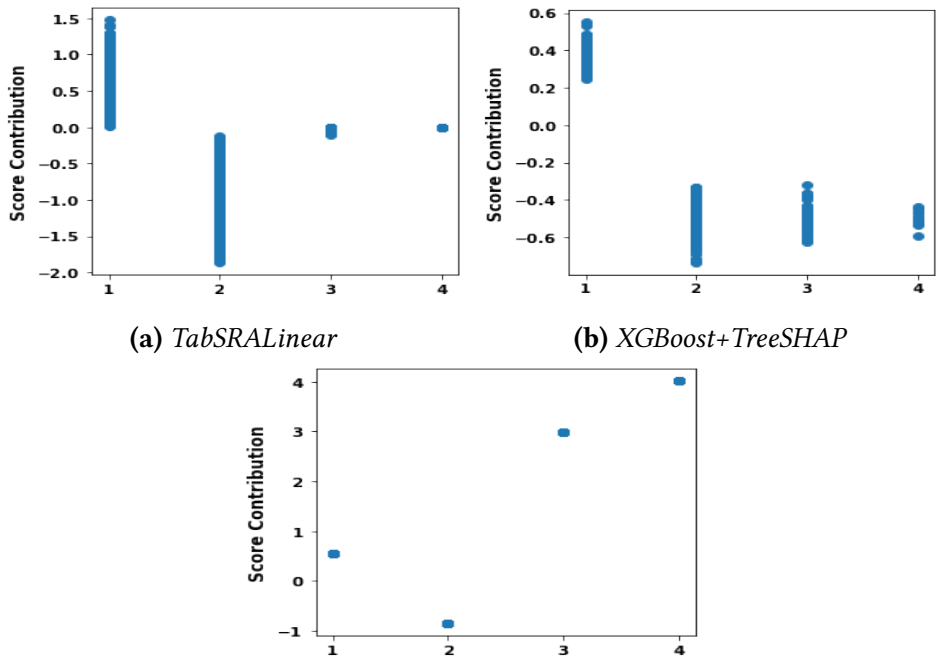
Application 1: Bank churn modeling

This first example illustrates how one can leverage TabSRALinear's knowledge incorporation capabilities more specifically, the negative sense constraint (described in Section 3.3.3) to mitigate identified bias in data collection.
We consider the Bank churn modeling dataset where the objective is to predict if a customer is going to churn. For this type of problem, it is important to explain, or find out for customers of interest, the characteristics that may lead them to want to close their accounts. This enables customer services to accurately target the profiles to be contacted.

More importantly, there is one feature called *NumOfProducts*, which refers to the number of products purchased by a customer through the bank (Table 3.7). Intuitively, a customer with a high *NumOfProducts* is less likely to leave the bank. However, according to the exploration of the collected data of 10000 recordings (Table 3.7), 100% of customers with 4 products have left the bank. This probably indicates a bias in the data, especially since only 60/10000 observations have 4 products. Similarly, customers with 3 products (266/10000) have $\approx 83\%$ as churn rate whereas for the majority of customers (with 1 and 2 products), the risk of churn decreases with the number of products.

Table 3.7: *Statistics for the target feature (Exited) with respect to the feature NumOfProducts of the churn modeling data set: fraction (Frac), number (Nb).*

NumOfProducts	Statistics		
	Frac. Exited (%)	Nb. Exited	Nb. of observations
1	28	1409	5084
2	8	348	4590
3	83	220	266
4	100	60	60



(a) *TabSRALinear* **(b)** *XGBoost+TreeSHAP*

(c) *EBMs*

Figure 3.8: *Bank churn modeling: Effect contribution of the feature NumOfProducts ranging from 1 to 4*

Logistic Regression (LR) is in accordance with human intuition when producing a negative coefficient for the feature *NumOfProducts*. Nevertheless, its overall predictive performance (i.e., test AUCROC) is 0.781 compared to 0.857 and 0.874 for EBM_S and EBM respectively. This performance gap is arguably due to nonlinear effects and interactions present in this data (see Section A.2.3 for more details). Although their test AUCROC is good, Explainable Boosting Machines (EBMs) exactly reproduce the bias in the data, as shown in Figure 3.8c.

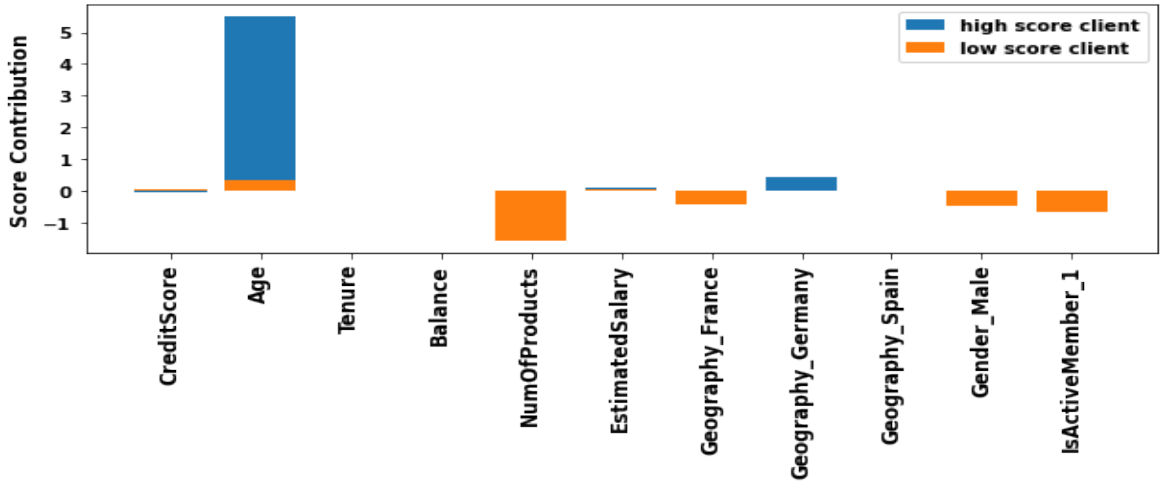
Regarding XGBoost, there is a monotonic option in which one can typically constrain the effect of a given feature to grow monotonically (decreasing). Doing so for XGBoost, as shown in Figure 3.8b results in a drop of the test AUCROC from 0.872 to 0.821[6][7]
In this application, the uniqueness of TabSRALinear lies in its negative weight (sense) constraint (Section 3.3.3) which helps maintain a test AUCROC of 0.854 in the addition of the inherent intelligibility. That is, instead of using a global monotonic constraint (as for LR and XGBoost), the negative weight constraint helps TabSRALinear to consider a decreasing effect for a *NumOfProducts* from 1 to 2. At the same time, this constraint forces the TabSRALinear to not directly use *NumOfProducts* to assign a high churn score for customers with more than 3 products, highlighted by a nearly zero effect contribution (Figure 3.8a). This is achieved by producing an attention weight close to zero for *NumOfProducts*. For

[6]One may want to process the *NumOfProducts* in binary feature, i.e., 1 product versus more than 1 product. This will help maintain the test AUCROC of XGBoost at approximately 0.828 for this dataset.

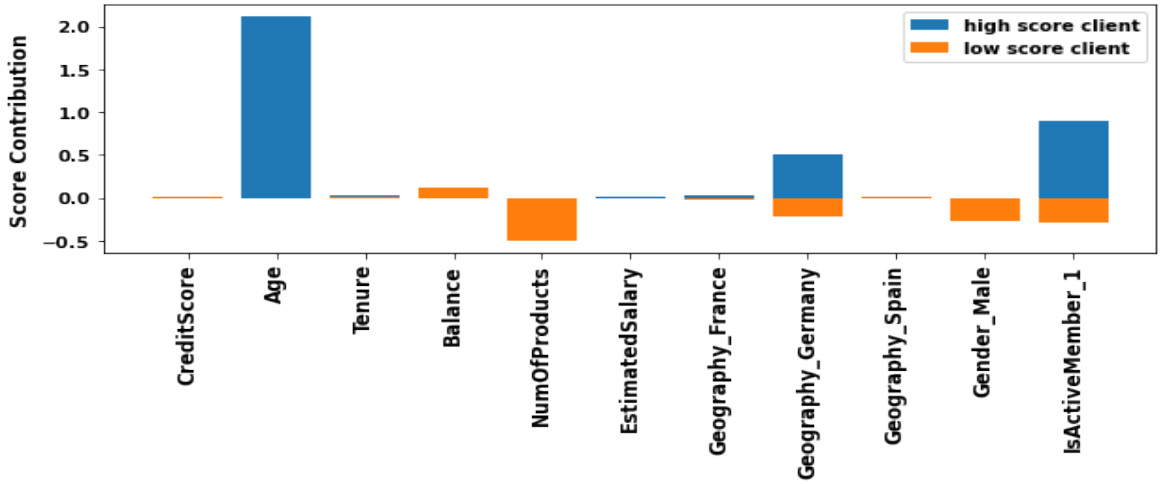
[7]The obtained explanations using TreeSHAP are not actually monotonic decreasing as they are supposed to be Figure 3.8b. That is, the effect of the *NumOfProducts* for some customers with 3 products is higher than the one of some customers with 2 products.

these customers, the churn risk, according to TabSRALinear, is usually due to their *Age* and the fact that their *Geography=Germany*.

In Figure 3.9, we illustrate the feature attribution for two customers, one with a high churn risk (score) and the remainder with a low score. Specifically, we consider a high churn risk customer with *NumOfProducts=3* (meaning that *NumOfProducts* should not be an important churn risk factor/feature for this customer after the bias correction). Both TabSRALinear and XGBoost+TreeSHAP indicate that *Age=63* and *Geography=Germany* are the two most important features. On the other hand, the low churn risk customer has 2 products, i.e., *NumOfProducts=2*, is an active member (*IsActiveMember_1=1*) and is a male (the churn rate is 16% for *Gender=Male* and 25%, remaining).



(a) *TabSRALinear*



(b) *XGBoost+TreeSHAP*

Figure 3.9: *Individual prediction understanding for the bank churn modeling.*

These explanations are confirmed by our data exploration because the churn rate of customers in Germany is twice that of other countries, and customers with *Age* around 60 are very likely to churn, especially when they are not active members ($\approx$ 90% of churn

rate). We argue that this category includes customers who are close to or at retirement age and who are, therefore, likely to be looking for a bank that can advise them on tax, pension, and wealth management issues. So they are apt to change banks, especially when they are less active (interacting less with their current bank). Additional results on the interactions of the *Age* and *ActiveMember* indicators are provided in Section A.2.3.

Application 2: Credit Card Default prediction

Another criterion for evaluating model interpretability is the human-friendliness of explanations, meaning they should be concise and easily understood. In this Section, we demonstrate that TabSRALinear provides concise explanations compared to existing solutions using the credit default dataset. This dataset predicts the default probability of credit card clients in Taiwan and includes a group of correlated features with:

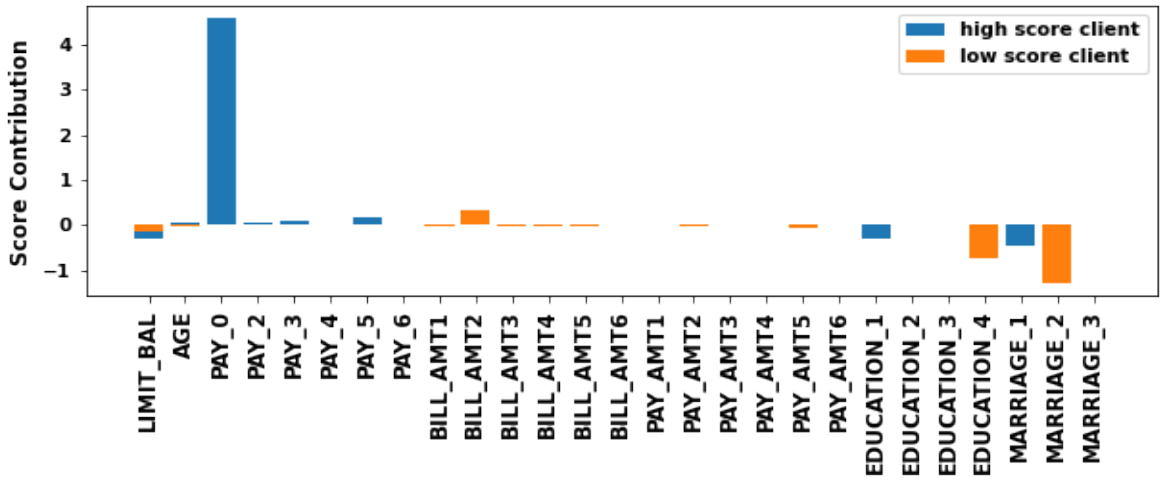
- 6 related to the repayment status (ranging from -2, paid two months in advance, to 8, eight months overdue);
- 6 related to the bill statement amount;
- 6 related to the previous payment amount.

In the case of a high default risk client shown in Figure 3.10, TabSRALinear focuses mainly on PAY_0, which represents the repayment status in the last month before the prediction. Our exploratory analysis confirms that PAY_0 is the most important risque factor for this dataset. Unlike TabSRALinear, XGBoost+TreeSHAP spreads the contribution among the set of correlated features, resulting in less sparse feature attribution (Amoukou, Salaün, and Brunel [9], Kumar, Venkatasubramanian, Scheidegger, and Friedler [10], and Chen, Covert, Lundberg, and Lee [63]).

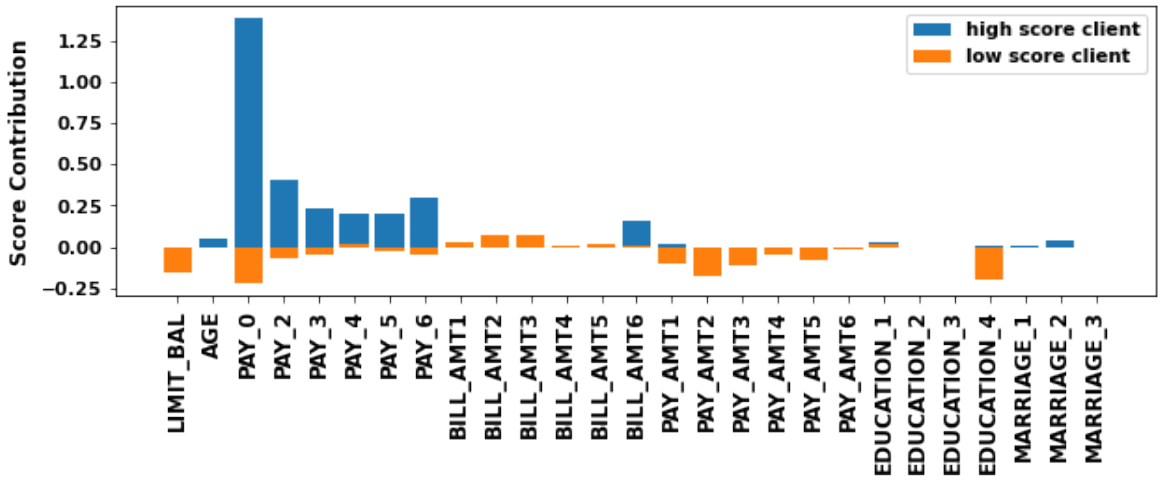
To confirm this observation, we illustrate (in Figure 3.11) using all the test points (6000 precisely), the capacity of PAY_0 to separate high default score customers (in yellow). According to both TabSRALinear and XGBoost, the default risk becomes important from 2 months of delay in the repayment status PAY_0. In addition, TabSRALinear came up to isolate almost its high default score (in yellow) based on PAY_0 (with the simple rule contribution PAY_0$>$ 3). For XGBoost+TreeSHAP, this separation is less clear, demonstrating the need to add information or visualization regarding the remaining features (which are already correlated to PAY_0). We recall that test AUCROC is very close for these two models: 0.794 for XGBoost and 0.791 for TabSRALinear.

To produce concise explanations (feature attribution), TabSRALinear shows its superiority over other solutions such as XGBoost+TreeSHAP (the conclusion is the same for EBMs, Section A.2.3). We recall that the feature attribution of TabSRAlinear becomes very sparse for problems when one hot encoding is used for a categorical feature. That is, all reference features (with value 0) have exactly 0 effect contribution as for classic linear models. We summarize the **main takeaways** from the case studies in two points:

- Full-complexity models can provide superior predictive performance. However, after performing important model diagnostics (e.g., bias correction, adding monotonic



(a) *TabSRALinear*



(b) *XGBoost+TreeSHAP*

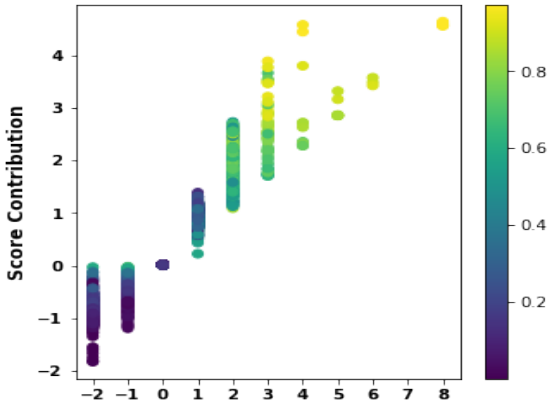
Figure 3.10: *Individual prediction understanding for the credit card default dataset.*

constraints), their performance can drop significantly, even under an inherently interpretable solution that is more flexible for knowledge incorporation (such as TabSRALinear).

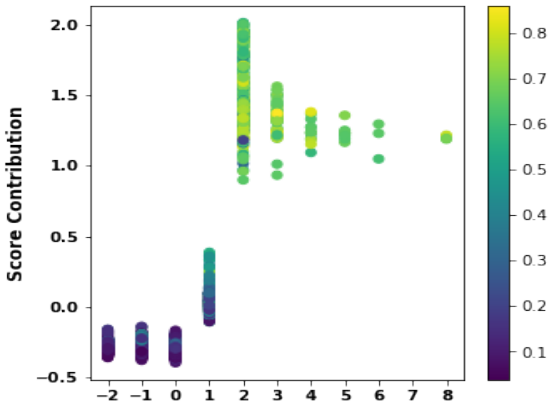
- When the sparsity of the explanations is an important consideration, it is worth nothing to test an inherently interpretable model such as TabSRALinear, especially for datasets with an important number of one-hot encoded features.

3.5 Limitations and Recommendations

In this study, we demonstrated through benchmarking that state-of-the-art inherently interpretable machine learning models generally achieve predictive performance very close to that of full-complexity counterparts (with less than 4% relative performance gap on 59



(a) *TabSRALinear*



(b) *XGBoost+TreeSHAP*

Figure 3.11: *From individual to global effect understanding for the PAY_0 feature of the credit card default dataset. On x-axis, we show the number of months of delay in the payment ranging from -2 to 8.*

tasks/45 datasets). However, to draw rigorous conclusions about the inductive biases of the algorithms, the datasets were selected to ensure no concept drift between the training and test data, balanced classes for binary classification problems, and a limited number of 10,000 observations for training data and 50,000 for test data following Grinsztajn, Oyallon, and Varoquaux [53]. How would the inclusion of concept drift, class imbalance, and big data (datasets comprising millions of observations) impact results? With respect to the data size, our findings from the *Default benchmark* (Table 3.2), which includes up to 285,000 observations, and those from Chang, Caruana, and Goldenberg [66] (6 datasets with over 500,000), suggest that the performances are still comparable. However, further investigation is necessary for datasets that are (highly) imbalanced and problems involving concept drift, and we will address this in the next Chapter.

In addition, we listed some models in Section 3.2.1 as inherently interpretable solutions, but different regulators or stakeholders may have different requirements for the use case (the model must be very transparent, such as a small set of decision rules, linear models with few predictors or features). For certain classes of models, interpretability can be assessed based on the number of parameters or complexity. For instance, in statistical models like

Logistic Regression, the model's interpretability increases as the number of predictors or non-null parameters decreases. Similarly, in decision trees, interpretability can be assessed based on the depth or the number of nodes. However, in models such as the Explainable Boosting Machine (EBM) within Generalized Additive Models (GAMs), interpretability is not directly quantified by the number of trees or parameters (which can often be in the thousands), but also in terms of predictors, akin to Logistic Regression. This is because all these trees are aggregated into a one-dimensional piecewise constant function per predictor. Similarly, in the case of TabSRALinear, interpretability is not directly linked to the number of parameters or weights, but rather to the predictors themselves. Therefore, merely counting the number of nodes or weights does not necessarily enhance interpretability (e.g., for feature attribution), but rather serves the purpose of transparency. Overall, we recommend that interested readers/practitioners do not consider the term "inherently interpretable" as generic, but instead define it based on the specific use cases and requirements.

In general, it is challenging, if not impossible, to find a single solution that offers optimal performance across all metrics, including predictive performance, faithfulness, stability, and knowledge incorporation, irrespective of the dataset or problem at hand (a concept known as the "no free lunch theorem").

In some scenarios, accuracy may be the primary concern, while in others, interpretability takes precedence. For tabular learning tasks, where understanding the predictions and discovering hidden patterns is an important consideration, we recommend practitioners to begin by using both full-complexity models (typically tree ensembles, which, in addition to performance, are not greedy in terms of computational cost and can be easily explained using TreeSHAP [64]) and inherently interpretable solutions (typically GAMs: piecewise constant approach such as EBMs (Nori et al. [27]), piecewise linear approach such as GAMI-Net (Yang, Zhang, and Sudjianto [30]), and attention-based solution such as TabSRALinear). Depending on the available computational resources, one could begin by using default hyperparameters or a fixed set of hyperparameter configurations. One may opt to keep only the inherently interpretable models if their predictive performances are *comparable* to those of the full-complexity model under consideration, and discard the latter. Otherwise, it may be advisable to also explain the full-complexity model using *post hoc* tools. After conducting any necessary model diagnosis (e.g., bias correction), the final model can be chosen based on the number of explanations that align with the data knowledge, after taking advantage of the Rashomon effect (Müller et al. [161]) (where different "good" predictive models can offer different explanations[8]). Additionally, it is important to note that inherently interpretable solutions generally do not require additional computational cost to produce explanations.

Regarding TabSRALinear, we recommend using it with a good feature selection process for the following reasons: (i) As NNs based model, it is more sensitive to uninformative features compared to decision tree-based or piecewise constant approximators (Grinsztajn, Oyallon, and Varoquaux [53]). (ii) With the current implementation, the number of learned parameters is quadratic with respect to the number of the input features (for more details, please refer to Section 3.4.3).

[8] In general, these explanations do not represent causality, but rather a potential description of the association between the observed feature and the target variable

3.6 Conclusion

In many tabular learning use cases, researchers and practitioners justify the choice of full-complexity models based on their superior predictive performance over classical statistical models (e.g., linear models). The latter, on the other hand, are favored for their transparency, which usually leads to a dilemma when it comes to choosing a model for many real-life use cases. In this study, we investigated the consideration of machine learning (ML) based inherently interpretable (II) models. Through a thorough benchmarking, we showed that Inherently Interpretable. II models such as Explainable Boosting Machine, can produce a predictive performance that is usually very close to full-complexity ML models. Second, we proposed TabSRALinear, an attention based II model that demonstrates its superiority when the robustness of explanations and human knowledge incorporation are key considerations.

In this next Chapter, we will present the generalization of results presented in this Chapter to dynamic environnements where the production or test data evolves over time and may be subject to concept drift.

Chapter 4

Evaluating the Efficacy of Instance Incremental vs. Batch Learning in Delayed Label and Dynamic Environments

Real-world tabular learning production scenarios typically involve evolving data streams, where data arrives continuously and its distribution may change over time. In such a setting, most studies in the literature regarding supervised learning favor the use of instance incremental algorithms due to their ability to adapt to changes in the data distribution. Another significant reason for choosing these algorithms is *avoid storing observations in memory* as commonly done in batch incremental settings. However, the design of instance incremental algorithms often assumes immediate availability of labels, which is an optimistic assumption. In many real-world scenarios, such as fraud detection or credit scoring, labels may be delayed. Consequently, batch incremental algorithms are widely used in many real-world tasks. This raises an important question: "In delayed settings, is instance incremental learning the best option regarding predictive performance and computational efficiency?" Unfortunately, this question has not been studied in depth, probably due to the scarcity of real datasets containing delayed information. In this Chapter, we conduct a comprehensive empirical evaluation and analysis of this question using a real-world fraud detection problem and commonly used generated datasets. Our findings indicate that instance incremental learning is not the superior option, considering on one side state-of-the-art models such as Adaptive Random Forest (ARF) and on the other side batch learning models such as XGBoost. Additionally, when considering the interpretability of the learning systems, batch incremental solutions tend to be favored.

4.1 Introduction

Monitoring and adapting/updating machine learning models is a clear requirement in many real-world problems. This is mainly due to the changes that may occur in the data distribution over time, violating the identical distribution hypothesis (between the training and production/test datasets) usually made in a classical offline machine learning setting. Additionally, it is impractical to store all data in memory indefinitely in tasks where data is generated at high speed. Over the last two decades, many efforts have been made to develop learning systems that respect these requirements and are able to provide predictions in real-time. Within these solutions, we can differentiate instance incremental models, which have the capability to update their weights or architecture using just one instance. Most state-of-the-art models in this category are based on the Hoeffding Tree (Domingos and Hulten [99]) and are often equipped with drift or change detection mechanisms such as ADaptive WINdowing (ADWIN) (Bifet and Gavalda [96]). We also have batch incremental learning systems in which the stream datasets are stored in chunks or batches that are further used to learn a new static model or update the old one. An important consideration in the evolving data stream is label delay (Plasse and Adams [133], Grzenda, Gomes, and Bifet [134], Gomes et al. [132], and Haug, Tramountani, and Kasneci [137]), where labels can be received with important delays depending on the task. For example, in card fraud detection, the true nature of a card transaction (fraudulent or genuine) is only known after some days or weeks, unless the card owner notices an anomaly on its card and reports it immediately to the bank. In such a situation, it is unavoidable in a supervised paradigm to store observations waiting for their labels; otherwise, semi-supervised or unsupervised systems should be considered (Gomes et al. [132]). Besides labeling delay and for most real-world problems, it is often a requirement to store observations for some fixed periods. For instance, banks are legally obliged to store transaction information for several months and should make it available upon customer request. In this study, we investigate whether instance incremental solutions remain the best option in such delayed situations by considering their predictive performance and computational efficiency.

Moreover, the interpretability of automatic decision-making systems is required in many real work cases (e.g., GDPR, Article 22, IA Act in Europe). Regarding learning with evolving data streams, interpretability involves understanding the decisions made by the model/system, as well as changes within the data distribution and the model itself (Haug, Broelemann, and Kasneci [122]). We show that state-of-the-art batch incremental interpretable models outperform instance incremental models in terms of accuracy and explain why they might be preferable for inherent interpretability for evolving data streams. Overall, we summarize the contributions of this Chapter as follows:

- We designed a realistic supervised evaluation framework based on interleaved chunks, combined with label delays (Section 4.2).
- We empirically compare instance incremental, and batch incremental algorithms on the designed evaluation framework and revealed the superior performance of the latter in delayed settings using benchmark tabular stream datasets and real-world fraud modeling problem (Section 4.3).
- We analyze the performance of batch incremental models and demonstrate the im-

portance of storing past observations whenever possible, especially for tasks where the target event is rare, such as fraud detection (Section 4.3.3).

4.2 Problem formalization

4.2.1 Label delay

We consider a supervised learning setting for a continuous data stream $S_i = \{(\mathbf{x}_i, y_i)\}$ with $i = 1, ..., T$ where $T \longrightarrow \infty$, i is the unique identifier of each instance. For binary classification $y_i \in \{0, 1\}$ or $y_i \in \mathbb{R}$ for regression tasks. The input feature vector is $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$ (we do not consider settings where the input dimension changes over time).

In a delayed setting, labels are available with some delay. We denote by $dt(\mathbf{x}_i)$ the timestamp when observation $\mathbf{x}_i$ becomes available, $dt(y_i)$ is the timestamp of the corresponding label's arrival, and $\Delta t(\mathbf{x}_i) = dt(y_i) - dt(\mathbf{x}_i)$ is the label delay. Obviously, $\delta t(y_i) \geq dt(\mathbf{x}_i)$ i.e., $\Delta t(\mathbf{x}_i) \geq 0$ and $\Delta t(\mathbf{x}_i) = 0$ corresponds to the particular case where y_i becomes immediately available (before the observation following $\mathbf{x}_i$). In general, the units of delay are seconds, minutes, days, and months, but there are also simplified settings where the delay is evaluated in terms of instances (Gomes et al. [97] and Grzenda, Gomes, and Bifet [134]). In such delayed settings, it is unavoidable to store observations until their labels become available. More precisely, once available, label y_i can be used to update the metric and couple $\{(\mathbf{x}_i, y_i)\}$ to update the learning model. Following Grzenda, Gomes, and Bifet [134], we denote by $S_i = \{(\mathbf{x}_i, ?)\}$ at the timestamp $dt(\mathbf{x}_i)$, i.e., when the label is not yet available and $S_i = \{(., y_i)\}$ at the timestamp $dt(y_i)$, i.e., when the label becomes available.

4.2.2 Predictive performance evaluation methodology

Algorithm 1 summarizes the learning process for instance incremental models, and algorithm 2 summarizes the one for batch incremental models.

The inference is made in real-time, i.e., incrementally per instance, as in common stream learning, but the evaluation metric is applied to a chunk or batch of predictions (with potentially varying sizes). The batch is chosen such that there are sufficient labeled observations for consistent performance evaluation (we refer to this $\mathrm{B\_label}[id_B].isFull()$ line 13 Algorithm 1 and line 11 Algorithm 2). For real-world problems, this batch is usually defined in terms of time, for example, one day, one month of predictions, or one year, depending on the speed of the stream and the label delay. For the instance incremental and batch incremental methods, the observations are stored in a buffer B_X, waiting for their labels, which may be available after some delay. The main difference in data storage between the two learning strategies lies in the updating requirements. For instance incremental learning, every observation $\mathbf{x}_i$ is stored in the buffer, and once its label y_i becomes available, the pair $(\mathbf{x}_i, y_i)$ is used immediately to update the model. For batch incremental learn-

Algorithm 1 Evaluation for instance incremental models

Require: S_1, S_2, S_3,... - data stream, f-initial pretrained model, $eval\_metric$ - evaluation metric e.g., $Accuracy$, AUCROC, AUCPR.

Ensure: B_X, B_label, B_pred - buffer for storing observations, labels, predictions respectively

for $S_i, i = 1, ...$ **do**
  $i \leftarrow get\_index(S_i)$
  $id_B \leftarrow get\_batch(S_i)$
  if $S_i = \{(\mathbf{x}_i, ?)\}$ **then** ▷ New unlabeled instance arrived
    $y_{pred} \leftarrow f(\mathbf{x}_i)$ ▷ Get the real-time prediction
    $\text{B\_X}[id_B][i].add(\mathbf{x}_i)$ ▷ Add the observation to the buffer
    $\text{B\_pred}[id_B][i].add(y_{pred})$
  else if $S_i = \{(., y_i)\}$ **then** ▷ The label of the instance i becomes available
    $\text{B\_label}[id_B][i].add(y_i)$
    $\mathbf{x} \leftarrow \text{B\_X}[id_B][i].pop()$ ▷ get the observation and remove it from the buffer
    $train(f, \{(\mathbf{x}, y_i)\})$ ▷ Update the model
  end if
  if $\text{B\_label}[id_B].isFull()$ **then** ▷ There is enough observations for evaluation
    $\text{result} \leftarrow eval\_metric(\text{B\_label}[id_B], \text{B\_pred}[id_B])$
    $display(id_B, \text{result})$
    $update(\text{B\_label})$ ▷ update the buffer by deleting old labels
    $update(\text{B\_pred})$
  end if
end for

ing, observations are stored until there are enough labels for retraining or updating the batch model (line 14 Algorithm 2). Therefore, batch incremental systems may require more storage resources than instance incremental ones. However, in situations where delay is significant and variable, the storage requirements can be quite similar, as highlighted in the *fraud dataset* (Section 4.3.1). Without loss of generality, we assume in this study that the number of labeled instances required for retraining or updating the supervised batch models is equal to or greater than the number required for periodic evaluation.

Finally, our evaluation framework includes the option of using a pre-trained model at the beginning of the stream for both instance incremental and batch incremental learning. For many real-world problems, an initial model is typically optimized offline using data collected over time before deploying a learning system. Therefore, the stream data represents the starting point of production or deployment. This practice is becoming common in the incremental learning literature, where a fraction of data is used either to train an initial model before the stream begins [122] or for hyperparameter optimization (Montiel et al. [117]).

Algorithm 2 Evaluation for batch incremental models

Require: S_1, S_2, S_3,... - data stream, f-initial pretrained model, $eval\_metric$ - evaluation metric e.g., $Accuracy$, AUCROC, AUCPR.

Ensure: B_X, B_label, B_pred, - buffer for storing observations, labels, predictions respectively

1: **for** $S_i, i = 1, ...$ **do**
2: $i \leftarrow get\_index(S_i)$
3: $id_B \leftarrow get\_batch(S_i)$
4: **if** $S_i = \{(\mathbf{x}_i, ?)\}$ **then**
5: $y_{pred} \leftarrow f(\mathbf{x}_i)$ ▷ Get the real-time prediction
6: $\text{B\_X}[id_B][i].add(\mathbf{x}_i)$ ▷ Add the observation to the buffer
7: $\text{B\_pred}[id_B][i].add(y_{pred})$
8: **else if** $S_i = \{(., y_i)\}$ **then**
9: $\text{B\_label}[id_B][i].add(y_i)$
10: **end if**
11: **if** $\text{B\_label}[id_B].isFull()$ **then**
12: $\text{result} \leftarrow eval\_metric(\text{B\_label}[id_B], \text{B\_pred}[id_B])$
13: $display(id_B, \text{result})$
14: **if** $\text{B\_label}.isFullForTrain()$ **then**
15: $i_{train}, labels \leftarrow \text{Y\_label}.get\_train\_batch()$
16: $X \leftarrow \text{B\_X}.get\_batch\_data(i_{train})$
17: $train(f, \{X, labels\})$ ▷ Update the model
18: $update(\text{B\_X})$ ▷ update the buffer by deleting unnecessary observations
19: $update(\text{B\_label})$
20: $update(\text{B\_pred})$
21: **end if**
22: **end if**
23: **end for**

4.3 Experiment analysis

4.3.1 Experiment setup

Datasets

We consider two types of datasets: (1) benchmark datasets for generated binary classification, well-known in the literature on evolving data streams (Gomes et al. [97], Montiel et al. [117], Gunasekara et al. [106], and Haug, Broelemann, and Kasneci [122]), denoted as generated benchmark, and (2) a real-world bank transfer Fraud detection dataset.

Generated benchmark. The datasets in this benchmark are publicly available and are summarized in Table 4.1:

- **AGR**$_a$ (Gomes et al. [97], Montiel et al. [117], and Gunasekara et al. [106]): This dataset includes six discrete features and three continuous features. Instances are categorized into two classes using various functions, some of which adhere to decision rules, making it conducive to decision tree analysis. An abrupt drift occurs after every 250,000 instances (three in total).
- **AGR**$_g$ (Gomes et al. [97], Montiel et al. [117], and Gunasekara et al. [106]): Similar to AGR$_a$. Here, a gradual drift is used.
- **HYPER**$_f$ (Gomes et al. [97], Montiel et al. [117], and Haug, Broelemann, and Kasneci [122]): Simulates an incremental (fast) drift by changing the equation of hyperplane separating the two classes over time. This dataset is, therefore, linear model friendly. The number of features is set to 10.
- **SEA**$_a$ (Gomes et al. [97], Montiel et al. [117], and Haug, Broelemann, and Kasneci [122]): It comprises three continuous features (x_1, x_2, x_3), with only the first two being pertinent to the target class. The first two dimensions are divided into four blocks. Within each block, an instance is assigned to class 1 if $x_1 + x_2 \leq \theta$, and to class 0 otherwise, with θ taking values from the set $8, 9, 7, 9.5$. Additionally, an abrupt drift occurs after every 250,000 instances (three in total).
- **SEA**$_g$ (Gomes et al. [97] and Montiel et al. [117]): Similar to SEA$_a$. Here, a gradual drift is employed rather than an abrupt one.

Induce label delay in the generated benchmark. First, 10% of each dataset (100,000 instances) is reserved as offline collected data, used for hyperparameter tuning (Montiel et al. [117]) and pretraining an initial model (Haug, Broelemann, and Kasneci [122]) in both instance incremental and batch incremental learning. Therefore, 90% of each dataset is used for stream (online) evaluation. Each evaluation batch (Algorithm 2) contains approximately 1% of the dataset. Specifically, the evaluation batch size is generated following a Poisson distribution with a mean of 10,000. This varying evaluation batch size effectively reflects real-world settings, where monitoring logs of deployed learning systems must be

Table 4.1: *Tabular benchmark for incremental algorithms evaluation. N: number of instances, p: number of features, Type: **S**ynthetic and **R**eal. Drift **A**: abrupt, **G**: gradual, $\mathbf{I}_f$: incremental fast, and ?: unknown, MC: Minority Class*

Datasets	N	p	# classes	Type	Drift	MC (%)
AGR_a	1M	9	2	S	A	47.17
AGR_g	1M	9	2	S	G	47.17
$HYPER_f$	1M	10	2	S	I_f	50. 00
SEA_a	1M	3	2	S	A	40.09
SEA_g	1M	3	2	S	G	40.09
Fraud	$\sim$ 6.5 M	18	2	R	?	<0.10

displayed periodically (every day, week, or year, depending on the stream's speed). However, the number of instances may vary from one period to another.

In this study, we generated the label delay by assuming it is stochastic and follows a Poisson distribution. More precisely, we assume that the delay $\Delta t(\mathbf{x_i})$ for each instance observation $\mathbf{x_i}$ follows a Poisson distribution with a mean of $\alpha \times 10,000$ instances, where $\alpha \in \{0, 0.1, 1, 2, 3, 4, 5, 6, 7\}$.

Fraud dataset. The task consists in using known information about fraudulent bank transfer scenarios (via supervised machine learning) to assign a fraud score to each newly added IBAN (International Bank Account Number). Specifically, the goal is to predict the probability that a new IBAN will be used for fraudulent transfers within the next 30 days. The motivation for investigating instance incremental learning is the variation in label delay based on class labels. Most fraudulent IBANs are identified within a few days, while the remaining IBANs are automatically labeled as genuine after 30 days. Hence, one might question whether quickly adapting the learning model with these fraudulent instances could yield better performance compared to a batch learning strategy, where the model is updated only after collecting labels for at least a month. For example, in batch learning, labels from September are collected by the end of October, and the model is updated in early November, whereas instance incremental learning updates the model with September's fraudulent data as soon as it becomes available. A key challenge is the class imbalance, where only about 0.10% of instances are fraudulent on average. As a result, it is necessary to store at least 99.90% of instances for 30 days due to label delay. Additionally, unlike datasets in the *Generated benchmark*, this dataset may experience various types of drift, such as gradual drift combining abrupt changes, and some concepts may reoccur over time. The sample used in this study comprises approximately 6.5 million observations collected from September 2021 to August 2022. The initial three months (September, October, and November) are reserved as offline collection for hyperparameter optimization and initial model training. The stream evaluation spans eight months (from January 2022 to August 2022), with each month comprising approximately 540,000 instances.

Models benchmarked

We consider the following instance incremental models (described in Section 2.2.3):

- **Adaptive Random Forest (ARF)** [97]: This model is one of the best performing among instance incremental algorithms (Gomes et al. [97], Montiel et al. [117], and Gunasekara et al. [106]). We use ARF with the ADWIN detector (Bifet and Gavalda [96]).
- **Leveraging Bagging with Hoeffding Tree as base learner (LB_HT)** [98]: We include this model in our comparison as it is the best performing model in Read et al. [113].
- **Leveraging Bagging with Logistic Regression as base learner (LB_LR)** [98]: We explore in this study a non tree base learner for Leveraging Bagging. More specifically we use the Logistic Regression (LR) with a Stochastic Gradient Optimizer (SGD).

Alongside state-of-the-art instance incremental models, we include the following glass box algorithms for interpretability purposes, as discussed in Section 4.1:

- **Logistic Regression (LR)**: This model can be used in both batch or instance incremental models. In this study, we use it as an instance incremental model by combining it with a Stochastic Gradient Optimizer (SGD)
- **Hoeffding Tree (HT)** [99]: This model offers interpretable and adaptive decision rules by following paths in the tree, particularly if it's shallow. Thus, we cap the maximum depth at 6. Additionally, we employ Naive Bayes Adaptive for leaf prediction, which enhances the performance of the traditional HT (Haug, Broelemann, and Kasneci [122] and Gama, Rocha, and Medas [162]).
- **Hoeffding Adaptive Tree (HAT)**: The same model as the Hoeffding Tree, but with ADWIN (Bifet and Gavalda [96]) for drift handling.

For all instance incremental models, we used the Python-based package River (Montiel et al. [115]). As batch incremental models, we consider:

- **XGBoost** [36]: It is a leading model in batch learning across various real-world applications and tabular learning competitions (as shown in Chapter 3). We make XGBoost adaptive by retraining it from scratch after every new batch (Algorithm 2, line 17), referred to as **r_XGBoost**. Following Montiel et al. [117] and Dal Pozzolo et al. [131], we use a stacking of the last ($M = 3$) retrained XGBoost models, denoted as **B_XGBoost**. Although stacking may increase prediction time, it helps preserve some previously learned concepts. We don't include Adaptive XGBoost ([117]) as it was already outperformed by the optimized version in **B_XGBoost**.
- **Explainable Boosting Machine (EBM)** [163]: We include EBM in our comparison for its inherent interpretability—its shape function can be monitored to detect

changes over time—unlike XGBoost. Similarly to XGBoost, we used retraining from scratch (**r_EBM**) and stacking (**B_EBM**) to make the EBM model adaptive over time. Rather than stacking naively ($M = 3$) EBM models, which may compromise interpretability, we merged the shape functions[1] of the stacked models into a single final EBM model.

- **TabSRA** [164]: This model refers to the TabSRALinear model proposed in Chapter 3. Monitoring TabSRA's attention weights and inherent feature attributions can help to understand changes over time. We also used the retraining (**r_TabSRA**) to make the TabSRA model adaptive. In addition, we investigate the continual fine-tuning of the weights of an initially trained TabSRA model. The learning rate is reduced over time to avoid a catastrophic forgetting of previously learned concepts. This strategy is named **u_TabSRA**.

Experiment details on the generated benchmark

Tuning step. The experiments on the *Generated benchmark* are done on a 64-Core Processor CPU machine. For both instance and batch incremental models, hyperparameter optimization is conducted for each dataset using 10% of the data (considered as offline collected data). Specifically, 30 trials of Bayesian optimization (using Optuna [165]) are employed, with a maximum search time of 6 hours for each model and dataset. The best validation model (using prequential evaluation for instance incremental models and a 70/30 train/validation split for batch incremental models) is initialized at the stream's outset and updated over time for instance incremental models. To ensure a fair comparison, hyperparameter optimization is not conducted during the stream for batch incremental models; instead, the best hyperparameter configuration obtained from offline optimization is used for model retraining and updating, as outlined in Montiel et al. [117]. Furthermore, only 70% of the batch is used for training during the data stream (with 30% reserved, as commonly practiced in batch learning, for validation, *post hoc* analysis, early stopping, bias correction, etc.). For simplicity, the evaluation batch in the *Generated benchmark* matches the training batch (Algorithm 2, lines 11 and 14)[2]. For the *Fraud* dataset, we examined a scenario where the evaluation batch spans one month while the training batch covers three months (Section 4.3.3).

Finally, for all ensemble models (ARF, LB, XGBoost), the maximum number of learners is set to 100, and for tree-based models (ARF, LB_HT, HAT, HT), the maximum depth is limited to 6 (please check GitHub[3] for information regarding the hyperparameter search space).

Evaluation metric and results aggregation. For numerous binary classification tasks, such as fraud detection and online credit scoring, practitioners often seek the classifier's confidence output rather than just the binary class obtained by applying a threshold. There-

[1] https://interpret.ml/docs/python/examples/merge-ebms.html

[2] In this specific scenario, our proposed evaluation methodology aligns with interleaved chunks for batch incremental models.

[3] https://github.com/anselmeamekoe/DelayedLabelStream

fore, we suggest using AUCROC (Bradley [111]) as the evaluation metric for each stream batch. To summarize the results across all batches (90 in total), we calculate the average and standard deviation of the AUCROC values. Additionally, we include the running time as a metric to assess the computational efficiency of each algorithm.

We point out that we do not use cross-validation in our evaluation process, as we cannot alter observations' order without introducing artificial drift (Haug, Broelemann, and Kasneci [122]). There are approaches of using different seeds in model training if applicable (Gunasekara et al. [106]) or using distributed evaluation (Bifet et al. [166]). However, they are very computationally intensive, especially when adding hyperparameters optimization steps. For more details about the statistical significance in evaluating data streams, we refer the interested readers to Bifet et al. [166], Haug, Tramountani, and Kasneci [137], and Haug, Broelemann, and Kasneci [122].

We analyze in the next Section the results on the *Generated benchmark*, which contains homogeneous datasets with well-known drift types and difficulties.

4.3.2 Results on the generated benchmark

Do instance incremental algorithms really outperform batch incremental ones?

We analyze the predictive performance (Table 4.2 and Figure 4.1) by considering first a no-delay setting (Read et al. [113], Gomes et al. [97], Montiel et al. [117], and Gunasekara et al. [106]), however, using an initial tuning step for both instance and batch incremental as described in Section 4.3.1 in contrast to previous work. The key findings are:

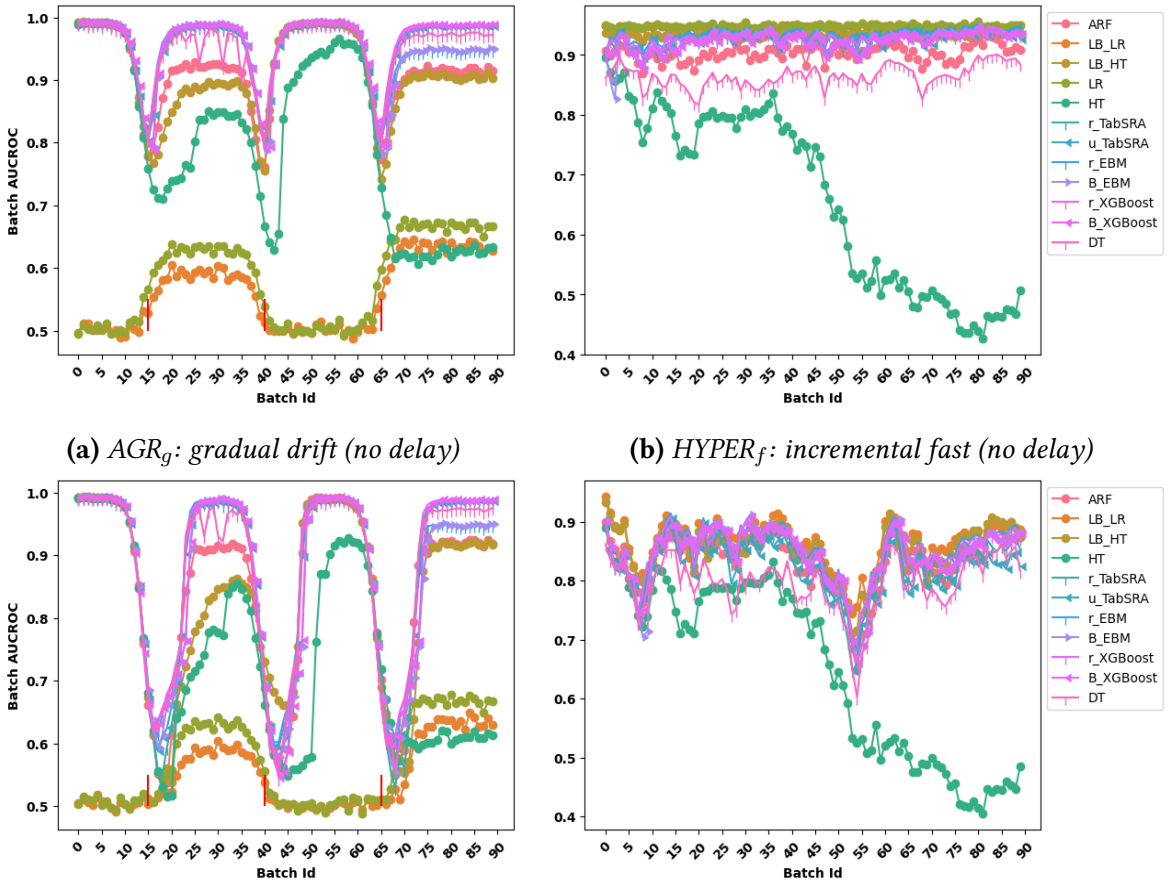
- With an initial tuning step, the best batch incremental system with an *appropriate* learning batch often achieves a comparable or even superior performance to instance incremental systems for known unique type of drift (including abrupt, gradual, incremental).
- When considering interpretable solutions, batch incremental models (such as EBM and TabSRA) offer the advantage of less frequent changes in weights and architecture, making them easier for humans to follow. In addition, these models generally outperform instance incremental models in terms of predictive performance.

An important observation regarding the *Generated benchmark* is the impact of the inductive biases of the learning model. Specifically, linear-based models like LR and LB_LR perform poorly on piece-wise constant-like datasets (AGR_a, AGR_g). Conversely, tree-based models (DT, HAT, HT) are less effective for modeling continuous (linear) data streams like $HYPER_f$, although ensembling mitigates this issue (for XGBoost, EBM, LB_HT, ARF). Hence, we emphasize the importance of alerting readers/practitioners that relying solely on average performance measures from *synthetic benchmark*, which encompasses varying (unequal) types of bias/difficulty, to select stream learning approaches may be misleading for real-world problems.

Regarding the adaptation over time, the Hoeffding Tree (HT) appears to be more affected by changes (Figure 4.1a and 4.1b) arguably due to its lack of explicit drift handling

Table 4.2: *Predictive performance on the generated benchmark in **no delay setting**. Mean and standard deviation AUCROC (%), reported for a data stream split in 90 batches. Italic highlights the best performance when comparing inherently interpretable (II) models, and bold is used for the overall best-performing model. We also report the average (resp. the average normalized by the best AUCROC of the given dataset) across the 5 stream datasets denoted as Avg (resp. N_Avg), as well as the average rank (Avg_Rank). Type: **II**=Inherently interpretable, **FC**= Full complexity model*

Model	Type	AGR_a	AGR_g	$HYPER_f$	SEA_a	SEA_g	Avg	N_Avg	Avg_Rank
LB_HT	FC	94.54 $\pm$4.43	90.96 $\pm$6.74	93.41 $\pm$0.39	88.90 $\pm$0.89	88.87 $\pm$0.83	91.34	98.01	6.20
LB_LR	FC	56.40 $\pm$5.94	55.92 $\pm$5.59	94.80 $\pm$0.23	88.83 $\pm$0.88	88.78 $\pm$0.84	76.94	83.14	9.00
ARF	FC	94.82 $\pm$4.12	92.12 $\pm$6.12	90.36 $\pm$1.29	88.92 $\pm$0.89	88.89 $\pm$0.84	91.02	97.67	6.00
r_XGBoost	FC	**97.36** $\pm$8.16	**96.25** $\pm$4.80	92.89 $\pm$0.98	88.80 $\pm$0.86	88.75 $\pm$0.87	**92.81**	**99.52**	5.40
B_XGBoost	FC	96.59 $\pm$9.13	95.91 $\pm$5.72	92.27 $\pm$1.40	88.89 $\pm$0.88	88.84 $\pm$0.86	92.50	99.21	5.80
HT	II	86.50 $\pm$12.52	79.77 $\pm$13.69	65.80 $\pm$15.17	88.28 $\pm$0.50	88.24 $\pm$0.49	81.72	87.92	11.60
HAT	II	93.86 $\pm$4.98	89.72 $\pm$7.23	89.98 $\pm$1.52	88.38 $\pm$0.90	88.21 $\pm$0.75	90.03	96.62	10.80
LR	II	58.34 $\pm$7.49	58.12 $\pm$6.98	***94.85*** $\pm$0.23	88.90 $\pm$0.88	88.86 $\pm$0.84	77.82	84.05	7.00
DT	II	96.12 $\pm$7.70	95.09 $\pm$4.90	86.52 $\pm$1.86	87.87 $\pm$1.29	87.74 $\pm$0.89	90.67	97.25	9.80
r_EBM	II	95.84 $\pm$8.29	95.21 $\pm$4.86	93.43 $\pm$0.97	88.91 $\pm$0.90	88.87 $\pm$0.86	92.45	99.16	4.40
B_EBM	II	95.49 $\pm$9.49	94.84 $\pm$5.69	92.41 $\pm$1.90	***88.94*** $\pm$0.90	***88.90*** $\pm$0.86	92.12	98.81	4.80
r_TabSRA	II	97.10 $\pm$8.23	95.73 $\pm$5.23	93.07 $\pm$1.20	88.89 $\pm$0.91	88.88 $\pm$0.86	*92.73*	*99.45*	***4.20***
u_TabSRA	II	*97.16* $\pm$7.79	*95.75* $\pm$5.22	92.78 $\pm$1.20	88.74 $\pm$1.30	88.83 $\pm$0.85	92.65	99.36	6.00



(a) *AGR_g: gradual drift (no delay)*

(b) *$HYPER_f$: incremental fast (no delay)*

(c) *AGR_g: gradual drift (delay factor=7)*

(d) *$HYPER_f$: incremental fast (delay factor=7)*

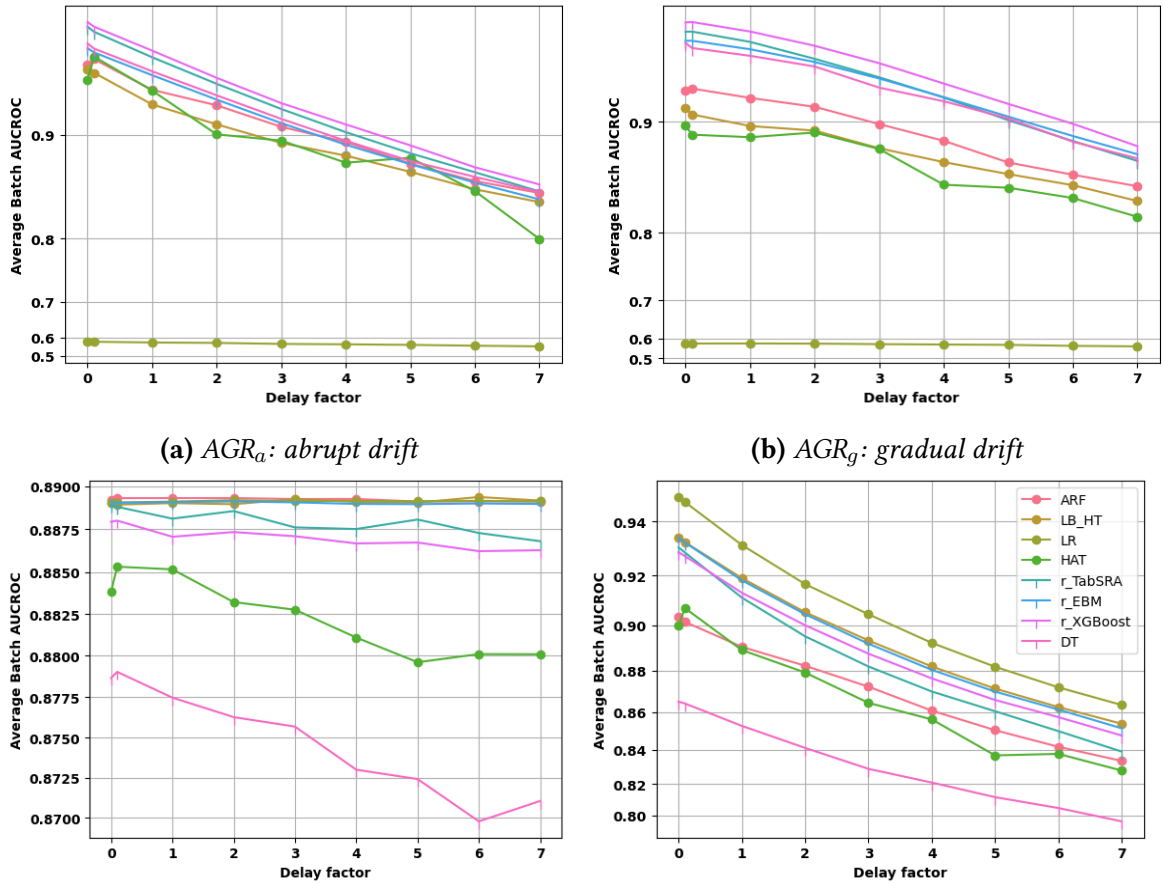
Figure 4.1: *AUCROC over time. The x-axis is the identifier (ID) of the batch/chunk, which is up to 90. The delay factor 0 corresponds to the no-delay setting, and 7 corresponds to the Poisson stochastic delay of average of 70,000 instances. The red vertical line indicates the change point for the SEA dataset Figure 4.1a and 4.1c.*

compared to the HAT (Hoeffding Adaptive Tree). We recall that the maximum depth of the tree is capped to 6 (for DT, HT, HAT) during the hyperparameters optimization; otherwise, it would be challenging to consider them as inherently interpretable. Overall, we observe that batch-based solutions (whether inherently interpretable or black box) can adapt to changes through incremental retraining, even without explicit drift management.

An important observation from this experimentation is the hyperparameter sensitivity of tree-based instance incremental algorithms. As long as the concept[4] on which the parameters were tuned is still valid, instance incremental models with good inductive biases (tree-based ones such as ARF, LB_HT) manage to match the performance of batch incremental counterparts (highlighted by the Figure 4.1a for AGR_g dataset with Batch Id $\in [0, 15]$). However, after the change or drift (Batch Id $\in [15, 40]$) these models struggle to converge, unlike batch learners. Yet, once the initial concept reoccurs (Batch Id $\in]40, 65[$) the models (instance incremental as well) converge to perfect AUCROC again.

Finally, we observe that for this *Generated benchmark*, employing stacking/updating of

[4]The function 5 of the Agrawal generator.



(a) *AGR_a: abrupt drift*

(b) *AGR_g: gradual drift*

(c) *SEA_a: abrupt drift*

(d) *$HYPER_f$: I_f*

Figure 4.2: *Impact of the label delay on the predictive performance. The x-axis indicates the delay factor ranging from 0 (no delay) to 7 (the Poisson stochastic delay of average 70,000 instances). The best model for each category is reported to make the figures more clearer (LB_LR, HT, u_TabSRA, B_EBM, B_EBM) as discarded. I_f = incremental fast drift.*

models learned over different periods (B_XGBoost, B_EBM, u_TabSRA) does not yield improved predictive performance; instead, it increases computational resources or running time (Table 4.3). We argue that this is due to the simplicity of the datasets, where typically small learning datasets (roughly 10,000 instances) suffice to learn the underlying concept. However, for more complex tasks, we anticipate that stacking may prove beneficial, as demonstrated in Section 4.3.3 for the ***Fraud*** dataset.

How does the label delay impact incremental algorithms?

After comparing algorithms in no delay settings, it is crucial to consider the delay setting, which involves answering the following questions: (1) *How does the label delay impact the performance of the stream learning system?* (2) *Is instance incremental learning the best option regarding the predictive performance in such delayed settings?*
From Figure 4.1, we can notice that the labeling delay impacts the recovery or adaptation

of both instance and batch learners to changes, consequently affecting overall performance (Figure 4.2), consistent with findings in Gomes et al. [97] and Gomes et al. [132]. However, this impact tends to be limited for low-severity drift/changes. For the SEA_a (Figure 4.2c) dataset for instance (where the changes in the simulated decision functions are not significant), the average drop in AUCROC in delayed context seems very low, even negative for some models (for both gradual and abrupt drift) in contrast to other generated datasets (AGR_g, AGR_a, $HYPER_f$).

Besides the impact of the labeling delay on all stream learning models, we can notice (Figure 4.2) that the best-performing batch learner usually provides similar or superior results compared to its instance counterpart regardless of the length of the delay.

Computational efficiency

In this part, runtimes of the compared models during the stream are analyzed, encompassing retraining and inference, as a measure of computational efficiency. We recognize that relying solely on runtime may not suffice, as it can vary with implementation. For tree-based models, parameter or node count is sometimes considered (Montiel et al. [117]), yet comparing computational efficiency across different model classes (e.g., neural networks versus tree-based models) using this metric is challenging.

As shown in Table 4.3, state-of-the-art instance incremental learners like ARF are over twice as costly as batch counterparts such as r_XGBoost. We argue that this is due to the explicit drift monitoring process employed in instance-based learning models. Furthermore, employing batch model stacking (B_XGBoost, B_EBM) tends to notably increase inference time and, consequently, the runtime of the stream.

Concerning inherently interpretable solutions, batch incremental options like EBM and TabSRA exhibit longer runtimes compared to instance incremental models (HT, HAT, LR). It is worth noting that the significant runtime difference between Decision Tree (DT) and shallow instance incremental learners (HT, HAT, LR) is arguably attributed to the data filtering process required for selecting new batches for model retraining/updating (Ligne 15 Algorithm 2).

4.3.3 Results on the Fraud dataset

In this Section, we compare the models using the *Fraud dataset* (described in Section 4.3.1). Unlike the *Generated benchmark*, this dataset displays a significant class imbalance (Table 4.1), a common situation in real-world applications such as fraud detection. For this dataset, we also consider both instance and batch incremental models presented in Section 4.3.1 except LB_LR which tends to be particularly ineffective at modeling non-linear functions and has high runtime demands (Table 4.3). Furthermore, we incorporate two learning strategies for batch learners:

- **Static**: All batch learners are optimized using the initial offline collected datasets of three months but are not retrained/updated during the stream. This approach saves

Table 4.3: *Running time of algorithms (s). The average (Avg.) and standard deviation (Std.) time (the smaller, the better) is reported from 9 delayed configurations (factor).*

Model	Avg. runtime	Std. runtime
LB_HT	5837.58	4428.63
LB_LR	7279.24	4273.45
ARF	3702.08	2622.41
r_XGBoost	1688.97	210.53
B_XGBoost	5101.62	1186.93
HT	60.07	11.90
HAT	248.20	63.00
LR	83.93	29.07
DT	976.94	47.11
r_EBM	1456.71	403.83
B_EBM	4167.64	3244.12
r_TabSRA	2117.04	298.78
u_TabSRA	2685.77	166.45

from retraining and validation costs, but may lead to significant performance drop after severe changes or drifts. Models trained with the static strategy will be prefixed with **static_**.

- **Propagate**: The instance from past fixed number of chunks (months in our case) are propagated and combined with the currently available one for retraining/updating the batch model. This approach may be very beneficial for tasks where certain groups or categories are very underrepresented or target event is very rare (Dal Pozzolo et al. [131]). However, this approach may add an additional cost for storing past batches and may impair the adaptation to the newest concept, especially when the number of past-considered chunks is important. In this study, we explore the trade-off by investigating the combination of the two previous months (chunks) with the current one. This situation is equivalent to the case where the learning chunk is three times the evaluation batch (Algorithm 2). Models trained with this strategy will be prefixed with **propagate_**.

Similarly to the *Generated benchmark*, we use an initial optimization step for the *Fraud* dataset, as highlighted in Section 4.3.1. We optimize the batch incremental algorithm using 10 different parameter configurations and the instance incremental algorithms using 15 configurations[5] including resampling methods (Ferreira et al. [125] and Aguiar, Krawczyk, and Cano [167]). We consider undersampling for ARF and LB_HT, which already have a long runtime due to online bagging, and investigate undersampling for HT and HAT. For LR, we examine cost sensitivity by modifying the misclassification cost of positive examples in

[5]The experiment on this dataset is conducted on a business Google Cloud Platform provided by our data provider, where running very long experiments is infeasible. Therefore, we used a maximum of 15 iterations (instead of 30 for the generated datasets) as a trade-off between tunability and predictive accuracy

Table 4.4: *Predictive performance on the fraud dataset. The evaluation is done in batches of one month, starting from January 2022 to August 2022, while predictions are made in real-time (one instance at a time). AUCPR is used as a metric, and we also report the Average (Avg) and Standard deviation (Std) over the 8 months. Italic highlights the best performance when comparing inherently interpretable models (HT, HAT, LR, DT, EBM, TabSRA) models, and bold is used for the overall best-performing model.*

Model	Jan	Fev	Mar	Apr	May	Jun	Jul	Aug	Avg	Std
LB_HT	5.26	7.32	10.89	6.76	10.18	8.27	10.65	19.27	9.83	4.03
ARF	9.38	14.08	22.73	13.12	12.50	12.27	9.90	19.87	14.23	4.39
static_XGBoost	**15.68**	23.12	**33.76**	14.69	19.48	17.31	12.75	22.55	19.92	6.25
r_XGBoost	**15.68**	23.00	24.96	17.53	18.23	20.54	14.65	18.60	19.15	3.29
B_XGBoost	**15.68**	23.00	24.49	**18.44**	21.02	23.05	16.36	23.78	20.73	3.24
propagate_XGBoost	**15.68**	**27.69**	32.04	18.24	**22.85**	**23.80**	**18.05**	**28.92**	**23.41**	5.46
HT	4.67	10.03	15.27	7.20	6.43	4.10	6.11	16.63	8.80	4.46
HAT	1.49	0.55	1.77	1.36	0.87	1.10	2.48	5.99	1.95	1.62
LR	6.91	9.11	12.25	6.74	7.32	5.77	7.52	15.79	8.93	3.19
static_DT	7.39	10.88	12.01	4.98	4.71	4.31	4.13	6.23	6.83	2.86
r_DT	7.39	11.31	8.07	7.05	7.52	8.13	7.62	6.59	7.96	**1.35**
propagate_DT	7.39	13.82	14.68	10.72	10.74	9.15	8.63	19.83	11.87	3.80
static_TabSRA	12.60	17.91	23.69	11.06	13.62	8.91	9.81	20.94	14.82	5.08
r_TabSRA	12.60	14.17	22.68	15.03	15.21	13.95	11.47	20.19	15.66	3.58
u_TabSRA	12.60	19.10	26.38	16.65	18.73	19.01	13.99	19.56	18.25	3.91
propagate_TabSRA	12.60	19.24	*26.74*	15.52	*18.88*	20.56	14.41	22.86	18.85	4.35
static_EBM	*13.36*	20.80	25.94	12.26	14.85	16.73	9.27	14.24	15.93	4.91
r_EBM	*13.36*	22.03	22.92	15.33	16.85	18.06	12.72	17.49	17.35	3.45
B_EBM	*13.36*	20.80	25.94	*17.21*	17.55	21.55	15.06	22.06	19.04	3.89
propagate_EBM	*13.36*	*22.88*	24.92	16.72	17.44	*21.59*	*15.14*	*26.20*	*19.78*	4.45

the hyperparameter configuration[6]. The performance of instance incremental models was particularly poor without explicitly handling class imbalance.

As shown in Table 4.4, the best batch solution, XGBoost, consistently outperforms (over the 8 months) the best instance incremental solution, namely Adaptive Random Forest (ARF). Similarly, the Explainable Boosting Machine (EBM) provides the best average AUCPR among the inherently interpretable solutions, surpassing all the instance incremental solutions. Besides the overall superior predictive performance of batch incremental solutions in this problem, the **propagate** strategy, which consists in using the tree last months (chunks) of labeled data, tends to provide the best results. For XGBoost, the average AUCPR gain is up to 22% compared to the retrain strategy (r_XGBoost) and 12% compared to the stacking strategy. We argue that this is because the target event (fraudulent examples) is rare. Consequently, using only one chunk (one month) of observation should not

[6] https://riverml.xyz/0.9.0/examples/imbalanced-learning/

be enough to retrain a consistent learning model, and finding the best weighting combination for the stacking approach is not an easy task. This finding confirms the results of Wang et al. [118] where the author demonstrates that varying the number of observations in the chunks (from 3,000 to 12,000 observations) increases linearly the benefit (the number of frauds detected) by the learning model. Particularly for the fraud dataset investigated in our study, the number of fraudulent examples seen during the training or the updating of the model is very crucial for the predictive performance, highlighted by the fact that a static approach with 3 months of training data (static_XGBoost) outperforms on average the periodical retraining with 1 month (chunk) of observations (r_XGBoost). It also indicates that some concepts may reappear over time (e.g., static_XGBoost outperforms propagate_XGBoost for March). Overall, the results on this dataset suggest that storing some past observations may be advantageous for the streaming learning model.

We also note an intriguing result (Table 4.4) for inherently interpretable instance incremental models: Hoeffding Tree (HT) outperforms the HT equipped with the ADWIN drift detector (HAT) in contrast to the results on the *Generated benchmark* (Table 4.2). We argue that for such a dataset where the target event of interest is very rare (highly imbalanced dataset), explicit drift handling may be biased toward negative examples. This is because many statistical drift detectors rely on accuracy metrics or error rates. Consequently, the HAT model may discard some parts (or branches) of the tree that are specialized in identifying concepts (fraudulent schemes) that may reoccur over time.

4.4 Conclusion and Discussions

We proposed a thorough comparison between batch and instance incremental models by examining the usefulness of the latter in situations where labels become available with varying delays, and interpretability is crucial. Our findings on commonly used synthetic data streams demonstrate that the best batch incremental approach (fully complex or inherently interpretable), combined with an effective updating strategy, provides similar or better performance than instance incremental counterparts. Additionally, the superior performance of batch learners is illustrated using a real-world fraud detection problem where the target class is scarce. We explain why, for such a dataset, it is necessary to store some observations while waiting for the labels due to the delay. Furthermore, we demonstrate how storing some old but limited chunks of observations can benefit batch incremental models, leading to up to a 22% relative improvement in AUCPR compared to using only one chunk. Although we illustrate the usefulness of batch incremental solutions, we do not cover the choice of the optimal batch (chunk) size, as it may depend on several factors (e.g., the frequency of the target event, bias handling for underrepresented groups) that we leave for future study. Nevertheless, we believe that choosing the learning batch based on evaluation date (daily, monthly) can be a good starting point, as illustrated with the fraud dataset.

We hope the results of this chapter will encourage and guide researchers in developing new learning strategies for evolving data streams, bridging the gap between academic research and real-world business use cases.

In the next chapter, we will focus on a bank transfer applicative case study where additional operational constraints are discussed, encompassing the feature retrieval and aggregation, model validation process, drift understanding as well as the batch size selection.

Chapter 5

Machine learning in realistic fraud detection: Application to bank transfer fraud

Bank transfer remains one of the most used payment means in France (after the payment card). In 2023, roughly 312 million was lost due to fraudulent bank transfers. BPCE[1], a key payment service provider in France, consistently invests resources each year to innovate and minimize both the incidence and impact of fraudulent activities in bank transfers and other methods, such as payment cards and cheques. In this Chapter, we provide a case study on bank transfer fraud, starting from the problem of operational constraints. Subsequently, we explain the motivation behind the choice of the learning paradigm and ML algorithms. In addition, we discuss the interpretability concerns including the understanding of potential concept drifts. However, due to the confidentiality constraints, some details are not provided. We hope that these constraints will not hide the important message of this Chapter which is to make academic researchers aware of some operational and working conditions, and provide elements of answer to some questions faced by practitioners working on fraud detection.

5.1 Bank Transfer using IBAN

The International Bank Account Number (IBAN) is one of the key tools for bank transfers between institutions/companies and individuals. In the Eurozone, this type of transfer is commonly referred to as SEPA (Single Euro Payments Area).
As illustrated in Figure 5.1, an IBAN-based bank transfer is typically processed as follows:

An IBAN holder (say, IBAN A) attempts to initiate a transfer to another user (IBAN B) using their phone or transfer interface.
If the IBAN in question, i.e., IBAN B, has already been added or is known to the user, he

[1] https://www.groupebpce.com/

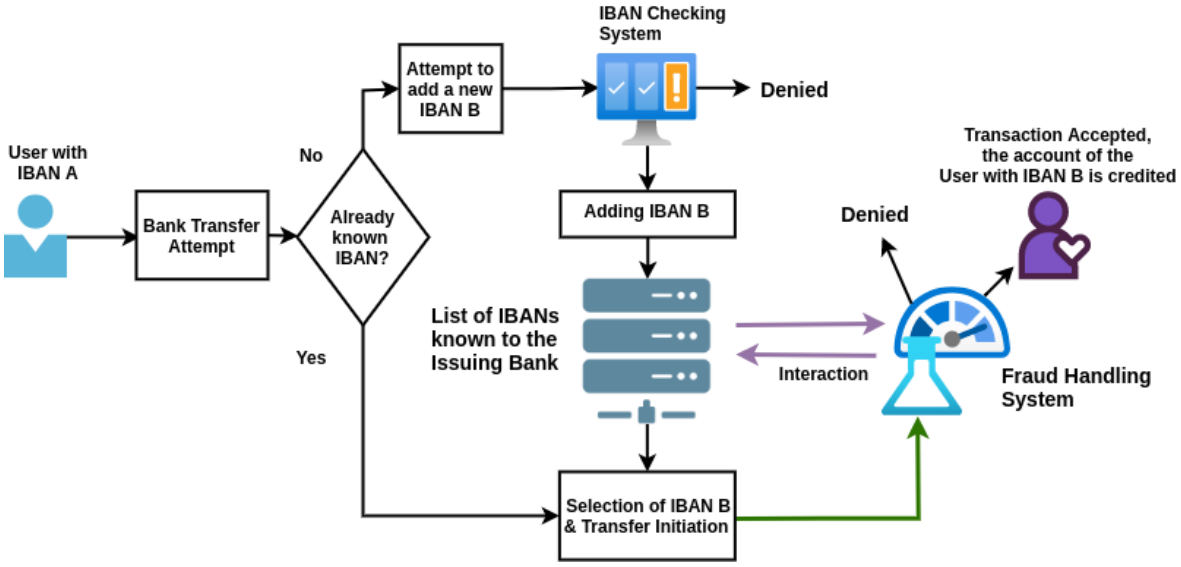


Figure 5.1: *Illustration of processing of the bank transfer with IBAN.*

can select it and initiate the transfer directly. Otherwise, he must first add the IBAN, a step verified by the *IBAN Checking Sytem*. Under certain conditions, the addition may be rejected/denied or delayed (in Chapter 4, we proposed a modeling for the machine learning module of this system).

Once the IBAN checking or verification step is validated, the IBAN B is added to the list of *IBANs known to the Issuing Bank* of IBAN A (if it is the first time it has been added by a client), and becomes visible on the interface or application of the user holding IBAN A.
At this stage, the user initiates the transfer (from IBAN A to IBAN B), and the *Fraud Handling System* (Figure 5.2) is used to either reject or validate the transfer, in which case the funds are credited to the account associated with IBAN B.
What is particularly interesting to note is that the fraud handling system can interact with the list of known IBANs, segmenting them based on their type or risk of receiving a fraudulent transfer even before a transfer initiation.

5.2 Context and Operational constraints

I completed my Ph.D. in a team that includes Data Scientists dedicated to fraud detection (cheque, bank transfert) using ML and statistical techniques. As highlighted in Chapter 1 (Figure 5.2), ML systems should collaborate with Expert systems (rules) which are traditionally used in payment fraud detection.
The main advantage of the Expert systems is that they usually produce their responses/inferences in few milliseconds and are typically transparent, especially when the number of the used rules is not too high. In addition, experts have the possibility to (manually) adjust their rules in real-time depending on noticed new types of fraud or the domain knowledge.
Regarding bank transfer fraud detection, there are some operational constraints:

- For the instantaneous bank transfer, the fraud detection system (Expert system, the

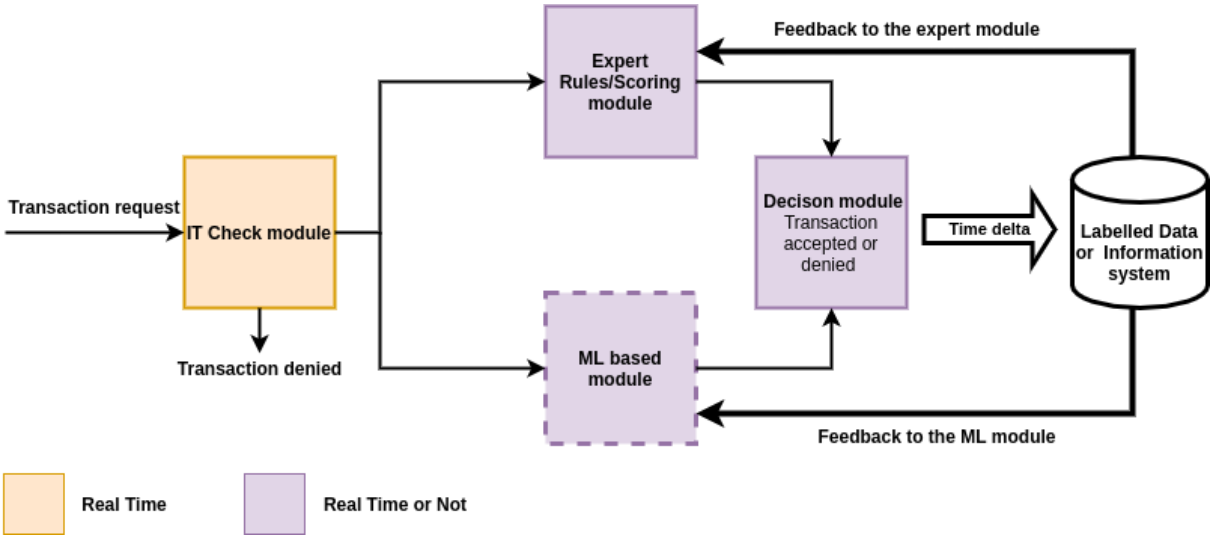


Figure 5.2: *Typical fraud handling pipeline (not exhaustive).*

ML system ...) should produce their responses in real-time i.e., in a few milliseconds. This includes the data retrieval, feature aggregations and the inference time of the ML model. For the classic bank transfers, typically, the time constraint is less and the detection system can have up to one day to respond or even more depending on the bank.

- The developed ML model should normally pass a thorough validation process to measure or quantify potential risk before the deployment or the use for business cases. The validation time can increase with the complexity (or the *blackboxness*) of the underlying ML model.

- Particularly for supervised ML systems, the labels or feedback are not obtained in real and sometimes there may be technical problems of data injection in the information/data impairing the real-time learning or new model or testing a model.

With more details concerning the first point, banks are mandated by regulatory guidelines to promptly respond to instantaneous (real-time) bank transfer transactions, often within mere seconds. Furthermore, users tend to favor banks or payment service providers that offer faster response times for these instantaneous transfers. Consequently, an ML module integrated into the payment pipeline generally has only a few milliseconds to approve or reject a transaction attempt.

Regarding the ML model validation, a bank that doesn't have a convenient validation process for its automatic decision-making systems can be subject to serious penalties from regulators.

For label or delay concern, it is infeasible for the investigator to contact every customer for every single transaction in real-time in the attempt to know the true label of the transaction. Some customers can notice fraudulent activity on their account and report it, or the investigator may contact some customers for particular transactions. Otherwise, transactions are typically considered genuine after some prescription time.

Given these crucial constraints/limitations, where addressing them could incur high costs, what is the optimal *way* for implementing an ML based fraud detection module within a bank transfer system (including instantaneous transfer)?

5.3 Construction of the target feature

In this Section, we elucidate the construction of the target or dependent feature:

y: the IBAN_x *will be involved in a fraudulent bank transfer within the upcoming 30 days*

Considering the previously described target feature leads to the use of the ML module to periodically assign (e.g., at the start of each month or upon the addition of a new IBAN) a fraud risk score to each beneficiary IBAN in the *List of known IBANs* (see Figure 5.1), operating in batch mode (i.e., with tolerable latency or time delay).
The *Fraud Handling System* can access this score in real-time through the interaction depicted in (Figure 5.1), allowing it to be utilized or retrieved as needed for every initiated transfer.

This option provides a substitute to employing ML for scoring each initiated transaction or transfer, as depicted by the green arrow in Figure 5.1. We recall that the response of the ML in this case should be in real-time or online for instantaneous bank transfers.

Choosing online or real-time transaction inference would evidently limit the use of certain sophisticated ML models, which, despite their effectiveness, exhibit substantial latency (exceeding a few milliseconds). Moreover, it is known to practitioners that the retrieval of raw features and computation of aggregated features can quickly reduce the response time of ML module unless an important computational resource is orchestrated and maintained. Furthermore, real-time inference of transactions could be more reliable if we have the capability to promptly evaluate them (as automatic decision making). Unfortunately, we are often compelled to delay the evaluation until a significant number of labels are accessible, such as a period of one month, during which numerous unverified inferences (of transactions) may be made for a single IBAN.

We argue that the advantage of the ML module should be to complement the Expert module by extracting hidden or weak signals that are not easy to extract with simple rules or simple models. We recall that the experts typically have the possibility to change and deploy their (rules) module quickly due to the drift or new fraud schema they notice.
Therefore our choice is not to model directly if a transfer (transaction) emitted from an IBAN A to an IBAN B is fraudulent or not but to interact or work on the IBAN level (Figure 5.1). That is, we model the surrogate consisting in extracting (modeling) at a fixed time, patterns (features) that may push any known receiver IBAN B to be subject to fraudulent activities within a time horizon or not. This horizon noted δ is typically the necessary time (delay) to obtain the true label of an emitted transfer.

In other words, at the time t_j a fraud likelihood score is assigned to each active IBAN in the list of known IBAN (Figure 5.1) and at time $t_j + \delta$, the true label is known. It is therefore **expected** that there is no drift in the concept learned by the ML model at least between

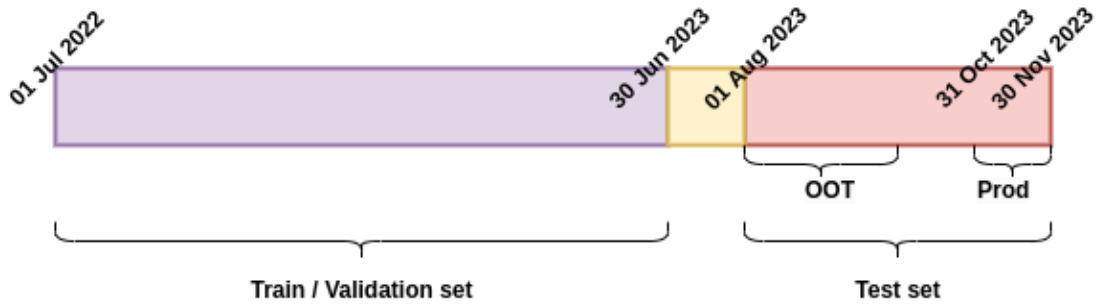


Figure 5.3: *Data splitting. OOT= Out Of Time Test data and Prod=Production. The yellow highlights the time delay for collecting the label for transactions that occur in June 2023.*

$[t_j, t_j + \delta[$ i.e., $\hat{\mathcal{P}}_{t_j}(y|\mathbf{x}) = \hat{\mathcal{P}}_{t_j+\delta}(y|\mathbf{x})$. Otherwise, the drift should be captured by the expert system, especially one resulting in newly fraudulent activities.

An additional benefit of utilizing batch inference for IBANs is the significant reduction in the number of calls to the ML module, thereby lowering inference costs, given that the average number of bank transfers an IBAN may receive in a horizon of δ typically exceeds one.

5.4 Data set

The applicative case study presented in this work is based on a real dataset covering roughly seventeen (17) months used to train and validate the ML module as depicted by the Figure 5.3. More precisely:

- Train and Validation set. Regardless of the adaptation strategies to changes or drifts, 12 months (from July 2022 to June 2023) is reserved for training the model (at least the initial one). Roughly **30%** of observations in this period is used for validating the model, for hyperparameters selection and the early stopping where required.
- Label delay. One month (July 2023) is used as the time necessary for getting all labeled information of June 2023.
- Test set. 4 months (from August to November 2023) are used to test the models and the adaptation strategie.

As highlighted in the Section 2.3.1, the used features (25 overall) are typically aggregated information for past transactions.

The original dataset considered for this application is extremely imbalanced and huge. Therefore, a sampling (undersampling) is used to speed up the training. More precisely, positive (fraudulent) IBAN are conserved and a fixed number of non-fraudulent IBAN are sampled in each category of a specific feature related to the target. As a consequence, the results presented in this Chapter cover, on average, 600,000 observations per month.
The considered models (listed below) are ones that can model to some extent non-linear

effects or class overlap (which may have a negative impact on the model performance). A post-calibration/processing step is further used on the scores provided by the model, accounting for class imbalance.

5.5 Choice of the Machine Learning model and Adaptation Strategy

After constructing the target feature and choosing the inference solution which is per batch, the next question is regarding the ML model to use. We recall that our goal is to identify which model to choose in order to bring trustworthy or at least interpretability in the use of the ML module (Figure 5.2). The interpretability is expected to help identify some hidden patterns in data and understand the decisions made by the ML model for particular examples (e.g., on request by fraud experts or business owners). At the same time, the ML module is expected to have a *good* predictive performance and maintain it over time; otherwise, it cannot be used for fraud detection (in risk of generating too many false alerts or false negatives).

In such situations, based on the discussion and the recommendations of Amekoe et al. [168] and results of the Chapter 4 we opt for testing tree models and retain one after testing different criteria (predictive performance, accordance of obtained explanations with human knowledge, training and inference cost):

- Tree ensemble model combined with TreeSHAP: This solution is investigated as the one favoring the predictive performance. The *post hoc* interpretability tool TreeSHAP (Lundberg et al. [64]) is used for interpretability purposes. We recall that the goal behind this option *is not to necessarily know what the model has learned, or what the model is doing, but to instead identify additional information or hidden information that the modeling can tell us about fraudulent activities.* Therefore, the explanations (the *why* question regarding the predictions) provided to fraud experts (on request) for individual examples are explanations according to the TreeSHAP algorithm. XGBoost (Chen and Guestrin [36]) is used as the underlying model.
- GAM based inherently interpretable model: we consider EBM (Nori et al. [27]) in this category.
- Attention-based inherently interpretable model: in this category, we test TabSRA (Amekoe et al. [169, 168]).

As the response of the ML module is per batch and there is an important label delay (1 month for the majority of observations), the possible adaptation strategies can be: (1) Periodical retraining; (2) Performance-based retraining; (3) Retraining on demand/request. With *retraining*, we refer to the process of training a model from scratch, employing the optimal architecture or parameter configuration identified during the initial training phase (or parameter tuning). We do not investigate the partial retraining in this Chapter.

The ***Periodical retraining*** consists in dynamically retraining the model, for example, every month, regardless of the fact that there are drops in performance or not. This strategy is typically a passive approach that does not necessarily require drift detection.
The ***Performance-based retraining*** approach requires the availability of labeled data. However, the majority of labels are known after a period of one month, making this approach result in not retraining the model before the period of one month.
Another interesting approach is the ***retraining based on demand/request***. This approach is used whenever an important feature or predictive variable is identified and should be used in the modeling. Another common situation for retraining on demand is the case where an error is made in the modeling (e.g., data leakage) or in the data collection (or feature computation) and, fortunately, it is noticed before affecting the model performance. We do not explicitly cover this approach in this applicative study.

For these 3 approaches, an important question is *which data to use for the retraining?* A *fixed window* (for example the x last months can be used every retraining), or a the *dynamic window* (of sometimes 3, 6, 9 months...) is typically used for retraining the model. The dynamic window size can be the one maximizing the performance on a fixed test dataset. There are also sample selection techniques that consist in selecting only observations/batches that are representative of the test set. However, this approach is trickier and requires knowledge of dynamics that can occur after the production.

We consider a performance-based retraining strategy, where a model initially trained is not re-trained unless its performance drops below a given *threshold*. A model with this strategy will be preceded by _Pbr_. For example, **XGBoost_Pbr_9m** refers to a XGBoost model used with Performance-based retraining (Pbr) and using a window of 9 months for training the model.

We also investigated the periodical retraining of 1 month. For example **TabSRA_Dyn_6m** refers to a TabSRA model trained periodically (dynamically) every month using a sliding window of 6 months.

XGBoost and EBM are optimized (only on the first/initial training) using 5 different hyperparameter configurations and using early stopping (with a validation dataset for XGBoost). The total number of trees was set to 1,000 (100 for the early stopping round) for XGBoost and 50,000 for EBM.

For TabSRA which has a longer training time (between 3x and 6x depending on the number of months used for the training, x being the training time of XGBoost), we use only one parameter configuration with the maximal number of iterations (epochs) set to 100. Learning rate scheduling (with a patience of 10 iterations) and early stopping (with a patience of 20 iterations) are used. All the experiments are done on a Google Cloud Platform (GCP) on a CPU machine (with 64 cores and 240 Go of RAM).

As shown in Table 5.1, XGBoost appears to be the best performing model in 3/4 months and the TabSRA in November (1/4). The Performance-based retraining (Pbr) has declined to a static approach (where the model is trained a single time) due to the absence of notable performance degradation from August 2023 to November 2023, thereby eliminating the necessity for retraining. Nevertheless, degradation in performance is anticipated with extended test or production duration.

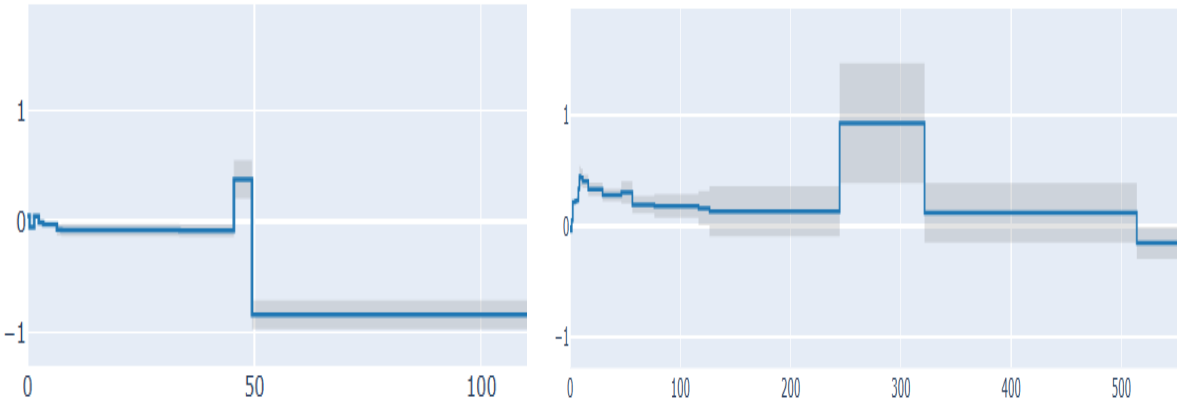
Furthermore, the performance obtained with this strategy is consistent compared to the periodical retraining approach for TabSRA and XGBoost especially when using 12 months

Table 5.1: *Predictive performance on the test months. AUCPR is reported as a predictive performance measure and the bold highlights the best strategy over each test month. The red, green, purple colors show respectively the best strategy combined with XGBoost, EBM and TabSRA respectively. The italic indicates the best model within each strategy (e.g., _Pbr_12m).*

Model	August 2023	September 2023	October 2023	November 2023
XGBoost_Pbr_12m	***20.23***	**20.00**	*23.00*	22.58
EBM_Pbr_12m	17.98	16.83	20.14	19.34
TabSRA_Pbr_12m	19.28	18.27	22.09	*23.59*
XGBoost_Pbr_9m	*19.31*	*19.48*	*22.95*	22.25
EBM_Pbr_9m	18.07	16.64	20.49	19.24
TabSRA_Pbr_9m	19.16	17.72	21.87	*23.24*
XGBoost_Dyn_9m	*19.31*	*18.89*	*23.27*	21.92
EBM_Dyn_9m	18.07	16.19	21.31	19.03
TabSRA_Dyn_9m	19.16	17.91	21.67	***23.72***
XGBoost_Dyn_6m	*19.71*	*18.41*	***23.56***	22.87
EBM_Dyn_6m	17.77	15.40	21.64	19.18
TabSRA_Dyn_6m	17.79	16.41	20.76	*23.38*
XGBoost_Dyn_3m	*19.43*	*19.68*	*22.82*	*22.57*
EBM_Dyn_3m	17.62	16.96	20.88	20.44
TabSRA_Dyn_3m	18.67	16.61	21.20	22.19

for the training.
For EBM the periodical retraining seems to work better. We suspect that it is due to the way that different shape functions are aggregated into one shape function with potential high variance especially for highly imbalanced datasets. As shown in Figure 5.4, there is a significant jump in the shape function which may affect the model's decision/performance after a small change in input features. Consequently, a frequent retraining of EBM seems beneficial for adaptation to recent concepts.
Overall, not only window size (between 3m, 6m, 9m) appears to be the best choice in the periodical retraining with the three models. We recall that the goal of periodical retraining is to use recent observations (small window) but train the underlying model frequently while the Performance-based retraining (Pbr) strategy aims to use a representative (long window) to train a stable model and to conserve it as long as possible unless there is a significant drift requiring a retraining.

From a practical point of view, using only a validation data is not enough to estimate the model's performance in production, especially for non-transparent models. In general, an Out Of Time (OOT) test/backtest data is reserved for additional evaluation; otherwise, a shadow model deployment is typically considered (deploy the model but its decision doesn't affect the fraud detection unless a sufficient number of labels are collected to evaluate its performance in production). In our case, the months August 2023 and September 2023 are



(a) *Shape function feature2* **(b)** *Shape function feature3*

Figure 5.4: *Illustration of uncertainty in the shape functions of EBM (EBM_Pbr_12m). There are some abrupt changes with important variance or uncertainty zones (in gray) that are [45, 50] in Figure 5.4a and [240, 320] in Figure 5.4b. The x-axis represents the feature values and the y-axis represents the contribution to model output (the log-odd).*

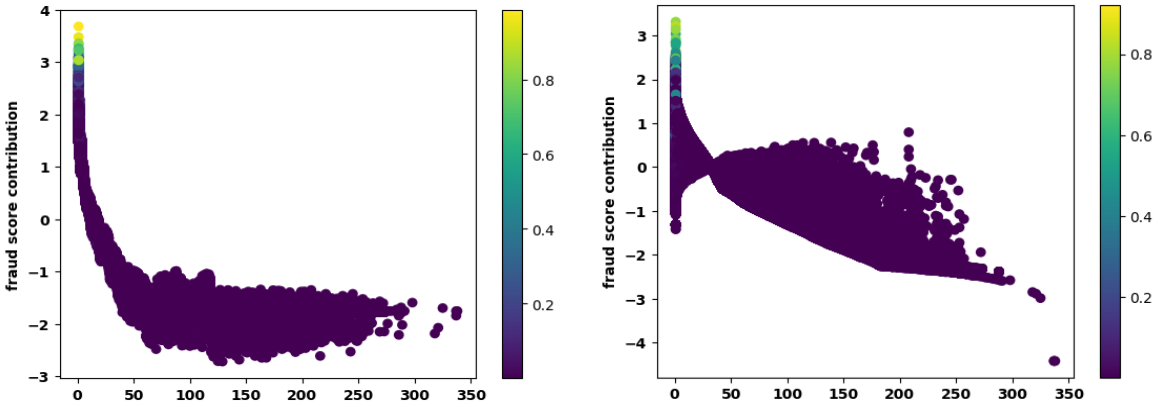
Table 5.2: *Effect of preprocessing on TabSRA's performance. The performance is reported for the strategy TabSRA_Pbr_12m. (a) corresponds to the use of Gaussian quantile transformation and (b) is the standard scaling (using the mean and standard deviation).*

Model	August 2023	September 2023	October 2023	November 2023
(a)	19.28	18.27	22.09	23.59
(b)	17.84	15.07	20.14	18.72

used as out-of-time test data, and, therefore, the real production starts from November 2023 (Figure 5.3). With this consideration, _Pbr_12m (combined with XGBoost or TabSRA) seems to be the most convenient approach for this case study (this decision would be challenged if the periodical retraining demonstrates a clear superiority).
It is important to point out that, unlike EBM and XGBoost, convenient data preprocessing or scaling was necessary for TabSRA as shown in Table 5.2, otherwise its performance drops under that of EBM. We argue that the long tail distribution of some transaction-related features may affect the training of NNs and particularly TabSRA's explanations as highlighted in the 3.3.4.

5.6 Understanding of decisions and changes

Once a *good* ML model and the adaptation strategy are identified (here we have XGBoost_Pbr_12m and TabSRA_Pbr_12m), the next step is to try to understand the decisions made by the model and changes over time.



(a) *XGboost+TreeSHAP* **(b)** *TabSRA*

Figure 5.5: *Illustration of the contribution of feature1 in November 2023. The color bar indicates the output score between 0 and 1. The x-axis represents the feature values and the y-axis the contribution to model output (the log-odd).*

5.6.1 Understanding the predictions

For our application, this step consists in local decision understanding i.e., identifying for some examples why a high/low fraud score is attributed (and which features or group of features are influential in this decision) as well as the global decision understanding. The latter condition involves, for example, identifying trends in the effect/contribution or summarizing the global importance of a given feature (relative to remaining) with one real number. As shown in the Figure 5.5, both TabSRA and XGBoost (interpreted by TreeSHAP) suggest a decreasing trend in the effect of the *feature1*. Moreover, according to these two models, an important number of risky transactions have a value of *feature1* close to 0 in accordance with our data knowledge. A heatmap plot can be used to plot simultaneously the contribution of all input features. We recall that the idea behind the comparison of the explanations of the two models (with a *good* performance) is to identify the possible differences/contradictions in their explanations in order to see how it can help to improve the feature engineering, but more importantly, to identify the one which agrees more with the domain or data knowledge.

In the present application, we do not find any clear contradiction in the attributions of the two models unless the scale of attributions, which are arguably due to the correlation among features.

5.6.2 Detecting and Understanding changes over time

As highlighted in the Section 5.3 a concept drift occurring between the batch scoring or inference time t_j and $t_j + \delta$ with $\delta =$ 1 month can not be easily detected, as the majority of labels are disclosed at time $t_j + \delta$ (Žliobaite [88]). Therefore, our goal is to detect and understand at the inference time t_j the change in the data distribution with respect to a reference distribution or the one learned during the model training $\hat{\mathcal{P}}_{t_0}(\mathbf{x})$. In other words,

Table 5.3: *Drift quantification for XGBoost. The first line represents the performance (**Perf**) measured (in AUCPR) when the label becomes available for the XGBoost_Pbr_12m. The shift is quantified by considering the **training** data as reference.*

Model	Valid	Aug 2023	Sept 2023	Oct 2023	Nov 2023
Perf (AUCPR)	22.48	20.23	20.00	23.00	22.58
Input Shift (AUCROC)	50.05	87.90	88.67	83.50	89.12
Explanation Shift (AUCROC)	50.08	68.55	72.75	68.38	68.64

it is particularly interesting for us to identify at the time t_j the change in the input data that can negatively affect the expected performance of ML module between $[t_j, t_j + \delta[$. At the time $t_j + \delta$ the collected labels can be used to evaluate (or backtest) the performance of ML module in terms of AUCPR for example.
To test if there is a significant change, we use a virtual classifier which should identify the difference between B_{t_0}, the batch of observations used for training and B_{t_j} the test batch on which inference is done at the time t_j. More practically, the artificial label y_i is assigned to every observation $\mathbf{x}_i \in B = B_{t_0} \cup B_{t_j}$:

$$y_i = \mathbb{1}_{\mathbf{x}_i \in B_{t_j}}$$

and a classifier is trained for identifying the observations having $y = 1$ i.e., belonging to B_{t_j}. A performance superior to that of a random guessing model indicates that the used process or algorithm is surely detecting a drift (Hinder, Vaquet, and Hammer [90]).
The main advantage of a virtual classifier (over statistical test-based approaches) is that it eases the localization, that is, the regions of the data space affected by the drift. With more details, the estimate score $\hat{\mathcal{P}}(y|\mathbf{x} \in B_{t_0} \cup B_{t_j})$ can help to filter drifted regions/observations.
In this application, we opt for using simple or transparent models with the goal to have a clear and interpretable understanding of changes:

- We consider a shallow Decision Tree (DT) model of maximum depth set to 4 for identifying the changes in the initial input space. The advantage of the DTs is that they don't require a preprocessing (scaling) of long tail feature distribution.
- We use Logistic Regression (LR) for identifying the change in the feature attribution (explanations). The goal behind this investigation is to identify changes that only affect important features (Mougan et al. [104]).

As shown in Table 5.4 and 5.3 there is no shift between the train and validation data (AUCROC ≈ 50). This is expected, as these two datasets came from the same period.
In the test periods (from August to November 2023), we notice that the changes identified in input are clearly more important than the ones in explanations. This indicates that input shift detection identifies changes that are deemed unimportant for the model's decisions or explanations. Particularly, an important input change was noticed in November 2023, although it is one of the best performing periods of the ML models.
Regarding the explanations shift, we notice an increase in the AUCROC (therefore in identified changes) for TabSRA in October and November 2023; however, there is an increase

Table 5.4: *Drift quantification for TabSRA. The first line represents the performance (**Perf**) measured (in AUCPR) when the label becomes available for the TabSRA_Pbr_12m. The shift is quantified by considering the **training** data as reference.*

Model	Valid	Aug 2023	Sept 2023	Oct 2023	Nov 2023
Perf (AUCPR)	20.16	19.28	18.27	22.09	23.59
Input Shift (AUCROC)	50.05	87.90	88.67	83.50	89.12
Explanation Shift (AUCROC)	50.06	73.86	72.93	74.79	75.89

in the performance computed when labels become available. We argue that this is due to the fact that changes (in input density) that positively affect the model's performance also result in an increase of the estimate shift via AUCROC.
In addition, the use of explanations shift requires the storage of explanations obtained for reference data (the training data in this application) in addition to the initial input data itself and the computation of explanations over each test or inference period, which may be very costly, especially when using *post hoc* interpretability tools. For XGBoost with an acceptable depth (we consider a depth less than 4 in this application) the Tree path dependent TreeSHAP Lundberg et al. [64] manages to provide feature attributions for roughly 600,000 observations in roughly 2 min. However, for some models such as NNs with KernelSHAP (Lundberg and Lee [7]), the computational resources can be very important, especially for very huge datasets.

It is important to point out that when a complex virtual classifier such as Random Forest (RF), XGBoost is used (instead of simple or transparent models), techniques such as cross-validation (instead of a single train/test split) are necessary to estimate/quantify the shift (Hinder, Vaquet, and Hammer [90]).

Towards the use error analysis for drift detection and understanding

Due to the limitations of the input and explanation shift identified and highlighted in the previous Section, we investigate the use of *error analysis* for monitoring (detection and understanding) of changes that may negatively affect the model performance.
In a nutshell, given the trained model, an error is computed for every single point in the reference data used for drift detection:

$$l_i = l(\hat{y}_i, y_i) \tag{5.1}$$

where l can be, for example, the 0-1 loss for classification tasks, the cross-entropy loss for scoring tasks.
For understanding requirements, a shallow DT is used for the partition/segmentation of the reference data $\mathcal{D}_{reference} = \{(\mathbf{x}_i, l_i)\}_{i=1}^{n}$ with respect to the loss. The success of the segmentation can be evaluated using, for instance, the empirical performance (Equation 2.1) with label l_i and the prediction $\hat{l}_i$ outputted by the segmentation model.
After this step, important statistics such as the average and standard deviation of the error are computed for every leaf node of the tree. Statistics value above a predefined threshold indicate that the concerned node is *risky* or red-flagged. Under the no concept drift

assumption between the inference time t_j and the time $t_j + \delta$ (the time when all labels become available), a significant increase of the proportion of observations arriving in risky leaf nodes (with respect to the one measured at the time t_0) is synonymous with a change that can negatively affect the expected predictive performance between the interval $[t_j, t_j + \delta[$. At the time $t_j + \delta$, the obtained labels can additionally be used to check eventual leaf nodes or sub-concept subject to concept drift.
The advantage of this approach over the previous ones:

- Prevent from storing all the reference data or its explanations, but instead only important statistics are stored;
- Help to split directly drift into changes that can affect negatively or positively the expected performance, and therefore to raise alerts conveniently.

It is however important to point out that as loss function (Equation 5.1) is applied per instance, non-additive evaluation metrics such AUCPR, AUCROC cannot be used (these metrics cannot be easily decomposed). In addition, this approach requires a thorough choice of the reference data. For example, using the training data as reference data may result in underestimation of expected test error especially for strong classifiers that can memorize that training data.

We investigated for the TabSRA_Pbr_12m the error analysis based drift detection and understanding approach by considering the training data as reference data. The minimum number of observations in each leaf node is set to 1,000 in order to compute consistent statistics, and the cross-entropy loss is used as the loss function (Equation 5.1).

As shown in Table 5.5, in August and September 2023, an increase in the proportion of observations (more than 3x) in node 20 (which is a red-flagged node) corresponds to a drop in the model's performance (Table 5.1). Inversely, October and November 2023 indicate a reduction in the proportion of observations in the same node. The interesting aspect of the error approach is that it helps to identify and understand the regions affected negatively by the drift.

Furthermore, it helps to identify the regions affected by an increase in the error or entropy once the labels become available. For example, by doing the same analysis for XGBoost_Pbr_12m, we notice the variation of the predictive performance is not due to the change in the proportion of observations in the riskiest nodes (the proportion was almost the same over time in these nodes) but due to an increase in the entropy in some nodes over time.

It is important to highlight that for some problems, the error measure (cross entropy in the case study) may not be correlated to ranking metrics such as metrics such AUCPR, AUCROC used to evaluate the model performance during the development or monitoring. Overall, the error analysis used as drift detection and understanding techniques shows a promising result over a well-known virtual classifier approach, especially from a computational point of view and the identification of drift that negatively affects the model's expected performance.

Table 5.5: *Error analysis for drift detection and understanding with the model TabSRA_Pbr_12m. The red nodes indicate the riskiest ones, i.e., having an average cross-entropy greater than 0.40. The remaining columns highlight the changes in nodes' proportion from the reference or train data (Ref) to November 2023. The ↑ indicates an increase of more than 2x in the proportion of riskiest nodes, and the ↓ the inverse.*

Node id	Error	Ref	Valid	Aug 2023	Sept 2023	Oct 2023	Nov 2023
4	0.01	1.79	1.79	1.77	1.64	1.69	1.67
5	0.04	0.14	0.14	0.16	0.17	0.17	0.14
7	0.05	0.52	0.50	0.32	0.33	0.34	0.33
8	0.15	0.10	0.10	0.06	0.06	0.07	0.05
11	0.00	14.77	14.73	14.26	14.28	12.93	12.56
12	0.01	5.00	4.99	2.98	3.04	4.13	4.15
14	0.00	74.43	74.49	76.56	76.72	77.02	76.96
15	0.01	0.29	0.30	0.32	0.33	0.35	0.31
19	0.20	0.20	0.20	0.16	0.18	0.25	0.25
20	0.48	0.02	0.02	0.10 ↑	0.07 ↑	0.01 ↓	0.01 ↓
21	0.49	0.03	0.03	0.04	0.04	0.05	0.04
24	0.04	0.04	0.95	1.00	0.93	0.92	1.04
25	0.01	1.60	1.61	2.05	1.98	1.83	2.21
27	0.10	0.14	0.14	0.21	0.20	0.19	0.24
28	0.26	0.20	0.20	0.01	0.02	0.04	0.01

5.7 Can model stacking help to improve predictive performance?

One consideration that favors the choice of the Performance-based retraining (Pbr) over periodical dynamic retraining is the model validation process, encompassing the evaluation on Out Of Time (OOT) periods. In our case, the newly developed or retrained ML model should demonstrate a consistent performance on at least the two recent months with labeled data, as highlighted in the previous Section. However, these operational constraints prevent feeding the ML module with the most recent data.

However, these constraints can probably be relaxed by the model validator if the used model is quite simple or a transparent model demonstrates consistent performance as required for the ML module. Instead of considering a transparent model solely, we investigate its combination with the robust Pbr model (obtained in the previous Section) with a *stacking* method. The stacking refers to an ensemble technique that consists in aggregating the predictions coming from different models using a meta model. Similarly to Dal Pozzolo [1], the meta model is a simple weighting of scores outputted by learners in this application:

$$f_{stack_T} = \alpha * f_{Pbr} + (1 - \alpha) * f_{Transparent_{t-\delta}} \tag{5.2}$$

Table 5.6: *Stacking XGBoost with a simple model: XGBoost_Pbr_12m is used in combination with a transparent model. The AUCPR is reported as an evaluation measure.*

weight	Model	August 2023	September 2023	October 2023	November 2023
$\alpha = 0.00$	DT	09.28	06.99	10.14	10.29
$\alpha = 0.50$	DT	18.23	16.84	21.72	20.43
$\alpha = 0.75$	DT	19.62	18.96	22.90	21.95
$\alpha = 0.80$	DT	19.78	19.32	22.98	22.22
$\alpha = 0.00$	LR	16.87	14.77	18.57	19.99
$\alpha = 0.50$	LR	20.17	18.77	22.19	22.94
$\alpha = 0.75$	LR	20.39	19.64	22.90	22.99
$\alpha = 0.80$	LR	**20.41**	19.78	22.97	22.91
$\alpha = 1.00$	-	20.23	**20.00**	**23.00**	**22.58**

Table 5.7: *Stacking TabSRA with a simple model: TabSRA_Pbr_12m is used in combination with a transparent model. The AUCPR is reported as an evaluation measure.*

weight	Model	August 2023	September 2023	October 2023	November 2023
$\alpha = 0.00$	DT	09.28	06.99	10.14	10.29
$\alpha = 0.50$	DT	17.88	15.98	21.52	21.91
$\alpha = 0.75$	DT	19.05	17.73	22.34	23.50
$\alpha = 0.80$	DT	19.10	17.96	**22.36**	23.61
$\alpha = 0.00$	LR	16.87	14.77	18.57	19.99
$\alpha = 0.50$	LR	19.12	17.23	21.52	23.54
$\alpha = 0.75$	LR	**19.37**	18.00	21.99	23.84
$\alpha = 0.80$	LR	19.33	18.06	22.05	**23.90**
$\alpha = 1.00$	-	19.28	**18.27**	**22.09**	23.59

where $\alpha \in [0, 1]$, f_{Pbr} is the obtained Pbr model in the previous Section and $f_{Transparent_{t-\delta}}$ is a transparent model trained on data collected in the previous month ($\delta = 1$).
Table 5.6 and 5.7 report the obtained performances when considering as a simple model a DT of depth less than 5 and Logistic Regression (LR) model. The $\alpha = 0$ corresponds to the case where only the simple model is used.

We can notice that there is no significant improvement over the case where only the f_{Pbr} is used (α close to one). We argue that this is due to the fact that there was no abrupt change in our used data and the transparent model is not strong enough to learn new concepts (significantly different from what the f_{Pbr} learned). That is, the results reported by Dal Pozzolo [1] suggest that $\alpha = 0.5$ could help to improve the performance of ML module when a strong classifier (such RF) is used instead of the transparent one. However, as highlighted previously, the use of such a complex model required a thorough model validation using OOT test months.

5.8 Conclusion

In this Chapter we explored a real application of machine learning in fraud detection and highlighted some important operational constraints that guide our choice on batch inference solution. Our empirical evaluation favors the use of Performance-based retraining (Pbr) strategy approach for the adaptation where the TabSRA proposed during our Ph.D. as part of Inherently Interpretable (II) model provides a competitive predictive performance with respect to XGBoost (a gap of less than 4% AUCPR on average). Particularly, for this application where the explanations are requested only for a few observations, the solution XGBoost+TreeSHAP seems to be a good option especially as obtained explanations are very similar to (are in accordance with) the ones of TabSRA. However, for some problems for which the explanations can be requested for all predictions/inferences, TabSRA can be a more suitable solution as a good trade-off between explanations cost and predictive performance.

Regarding the drift detection and understanding, an error-based method was investigated and demonstrated very interesting results compared to the use of virtual classifiers, especially in identifying changes that may affect negatively the expected performance.

In the next Chapter, we provide some natural extensions or perspectives for the work presented in this Chapter and previous ones, as well as a summary of presented results.

Chapter 6

Conclusion and Perspectives

In this Ph.D. thesis, we addressed some issues faced by the use of Machine Learning in payment fraud detection. Our contributions mainly focused on the interpretability and the adaptation to changes or drifts that obviously occur over time in fraud detection settings. In what follows, we summarize the main takeaways and present some future perspectives of this work.

6.1 Conclusion

Considering the reliability concerns associated with black box ML models, even when paired with *post hoc* interpretability methods, our Ph.D. work begins by critically re-examining the necessity of using *post hoc* tools like SHAP to interpret black box models. Instead of directly contrasting their use with inherently interpretable models, we investigated in Chapter 3 the *trade-off* considering the predictive performance, faithfulness, and stability of explanations. It turns out that the predictive performance gap is less than 4% on average on the recent tabular benchmark introduced by Grinsztajn, Oyallon, and Varoquaux [53]. Is this gap acceptable? The answer will clearly depend on the use case (for which this gap may vary slightly) and we provide an applicative example of bank transfer fraud detection in Chapter 5.
The fidelity or reliability issue that arises with *post hoc* explanations (in contrast to inherent model explanations) with respect to the underlying ML model is well-known for researchers and informed practitioners. But what about the fidelity with respect to the modeled phenomenon? It turns out that an inherently interpretable or transparent model with poor expressiveness can provide misleading discovery in the data while *post hoc* tools may not. Therefore, for tasks for which discovering hidden patterns is important, we recommended in Chapter 3 the investigation of both inherently interpretable and *post hoc* explanations.

The stability of predictions or explanations being an important consideration for real-world applications and fraud detection particularly, we proposed an attention-based solution named TabSRA to complete the previous GAM based ML approaches that were not designed with this consideration. In addition, due to its formulation, TabSRA can handle

more than pairwise interactions without posing the problem of identification that arises with GAMs. Finally, TabSRA eases the human knowledge incorporation as demonstrated with the bank churn prediction case study in 2.

The second contribution is focused on the choice of the learning paradigm for evolving data stream problems when the label may be delayed, as is the case in fraud detection. In this sense, we propose an evaluation framework similar to the backtesting approach usually used in real-world settings where the predictions are made in real-time, but the evaluation is done when labels become available (typically stored in batch one day, one month depending on the speed of the stream and the delay time). Our evaluation in Chapter 4 reveals that in such settings, online or instance incremental algorithms do not have superior performance over batch incremental ones (trained periodically) even in no label delay situations, unlike what the academic research on the topic makes one believe. Furthermore, the consideration of class imbalance, as exhibited by the real-world fraud dataset investigated in Chapter 4 shows a clear superior performance of batch-based solutions such as XGBoost. In addition, the architecture or weights of batch incremental models may change less over time; therefore, making it easier for humans to inspect. Overall, this second part of the thesis sheds light on some limitations of existing online algorithms and invites researchers to not neglect main real-world challenges such as class imbalance and label delay when designing or evaluating real-time prediction or online algorithms.

As part of the applied Ph.D program, we investigate the use of the identified or developed approaches on a real world bank transfer (IBAN) fraud detection problem. We highlighted some important operational constraints that motivate our choices. We explain in Chapter 5 why the batch inference option for the ML module is an acceptable or justified choice, especially when considering the time needed to retrieve and compute aggregated features as well as the inference time for sophisticated models which have the advantage to provide a *good* predictive performance. Our experimental results reveal a strong performance of TabSRA (with a gap less than 4% AUCPR on average) with respect to a *post hoc* interpretability solution combining XGBoost and TreeSHAP.

Finally, we investigated the use of the error analysis as drift detection and understanding technique, which seems to be a promising solution, especially in identifying input changes that may affect negatively the expected performance in the absence or delay of labels.

6.2 Perspectives

In this Section, we provide some perspectives that we judge important as a natural continuity of this work.

6.2.1 Improve the intelligibility of TabSRA

TabSRA, as part of NNs may struggle in handling some irregular and long tail distributions (unlike tree-based models).
In the application of the Chapter 5 we found that Gaussian quantile transformation was very

useful for having a good performance in such long tail distribution situations. However, transformations such Gaussian quantile may impair the extraction of some linear relations (that can only be identified in the initial feature space). In the formulation

$$\sigma(\hat{y}) = \beta_0 + \beta_1 a_1 x_1 + ... + \beta_i a_i x_i + ... + \beta_p a_p x_p,$$

it would be beneficial to find an implementation of attention weights $\mathbf{a} = (a_1, .., a_i, ..., a_p)$ such that they could not only help to account for feature interactions but also the feature scale.

A numerical feature binning or discretization (Gorishniy, Rubachev, and Babenko [39]) can be investigated (in combination with TabSRA) in order to handle irregular distributions in which piece-wise constant approximators (such as EBM) seem to be the go-to solution. The advantage of TabSRA or similar approaches would be to extract or learn linear relations in some area where the latter approaches would apply an abrupt (discontinuous) thresholding.

The second opportunity is to explore the situation when TabSRA may provide correlated feature attributions as it is the case for linear models in multicollinearity situations and for GAMs in concurvity settings (Siems et al. [32]). This would help to implement a convenient regularization approach for training of TabSRA and improve the intelligibility.

6.2.2 On the use of the error analysis for drift detection and understanding

Due to its promising results in our applicative example in Chapter 5, the use of error analysis for drift detection and understanding needs serious investigation regarding theoretical aspects. For identifying risky nodes (red-flagged regions) we only considered the mean error. The consideration of the standard deviation of the error can help to improve the robustness of the approach by imitating, for example, the formulation used for Drift Detection Method (DDM) (Gama et al. [93]).

Regarding the change in the proportion of observations in each region, a statistical test approach similar to the one used in Liu et al. [170] can be used to judge the significance of the changes. That is, in Liu et al. [170], the ratio of proportions is expected to follow a Gaussian distribution under the hypotheses that the data distribution is the same for the two times (or batches).

Another important consideration is the evaluation of the success of the error segmentation in the reference data. This implies questioning first the choice of the reference data but also the evaluation metric. For example, for highly imbalanced scoring problems such as the fraud detection , using classic regression metrics such as R-square or mean square error may be misleading to evaluate the segmentation of cross entropy loss or the Brier score (Brier [171]). A promising approach is to consider the sampling of observations over regions.

From a practical point of view, business metrics can be used to compute the error whenever possible (for example, the amount of fraud) instead of using the 0-1 loss, cross-entropy that may not be always correlated to the ML model performance evaluation metrics such as AUCPR (which is in reality a proxy of the business metric).

6.2.3 Enhance the collaboration between Expert and ML system in fraud detection

To take advantage of the collaboration between the ML module and Expert scoring module in fraud detection, it is important for the two systems to be complementary. We therefore believe that the target feature modeled by the ML module should not only be the fraud proxy as done in Chapter 5 but the error made by the expert system on these transactions should also be considered in the modeling in order to enhance the diversity.

Furthermore, the fraud expert should be aware of important error regions of the ML which typically can be provided by an error analysis.

Bibliography

[1] Andrea Dal Pozzolo. “Adaptive machine learning for credit card fraud detection”. In: (2015).

[2] João Gama et al. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.

[3] Jie Lu et al. “Learning under concept drift: A review”. In: *IEEE transactions on knowledge and data engineering* 31.12 (2018), pp. 2346–2363.

[4] Fabian Hinder, Valerie Vaquet, and Barbara Hammer. “One or two things we know about concept drift—a survey on monitoring in evolving environments. Part A: detecting concept drift”. In: *Frontiers in Artificial Intelligence* 7 (2024), p. 1330257.

[5] Pattaramon Vuttipittayamongkol, Eyad Elyan, and Andrei Petrovski. “On the class overlap problem in imbalanced data classification”. In: *Knowledge-based systems* 212 (2021), p. 106631.

[6] Bartosz Krawczyk. “Learning from imbalanced data: open challenges and future directions”. In: *Progress in artificial intelligence* 5.4 (2016), pp. 221–232.

[7] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" 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*. 2016, pp. 1135–1144.

[9] Salim I Amoukou, Tangi Salaün, and Nicolas Brunel. “Accurate Shapley Values for explaining tree-based models”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 2448–2465.

[10] I Elizabeth Kumar et al. “Problems with Shapley-value-based explanations as feature importance measures”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5491–5500.

[11] Xuanxiang Huang and Joao Marques-Silva. “The Inadequacy of Shapley Values for Explainability”. In: *arXiv preprint arXiv:2302.08160* (2023).

[12] Marian Tietz et al. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*. July 2017. URL: https://skorch.readthedocs.io/en/stable/.

[13] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *arXiv preprint arXiv:1309.0238* (2013).

[14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[15] Charles Elkan. "The foundations of cost-sensitive learning". In: *International joint conference on artificial intelligence*. Vol. 17. 1. Lawrence Erlbaum Associates Ltd. 2001, pp. 973–978.

[16] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.

[17] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[18] Takaya Saito and Marc Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets". In: *PloS one* 10.3 (2015), e0118432.

[19] Skipper Seabold and Josef Perktold. "Statsmodels: econometric and statistical modeling with python." In: *SciPy* 7.1 (2010).

[20] Daniel Lüdecke et al. "performance: An R package for assessment, comparison and testing of statistical models". In: *Journal of Open Source Software* 6.60 (2021).

[21] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[22] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[23] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[24] Martı–n Abadi et al. "{TensorFlow}: a system for {Large-Scale} machine learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[25] Trevor Hastie and Robert Tibshirani. "Generalized additive models: some applications". In: *Journal of the American Statistical Association* 82.398 (1987), pp. 371–386.

[26] Grace Wahba. *Spline models for observational data*. SIAM, 1990.

[27] Harsha Nori et al. "Interpretml: A unified framework for machine learning interpretability". In: *arXiv preprint arXiv:1909.09223* (2019).

[28] Yin Lou et al. “Accurate intelligible models with pairwise interactions”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 623–631.

[29] Benjamin Lengerich et al. “Purifying interaction effects with the functional anova: An efficient algorithm for recovering identifiable additive models”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2402–2412.

[30] Zebin Yang, Aijun Zhang, and Agus Sudjianto. “GAMI-Net: An explainable neural network based on generalized additive models with structured interactions”. In: *Pattern Recognition* 120 (2021), p. 108192.

[31] Timothy O Ramsay, Richard T Burnett, and Daniel Krewski. “The effect of concurvity in generalized additive models linking mortality to ambient particulate matter”. In: *Epidemiology* 14.1 (2003), pp. 18–23.

[32] Julien Siems et al. “Curve your enthusiasm: Concurvity regularization in differentiable generalized additive models”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 19029–19057.

[33] László Kovács. “Feature selection algorithms in generalized additive models under concurvity”. In: *Computational Statistics* 39.2 (2024), pp. 461–493.

[34] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[35] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[36] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986”. In: *Biometrika* 71.599-607 (1986), p. 6.

[38] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[39] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. “On embeddings for numerical features in tabular deep learning”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24991–25004.

[40] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. “A review on the attention mechanism of deep learning”. In: *Neurocomputing* 452 (2021), pp. 48–62.

[41] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[42] Xin Huang et al. “Tabtransformer: Tabular data modeling using contextual embeddings”. In: *arXiv preprint arXiv:2012.06678* (2020).

[43] Yury Gorishniy et al. “Revisiting deep learning models for tabular data”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18932–18943.

[44] Gowthami Somepalli et al. “Saint: Improved neural networks for tabular data via row attention and contrastive pre-training”. In: *arXiv preprint arXiv:2106.01342* (2021).

[45] Dzmitry Bahdanau. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[46] Alexey Dosovitskiy. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[47] Sarthak Jain and Byron C Wallace. “Attention is not explanation”. In: *arXiv preprint arXiv:1902.10186* (2019).

[48] Jasmijn Bastings and Katja Filippova. “The elephant in the interpretability room: Why use attention as explanation when we have saliency methods?” In: *arXiv preprint arXiv:2010.05607* (2020).

[49] Sarah Wiegreffe and Yuval Pinter. “Attention is not not explanation”. In: *arXiv preprint arXiv:1908.04626* (2019).

[50] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30 (2017).

[51] Liudmila Prokhorenkova et al. “CatBoost: unbiased boosting with categorical features”. In: *Advances in neural information processing systems* 31 (2018).

[52] Kuan-Yu Chen et al. “Trompt: Towards a Better Deep Neural Network for Tabular Data”. In: *arXiv preprint arXiv:2305.18446* (2023).

[53] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. “Why do tree-based models still outperform deep learning on tabular data?” In: *arXiv preprint arXiv:2207.08815* (2022).

[54] Vadim Borisov et al. “Deep neural networks and tabular data: A survey”. In: *arXiv preprint arXiv:2110.01889* (2021).

[55] Duncan McElfresh et al. “When Do Neural Nets Outperform Boosted Trees on Tabular Data?” In: *arXiv e-prints* (2023), arXiv–2305.

[56] Rishabh Agarwal et al. “Neural additive models: Interpretable machine learning with neural nets”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4699–4711.

[57] Sercan Ö Arik and Tomas Pfister. “Tabnet: Attentive interpretable tabular learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 6679–6687.

[58] Ravid Shwartz-Ziv and Amitai Armon. “Tabular data: Deep learning is not all you need”. In: *Information Fusion* 81 (2022), pp. 84–90.

[59] Sajid Ali et al. “Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence”. In: *Information fusion* 99 (2023), p. 101805.

[60] Xuanxiang Huang and Joao Marques-Silva. “The Inadequacy of Shapley Values for Explainability”. In: *arXiv preprint arXiv:2302.08160* (2023).

[61] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Anchors: High-precision model-agnostic explanations”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[62] Joao Marques-Silva and Alexey Ignatiev. “Delivering Trustworthy AI through formal XAI”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 11. 2022, pp. 12342–12350.

[63] Hugh Chen et al. “Algorithms to estimate Shapley value feature attributions”. In: *Nature Machine Intelligence* (2023), pp. 1–12.

[64] Scott M Lundberg et al. “From local explanations to global understanding with explainable AI for trees”. In: *Nature machine intelligence* 2.1 (2020), pp. 56–67.

[65] Zachary C Lipton. “The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery.” In: *Queue* 16.3 (2018), pp. 31–57.

[66] Chun-Hao Chang, Rich Caruana, and Anna Goldenberg. “Node-gam: Neural generalized additive model for interpretable deep learning”. In: *arXiv preprint arXiv:2106.01613* (2021).

[67] Sergei Popov, Stanislav Morozov, and Artem Babenko. “Neural oblivious decision ensembles for deep learning on tabular data”. In: *arXiv preprint arXiv:1909.06312* (2019).

[68] Zhi Chen et al. “Using explainable boosting machines (ebms) to detect common flaws in data”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2021, pp. 534–551.

[69] David Alvarez-Melis and Tommi S Jaakkola. “On the robustness of interpretability methods”. In: *arXiv preprint arXiv:1806.08049* (2018).

[70] Wuxing Chen et al. “A survey on imbalanced learning: latest research, applications and future directions”. In: *Artificial Intelligence Review* 57.6 (2024), pp. 1–51.

[71] Nitesh V Chawla et al. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[72] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. “Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning”. In: *International conference on intelligent computing*. Springer. 2005, pp. 878–887.

[73] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. “DBSMOTE: density-based synthetic minority over-sampling technique”. In: *Applied Intelligence* 36 (2012), pp. 664–684.

[74] Georgios Douzas and Fernando Bacao. “Geometric SMOTE: Effective oversampling for imbalanced learning through a geometric extension of SMOTE”. In: *arXiv preprint arXiv:1709.07377* (2017).

[75] Peter Hart. “The condensed nearest neighbor rule (corresp.)” In: *IEEE transactions on information theory* 14.3 (1968), pp. 515–516.

[76] Dennis L Wilson. “Asymptotic properties of nearest neighbor rules using edited data”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), pp. 408–421.

[77] Ivan Tomek. “Two modifications of CNN.” In: (1976).

[78] David A Cieslak and Nitesh V Chawla. “Learning decision trees for unbalanced data”. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I 19*. Springer. 2008, pp. 241–256.

[79] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. “Exploratory undersampling for class-imbalance learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2008), pp. 539–550.

[80] Yoav Freund, Robert Schapire, and Naoki Abe. “A short introduction to boosting”. In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780 (1999), p. 1612.

[81] Nathalie Japkowicz and Shaju Stephen. “The class imbalance problem: A systematic study”. In: *Intelligent data analysis* 6.5 (2002), pp. 429–449.

[82] Vicente Garcı–a, Ramón Alberto Mollineda, and José Salvador Sánchez. “On the k-NN performance in a challenging scenario of imbalance and overlapping”. In: *Pattern Analysis and Applications* 11 (2008), pp. 269–280.

[83] Jerzy Stefanowski. “Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data”. In: *Emerging paradigms in machine learning*. Springer, 2013, pp. 277–306.

[84] Andrea Dal Pozzolo et al. “Calibrating probability with undersampling for unbalanced classification”. In: *2015 IEEE symposium series on computational intelligence*. IEEE. 2015, pp. 159–166.

[85] Miriam Seoane Santos et al. “A unifying view of class overlap and imbalance: Key concepts, multi-view panorama, and open avenues for research”. In: *Information Fusion* 89 (2023), pp. 228–253.

[86] Gustavo EAPA Batista, Ronaldo C Prati, and Maria C Monard. “Balancing strategies and class overlapping”. In: *International symposium on intelligent data analysis*. Springer. 2005, pp. 24–35.

[87] João Gama et al. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.

[88] Indre Žliobaite. “Change with delayed labeling: When is it detectable?” In: *2010 IEEE international conference on data mining workshops*. IEEE. 2010, pp. 843–850.

[89] Flavio Giobergia et al. “A Synthetic Benchmark to Explore Limitations of Localized Drift Detections”. In: *arXiv preprint arXiv:2408.14687* (2024).

[90] Fabian Hinder, Valerie Vaquet, and Barbara Hammer. “One or two things we know about concept drift—a survey on monitoring in evolving environments. Part A: detecting concept drift”. In: *Frontiers in Artificial Intelligence* 7 (2024), p. 1330257.

[91] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. “Failing loudly: An empirical study of methods for detecting dataset shift”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[92] Firas Bayram, Bestoun S Ahmed, and Andreas Kassler. “From concept drift to model degradation: An overview on performance-aware drift detectors”. In: *Knowledge-Based Systems* 245 (2022), p. 108632.

[93] Joao Gama et al. “Learning with drift detection”. In: *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer. 2004, pp. 286–295.

[94] Manuel Baena-Garcıa et al. “Early drift detection method”. In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. Citeseer. 2006, pp. 77–86.

[95] Isvani Frias-Blanco et al. “Online and non-parametric drift detection methods based on Hoeffding’s bounds”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.3 (2014), pp. 810–823.

[96] Albert Bifet and Ricard Gavalda. “Learning from time-changing data with adaptive windowing”. In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007, pp. 443–448.

[97] Heitor M Gomes et al. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* 106 (2017), pp. 1469–1495.

[98] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. “Leveraging bagging for evolving data streams”. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I 21*. Springer. 2010, pp. 135–150.

[99] Pedro Domingos and Geoff Hulten. “Mining high-speed data streams”. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, pp. 71–80.

[100] Fabian Hinder et al. “On the Hardness and Necessity of Supervised Concept Drift Detection.” In: *ICPRAM*. 2023, pp. 164–175.

[101] Heng Wang and Zubin Abraham. “Concept drift detection for streaming data”. In: *2015 international joint conference on neural networks (IJCNN)*. IEEE. 2015, pp. 1–9.

[102] Shujian Yu and Zubin Abraham. “Concept drift detection with hierarchical hypothesis testing”. In: *Proceedings of the 2017 SIAM international conference on data mining*. SIAM. 2017, pp. 768–776.

[103] Johannes Haug et al. “Change detection for local explainability in evolving data streams”. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2022, pp. 706–716.

[104] Carlos Mougan et al. “Explanation Shift: How Did the Distribution Shift Impact the Model?” In: *arXiv preprint arXiv:2303.08081* (2023).

[105] Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. “No free lunch theorem for concept drift detection in streaming data classification: A review”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10.2 (2020), e1327.

[106] Nuwan Gunasekara et al. “Gradient boosted trees for evolving data streams”. In: *Machine Learning* (2024), pp. 1–28.

[107] Joao Gama, Raquel Sebastiao, and Pedro Pereira Rodrigues. “On evaluating stream learning algorithms”. In: *Machine learning* 90 (2013), pp. 317–346.

[108] Joao Gama, Raquel Sebastiao, and Pedro Pereira Rodrigues. “Issues in evaluation of stream learning algorithms”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 329–338.

[109] Dariusz Brzezinski and Jerzy Stefanowski. “Prequential AUC for classifier evaluation and drift detection in evolving data streams”. In: *New Frontiers in Mining Complex Patterns: Third International Workshop, NFMCP 2014, Held in Conjunction with ECML-PKDD 2014, Nancy, France, September 19, 2014, Revised Selected Papers 3*. Springer. 2015, pp. 87–101.

[110] Shivani Agarwal et al. “Generalization Bounds for the Area Under the ROC Curve.” In: *Journal of Machine Learning Research* 6.4 (2005).

[111] Andrew P Bradley. “The use of the area under the ROC curve in the evaluation of machine learning algorithms”. In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[112] Kendrick Boyd, Kevin H Eng, and C David Page. “Area under the precision-recall curve: point estimates and confidence intervals”. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*. Springer. 2013, pp. 451–466.

[113] Jesse Read et al. “Batch-incremental versus instance-incremental learning in dynamic and evolving data”. In: *Advances in Intelligent Data Analysis XI: 11th International Symposium, IDA 2012, Helsinki, Finland, October 25-27, 2012. Proceedings 11*. Springer. 2012, pp. 313–323.

[114] Leszek Rutkowski et al. “Decision trees for mining data streams based on the McDiarmid’s bound”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2012), pp. 1272–1279.

[115] Jacob Montiel et al. “River: machine learning for streaming data in Python”. In: (2021).

[116] Nikunj C Oza and Stuart Russell. "Experimental comparisons of online and batch versions of bagging and boosting". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 359–364.

[117] Jacob Montiel et al. "Adaptive xgboost for evolving data streams". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[118] Haixun Wang et al. "Mining concept-drifting data streams using ensemble classifiers". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 226–235.

[119] Dariusz Brzeziński and Jerzy Stefanowski. "Accuracy updated ensemble for data streams with concept drift". In: *International conference on hybrid artificial intelligence systems*. Springer. 2011, pp. 155–163.

[120] Dariusz Brzezinski and Jerzy Stefanowski. "Reacting to different types of concept drift: The accuracy updated ensemble algorithm". In: *IEEE transactions on neural networks and learning systems* 25.1 (2013), pp. 81–94.

[121] Ryan Elwell and Robi Polikar. "Incremental learning of concept drift in nonstationary environments". In: *IEEE transactions on neural networks* 22.10 (2011), pp. 1517–1531.

[122] Johannes Haug, Klaus Broelemann, and Gjergji Kasneci. "Dynamic Model Tree for Interpretable Data Stream Learning". In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE. 2022, pp. 2562–2574.

[123] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. "Streaming random patches for evolving data stream classification". In: *2019 IEEE international conference on data mining (ICDM)*. IEEE. 2019, pp. 240–249.

[124] Gabriel Aguiar, Bartosz Krawczyk, and Alberto Cano. "A survey on learning from imbalanced data streams: taxonomy, challenges, empirical study, and reproducible experimental framework". In: *Machine learning* 113.7 (2024), pp. 4165–4243.

[125] Luis Eduardo Boiko Ferreira et al. "Adaptive random forests with resampling for imbalanced data streams". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–6.

[126] Lucas Loezer et al. "Cost-sensitive learning for imbalanced data streams". In: *Proceedings of the 35th annual ACM symposium on applied computing*. 2020, pp. 498–504.

[127] Shuo Wang, Leandro L Minku, and Xin Yao. "A learning framework for online class imbalance learning". In: *2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*. IEEE. 2013, pp. 36–45.

[128] Shuo Wang, Leandro L Minku, and Xin Yao. "Resampling-based ensemble methods for online class imbalance learning". In: *IEEE Transactions on Knowledge and Data Engineering* 27.5 (2014), pp. 1356–1368.

[129] Alberto Cano and Bartosz Krawczyk. "ROSE: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams". In: *Machine Learning* 111.7 (2022), pp. 2561–2599.

[130] Gregory Ditzler and Robi Polikar. "Incremental learning of concept drift from streaming imbalanced data". In: *IEEE transactions on knowledge and data engineering* 25.10 (2012), pp. 2283–2301.

[131] Andrea Dal Pozzolo et al. "Learned lessons in credit card fraud detection from a practitioner perspective". In: *Expert systems with applications* 41.10 (2014), pp. 4915–4928.

[132] Heitor Murilo Gomes et al. "A survey on semi-supervised learning for delayed partially labelled data streams". In: *ACM Computing Surveys* 55.4 (2022), pp. 1–42.

[133] Joshua Plasse and Niall Adams. "Handling delayed labels in temporally evolving data streams". In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. 2016, pp. 2416–2424.

[134] Maciej Grzenda, Heitor Murilo Gomes, and Albert Bifet. "Delayed labelling evaluation for data streams". In: *Data Mining and Knowledge Discovery* 34.5 (2020), pp. 1237–1266.

[135] Maximilian Muschalik et al. "isage: An incremental version of SAGE for online explanation on data streams". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2023, pp. 428–445.

[136] Ian Covert, Scott M Lundberg, and Su-In Lee. "Understanding global feature contributions with additive importance measures". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17212–17223.

[137] Johannes Haug, Effi Tramountani, and Gjergji Kasneci. "Standardized Evaluation of Machine Learning Methods for Evolving Data Streams". In: *arXiv preprint arXiv:2204.13625* (2022).

[138] Panpan Zheng. *Dynamic Fraud Detection via Sequential Modeling*. University of Arkansas, 2020.

[139] Bernardo Branco et al. "Interleaved sequence RNNs for fraud detection". In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020, pp. 3101–3109.

[140] Yvan Lucas and Johannes Jurgovsky. "Credit card fraud detection using machine learning: A survey". In: *arXiv preprint arXiv:2010.06479* (2020).

[141] Ali Yeşilkanat et al. "An adaptive approach on credit card fraud detection using transaction aggregation and word embeddings". In: *Artificial Intelligence Applications and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5–7, 2020, Proceedings, Part I 16*. Springer. 2020, pp. 3–14.

[142] Barış Bayram, Bilge Köroğlu, and Mehmet Gönen. "Improving fraud detection and concept drift adaptation in credit card transactions using incremental gradient boosting trees". In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2020, pp. 545–550.

[143] Oluwadare Samuel Adebayo et al. "Comparative Review of Credit Card Fraud Detection using Machine Learning and Concept Drift Techniques". In: *Int. J. Comput. Sci. Mob. Comput* 12 (2023), pp. 24–48.

[144] Asma Cherif et al. "Credit card fraud detection in the era of disruptive technologies: A systematic review". In: *Journal of King Saud University-Computer and Information Sciences* 35.1 (2023), pp. 145–174.

[145] Chao Chen, Andy Liaw, Leo Breiman, et al. "Using random forest to learn imbalanced data". In: *University of California, Berkeley* 110.1-12 (2004), p. 24.

[146] Shubham Ingole et al. "Service-based credit card fraud detection using oracle SOA suite". In: *SN Computer Science* 2 (2021), pp. 1–9.

[147] Jipeng Cui, Chungang Yan, and Cheng Wang. "ReMEMBeR: Ranking metric embedding-based multicontextual behavior profiling for online banking fraud detection". In: *IEEE Transactions on Computational Social Systems* 8.3 (2021), pp. 643–654.

[148] Yvan Lucas et al. "Dataset shift quantification for credit card fraud detection". In: *2019 IEEE second international conference on artificial intelligence and knowledge engineering (AIKE)*. IEEE. 2019, pp. 97–100.

[149] Hugo Thimonier et al. "Comparative Evaluation of Anomaly Detection Methods for Fraud Detection in Online Credit Card Payments". In: *International Congress on Information and Communication Technology*. Springer Nature Singapore Singapore. 2024, pp. 37–50.

[150] Zhenchuan Li et al. "A hybrid method with dynamic weighted entropy for handling the problem of class imbalance with overlap in credit card fraud detection". In: *Expert Systems with Applications* 175 (2021), p. 114750.

[151] Fabrizio Carcillo et al. "Combining unsupervised and supervised learning in credit card fraud detection". In: *Information sciences* 557 (2021), pp. 317–331.

[152] Fábio Pinto, Marco OP Sampaio, and Pedro Bizarro. "Automatic model monitoring for data streams". In: *arXiv preprint arXiv:1908.04240* (2019).

[153] Jannik Kossen et al. "Self-attention between datapoints: Going beyond individual input-output pairs in deep learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28742–28756.

[154] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature machine intelligence* 1.5 (2019), pp. 206–215.

[155] Alexey Ignatiev et al. "Using MaxSAT for Efficient Explanations of Tree Ensembles". In: *AAAI*. 2022.

[156] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *International conference on machine learning*. PMLR. 2017, pp. 3145–3153.

[157] Alfred Ultsch. "CLUSTERING WIH SOM: U* C". In: *Proc. Workshop on Self-Organizing Maps* (Jan. 2005).

[158] David Alvarez Melis and Tommi Jaakkola. "Towards Robust Interpretability with Self-Explaining Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[159] Chirag Agarwal et al. *Rethinking Stability for Attribution-based Explanations*. 2022. arXiv: 2203.06877 [cs.LG].

[160] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Learning hyperparameter optimization initializations". In: *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2015, pp. 1–10.

[161] Sebastian Müller et al. *An Empirical Evaluation of the Rashomon Effect in Explainable Machine Learning*. 2023. arXiv: 2306.15786 [cs.LG].

[162] Joao Gama, Ricardo Rocha, and Pedro Medas. "Accurate decision trees for mining high-speed data streams". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 523–528.

[163] Harsha Nori et al. "Interpretml: A unified framework for machine learning interpretability". In: *arXiv preprint arXiv:1909.09223* (2019).

[164] Kodjo Mawuena Amekoe et al. "TabSRA: An Attention based Self-Explainable Model for Tabular Learning". In: *The 31th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2023.

[165] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[166] Albert Bifet et al. "Efficient online evaluation of big data stream classifiers". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 59–68.

[167] G Aguiar, B Krawczyk, and A Cano. "A survey on learning from imbalanced data streams: taxonomy, challenges, empirical study, and reproducible experimental framework (2022)". In: *arXiv preprint arXiv.2204.03719* (2022).

[168] Kodjo Mawuena Amekoe et al. "Exploring accuracy and interpretability trade-off in tabular learning with novel attention-based models". In: *Neural Computing and Applications* 36.30 (2024), pp. 18583–18611.

[169] Kodjo Mawuena Amekoe et al. "TabSRA: An Attention based Self-Explainable Model for Tabular Learning". In: *ESANN 2023-European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Ciaco-i6doc. com. 2023, pp. 199–204.

[170] Anjin Liu et al. “Regional concept drift detection and density synchronized drift adaptation”. In: *IJCAI International Joint Conference on Artificial Intelligence*. 2017.

[171] Glenn W Brier. “Verification of forecasts expressed in terms of probability”. In: *Monthly weather review* 78.1 (1950), pp. 1–3.

[172] Hyunjik Kim, George Papamakarios, and Andriy Mnih. “The lipschitz constant of self-attention”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5562–5571.

[173] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[174] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

Appendix A

Appendix for Chapter 3

A.1 Additional theoretical results

A.1.1 On the Lipschitz estimate of TabSRALinear ensemble

Theorem 2. *The feature attributions produced by TabSRALinear ensemble (Equation 3.6) are locally stable in the sense of Lipschitz, that is, for all* $\mathbf{x} \in \mathbb{R}^p$*, there exist* $\delta > 0$ *and* $L_{\mathbf{x}} \geq 0$ *finite such that:*

$$\|\mathbf{x} - \mathbf{x}'\|_1 < \delta \implies \|\sum_{h=1}^{H} \boldsymbol{\beta}^h \odot a^h(\mathbf{x}) \odot \mathbf{x} - \sum_{h=1}^{H} \boldsymbol{\beta}^h \odot a^h(\mathbf{x}') \odot \mathbf{x}'\|_1 \leq L_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}'\|_1 \tag{A.1}$$

With $L_{\mathbf{x}} = \sum_{h=1}^{H} \|\boldsymbol{\beta}^h\|_\infty \left[\|\mathbf{a}^h\|_\infty + L_{\mathbf{a}}^h (\|\mathbf{x}\|_\infty + \delta)\right]$ *and* $L_{\mathbf{a}}^h \geq 0$ *the Lipschitz constant of the* h*-th SRA block.*

Theorem 2 demonstrates that the Lipschitz estimate of the TabSRALinear ensemble is theoretically additive with respect to the number of learners in the ensemble, denoted H. Hence, a large value of H may lead to less stable explanations. Our experimental results on *Default benchmark* and the ablation study (Table 3.6) demonstrate that $H = 2$ is typically sufficient to achieve strong predictive performance.
The proof of the Theorem 2 can be readily accomplished by using the fact that: (i) the feature attributions of each individual TabSRALinear in the ensemble are locally stable in the sense of Lipschitz in accordance with Theorem 1. (ii) A finite sum of Lipschitz continuous functions is Lipschitz continuous, as stated in Lemma 3).

A.1.2 Proof Theorem 1.

Before providing a proof of the theorem, we consider the following lemmas:

Lemma 3. *(i) The sum of two Lipschitz continuous functions is Lipschitz continuous*
(ii) The product of two Lipschitz continuous and bounded functions is Lipschitz continuous

Proof. We consider θ and ψ two functions from $\mathbb{R}^p \longrightarrow \mathbb{R}^p$, L_θ and $L_\psi-$ Lipschitz. For all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ we have:
(i)

$$\begin{aligned}\|(\theta+\psi)(\mathbf{x})-(\theta+\psi)(\mathbf{x}')\|_1 &= \|(\theta(\mathbf{x})-\theta(\mathbf{x}'))+(\psi(\mathbf{x})-\psi(\mathbf{x}'))\|_1 \\ &\leq \|\theta(\mathbf{x})-\theta(\mathbf{x}')\|_1+\|\psi(\mathbf{x})-\psi(\mathbf{x}')\|_1 \quad \text{Minkowski's inequality} \\ &\leq L_\theta\|\mathbf{x}-\mathbf{x}'\|_1+L_\psi\|\mathbf{x}-\mathbf{x}'\|_1 \\ &= (L_\theta+L_\psi)\|\mathbf{x}-\mathbf{x}'\|_1\end{aligned} \tag{A.2}$$

(ii) Moreover let's assume that θ and ψ are bounded i.e. $\|\theta\|_\infty< \infty$ and $\|\psi\|_\infty< \infty$, then we have:

$$\begin{aligned}\|(\theta\odot\psi)(\mathbf{x})-(\theta\odot\psi)(\mathbf{x}')\|_1 &= \|\theta(\mathbf{x})\odot(\psi(\mathbf{x})-\psi(\mathbf{x}'))+\psi(\mathbf{x}')\odot((\theta(\mathbf{x})-\theta(\mathbf{x}'))\|_1 \\ &\leq \|\theta(\mathbf{x})\odot(\psi(\mathbf{x})-\psi(\mathbf{x}'))\|_1+\|\psi(\mathbf{x}')\odot((\theta(\mathbf{x})-\theta(\mathbf{x}'))\|_1\end{aligned}$$

using Hölder's inequality:

$$\begin{aligned}&\leq \|\theta(\mathbf{x})\|_\infty\|\psi(\mathbf{x})-\psi(x')\|_1+\|\psi(\mathbf{x})\|_\infty\|\theta(\mathbf{x})-\theta(\mathbf{x}')\|_1 \\ &\leq L_\psi\|\theta(\mathbf{x})\|_\infty\|\mathbf{x}-\mathbf{x}'\|_1+L_\theta\|\psi(\mathbf{x})\|_\infty\|\mathbf{x}-\mathbf{x}'\|_1 \\ &= (L_\theta\|\psi(\mathbf{x})\|_\infty+L_\psi\|\theta(\mathbf{x})\|_\infty)\|\mathbf{x}-\mathbf{x}'\|_1 \\ &\leq (L_\theta\|\psi\|_\infty+L_\psi\|\theta\|_\infty)\|\mathbf{x}-\mathbf{x}'\|_1\end{aligned} \tag{A.3}$$

□

Lemma 4. *The attention vector outputted using the SRA block is stable in the sense of Lipschitz, i.e., for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$, there exists a constant $L_\mathbf{a} \geq 0$ finite such that:*

$$\|a(\mathbf{x})-a(\mathbf{x}')\|_1 \leq L_\mathbf{a}\|\mathbf{x}-\mathbf{x}'\|_1 \tag{A.4}$$

Moreover $\|a\|_\infty= 1$.

Proof. Each component k_i^j of the keys matrix K (resp. q_i^j of Q) is stable in the sense of Lipschitz as outputted by a fully connected layer [172] (linear transformations followed by common activation such as ReLU, Sigmoid) and is bounded in $[0, 1]$ using Sigmoid activation. Hence for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ there exists $\alpha_{k_i^j} > 0$, $\alpha_{q_i^j} > 0$, finite such that $|k_i^j(\mathbf{x})-k_i^j(\mathbf{x}')|= \alpha_{k_i^j}\|\mathbf{x}-\mathbf{x}'\|_1$ and $|q_i^j(\mathbf{x})-q_i^j(\mathbf{x}')|= \alpha_{q_i^j}\|\mathbf{x}-\mathbf{x}'\|_1$.

Then each component of the attention vector is also Lipschitz since:

$$\begin{aligned}
|a_i(\mathbf{x}) - a_i(\mathbf{x}')| &= |\frac{1}{d_k}(\sum_{j=1}^{d_k} k_i^j(\mathbf{x})q_i^j(\mathbf{x}) - \sum_{j=1}^{d_k} k_i^j(\mathbf{x}')q_i^j(\mathbf{x}'))| \\
&= |\frac{1}{d_k}\sum_{j=1}^{d_k}(k_i^j(\mathbf{x})q_i^j(\mathbf{x}) - k_i^j(\mathbf{x}')q_i^j(\mathbf{x}'))| \\
&\leq \frac{1}{d_k}\sum_{j=1}^{d_k}\alpha_i^j\|\mathbf{x}-\mathbf{x}'\|_1 \quad \text{product and sum of Lipsctiz function} \\
&= L_{a_i}\|\mathbf{x}-\mathbf{x}'\|_1
\end{aligned} \tag{A.5}$$

with $L_{a_i} = \frac{1}{d_k}\sum_{j=1}^{d_k}\alpha_i^j$.
Finally we have:

$$\begin{aligned}
\|a(\mathbf{x}) - a(\mathbf{x}')\|_1 &= \sum_{i=1}^{p}|a_i(\mathbf{x}) - a_i(\mathbf{x}')| \\
&\leq \sum_{i=1}^{p} L_{a_i}\|\mathbf{x}-\mathbf{x}'\|_1 \quad \text{using the Equation A.5} \\
&= L_{\mathbf{a}}\|\mathbf{x}-\mathbf{x}'\|_1
\end{aligned} \tag{A.6}$$

with $L_{\mathbf{a}} = \sum_{i=1}^{p} L_{a_i}$.
Since every $a_i \in [0,1]$ we have $\|\mathbf{a}\|_\infty = 1$. □

Proof. of Theorem 1

$$\begin{aligned}
|\boldsymbol{\beta}\cdot(a(\mathbf{x})\odot\mathbf{x}) - \boldsymbol{\beta}\cdot(a(\mathbf{x}')\odot\mathbf{x}')| &= |\boldsymbol{\beta}\cdot(a(\mathbf{x})\odot\mathbf{x} - a(\mathbf{x}')\odot\mathbf{x}')| \\
&\leq \|\boldsymbol{\beta}\|_\infty\|a(\mathbf{x})\odot\mathbf{x} - a(\mathbf{x}')\odot\mathbf{x}'\|_1
\end{aligned} \tag{A.7}$$

Using A.3 and considering $\psi(\mathbf{x}) = a(\mathbf{x})$, $\theta(\mathbf{x}) = \mathbf{x}$ we have $\|\psi\|_\infty = \|a\|_\infty = 1$ and $\|\theta\|_\infty = \|\mathcal{D}\|_\infty = \max_{\mathbf{x}\in\mathcal{D}}\|\mathbf{x}\|_\infty$ which is the overall maximal observable feature value.
Therefore:

$$|\boldsymbol{\beta}\cdot(a(\mathbf{x})\odot\mathbf{x}) - \boldsymbol{\beta}\cdot(a(\mathbf{x}')\odot\mathbf{x}')| \leq \|\boldsymbol{\beta}\|_\infty(\|\mathbf{a}\|_\infty + L_{\mathbf{a}}\|\mathcal{D}\|_\infty)\|\mathbf{x}-\mathbf{x}'\|_1 \tag{A.8}$$

In the formulation of the TabSRALinear, we are not necessarily interested in global stability or a uniform Lipschitz constant, as in Equation A.8 but rather in regional or local stability around a given target or anchor data point.
With this consideration, given the target data point $\mathbf{x}$, we can restrict $\mathcal{D}$ to its neighborhood i.e.,

$$\mathcal{D} = \{\mathbf{x}' \in \mathbb{R}^p / \|\mathbf{x}-\mathbf{x}'\|_1 < \delta\} \tag{A.9}$$

therefore $\|\mathcal{D}\|_\infty \leq \|\mathbf{x}\|_\infty + \delta$.
Using the Equation A.8, it results that for every $\mathbf{x} \in \mathbb{R}^p$, there exists a constant $\delta > 0$ such that $\|\mathbf{x} - \mathbf{x}'\|_1 < \delta$ implies:

$$|\boldsymbol{\beta} \cdot (a(\mathbf{x}) \odot \mathbf{x}) - \boldsymbol{\beta} \cdot (a(\mathbf{x}') \odot \mathbf{x}')| \leq \|\boldsymbol{\beta}\|_\infty (1 + L_{\mathbf{a}}(\|\mathbf{x}\|_\infty + \delta)) \|\mathbf{x} - \mathbf{x}'\|_1 \tag{A.10}$$

□

A.2 Additional empirical informations

A.2.1 Datasets

Middle-scale benchmark

This benchmark of 45 datasets (59 binary classification and regression tasks) is introduced in a paper titled: *Why do tree-based models still outperform deep learning on typical tabular data?* (Grinsztajn, Oyallon, and Varoquaux [53]). The main goal of this benchmark was to identify certain meta-features or inductive biases that explain the superior predictive performance of tree-based models over NNs in tabular learning. We take a step forward by incorporating inherently interpretable models in the assessment of the predictive performance. We provide essential details about datasets and refer the interested reader to the original paper (Grinsztajn, Oyallon, and Varoquaux [53]) for further information. The main criteria for selecting datasets are:

- The datasets contain heterogeneous features (this excludes images and signal datasets).
- The datasets are not high dimensional. That is, the ratio a p/n is below $1/10$, $p < 500$ with p the number of features and n the number of observations.
- The data are I.I.D. Stream-like datasets or time series are removed.
- They are real-world data.
- The datasets are not too easy and not too small, i.e., $p \geq 4$ and $n \geq 3000$.
- They are not deterministic datasets. Datasets where the target is a deterministic function of the predictors of the predictors.

Furthermore, in order to keep the learning as homogeneous as possible, some subproblems that deserve their own analysis are excluded. That is:

- The size of the datasets. In the *Middle-scale* regime, the training set is truncated to 10,000 and the test set to 50,000.
- There is no missing data. All data points with missing values are removed.

- Balanced classes. For classification, the target is binarised if there are several classes by taking the two most numerous classes, and we keep half of the samples in each class.
- Categorical features with more than 20 items are removed.
- Numerical features with less than 10 unique values are removed.

Finally, every algorithm and hyperparameters combination is evaluated on the same random seed **Train/Validation/Test** split or fold. More precisely, 70% of samples for the train set (or the percentage which corresponds to a maximum of 10,000 samples if 70% is too high). Of the remaining 30%, 30% are used for the validation set (truncated to a maximum of 50,000 samples), and 70% for the test set (also truncated to a maximum of 50,000 samples). Depending on the size of the test set, several random seed splits/folds are used (cross-validation). That is:

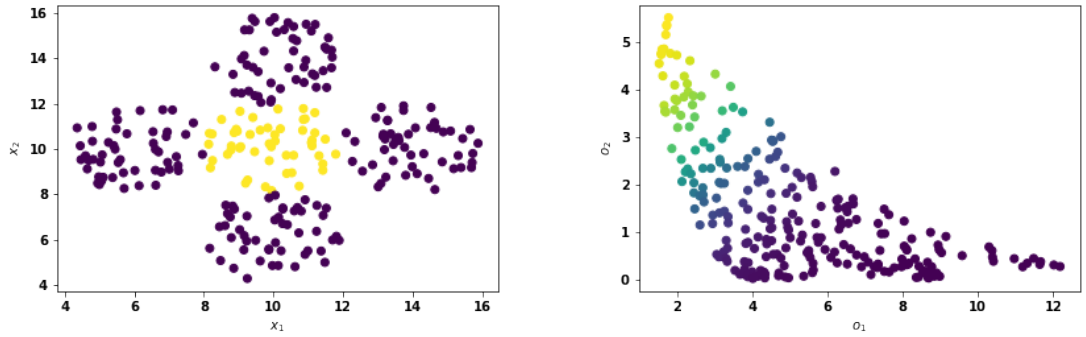
- If the test set is more than 6000 samples, we evaluate our algorithms on 1 fold.
- If the test set is between 3000 and 6000 samples, we evaluate our algorithms in 2 folds.
- If the test set is 1000 and 3000 samples, we evaluate our algorithms on 3 folds.
- If the test set is less than 1000 testing samples, we evaluate our algorithms on 5 folds.

Default benchmark

- **Bank Churn**. This dataset contains details of a bank's customers, and the target variable is a binary variable reflecting the fact whether the customer left the bank (closed his account) or continues to be a customer. We drop the CustomerId and Surname columns. We also drop the HasCrCard column, which is non really informative https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling.
- **Credit Default**. The goal is to predict the default probability (for the next month) of credit card clients in Taiwan using historical data. https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. We drop the SEX column as generally, the gender information is not used in credit scoring.
- **Credit Card Fraud**. For this dataset, the aim is to predict whether credit card transactions are fraudulent or genuine. The original dataset is PCA transformed for privacy purpose. We drop the time information in our study. https://www.kaggle.com/mlg-ulb/creditcardfraud.
- **Heloc Fico**. For this dataset, the target variable to predict is a binary variable called RiskPerformance. The value "Bad" indicates that a consumer was 90 days past due or worse at least once over a period of 24 months from when the credit account was opened. The value "Good" indicates that they have made their payments without ever being more than 90 days overdue. https://community.fico.com/s/explainable-machine-learning-challenge.

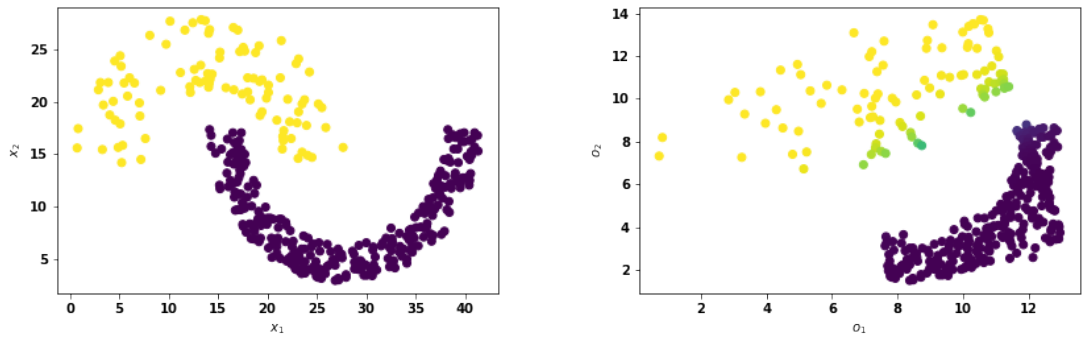
A.2.2 Additional results for TabSRAs: Visualization

How raw data are reinforced using the SRA block.



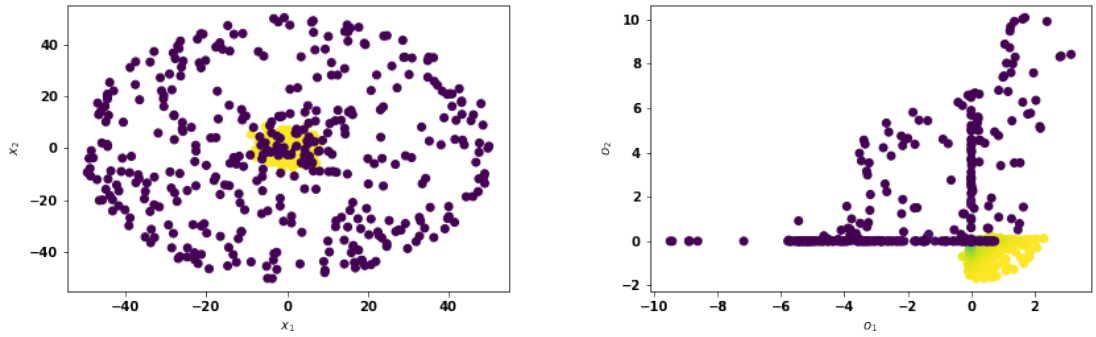
(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure A.1: *Illustration of the reinforcement process with the Noisy two Five sphere: 250 data points*



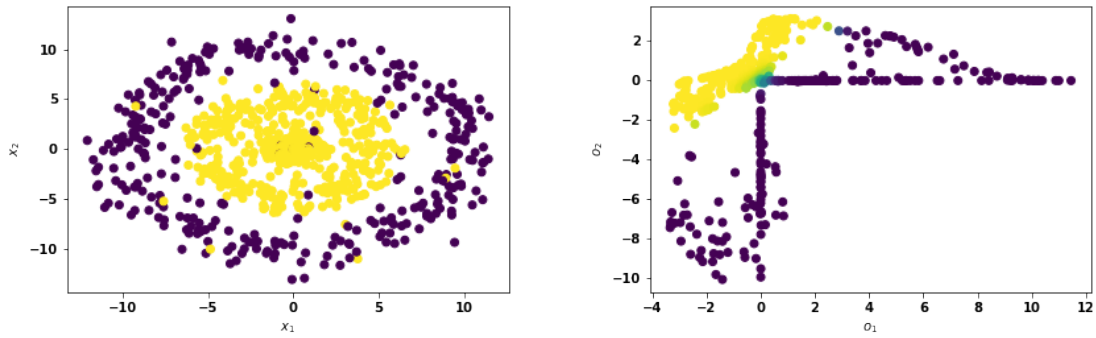
(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure A.2: *Illustration of the reinforcement process with the Two moon with: 373 data points*



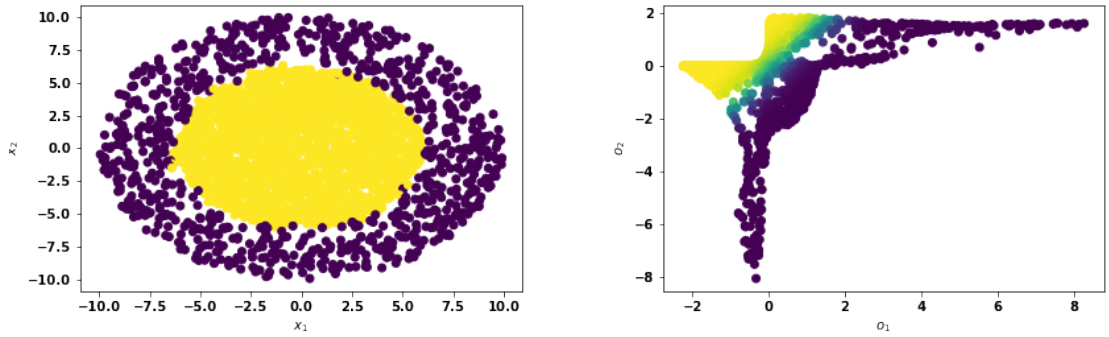
(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure A.3: *Illustration of the reinforcement process with the Two disks: 800 data points*



(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure A.4: *Illustration of the reinforcement process with the Rings: 1000 data points*

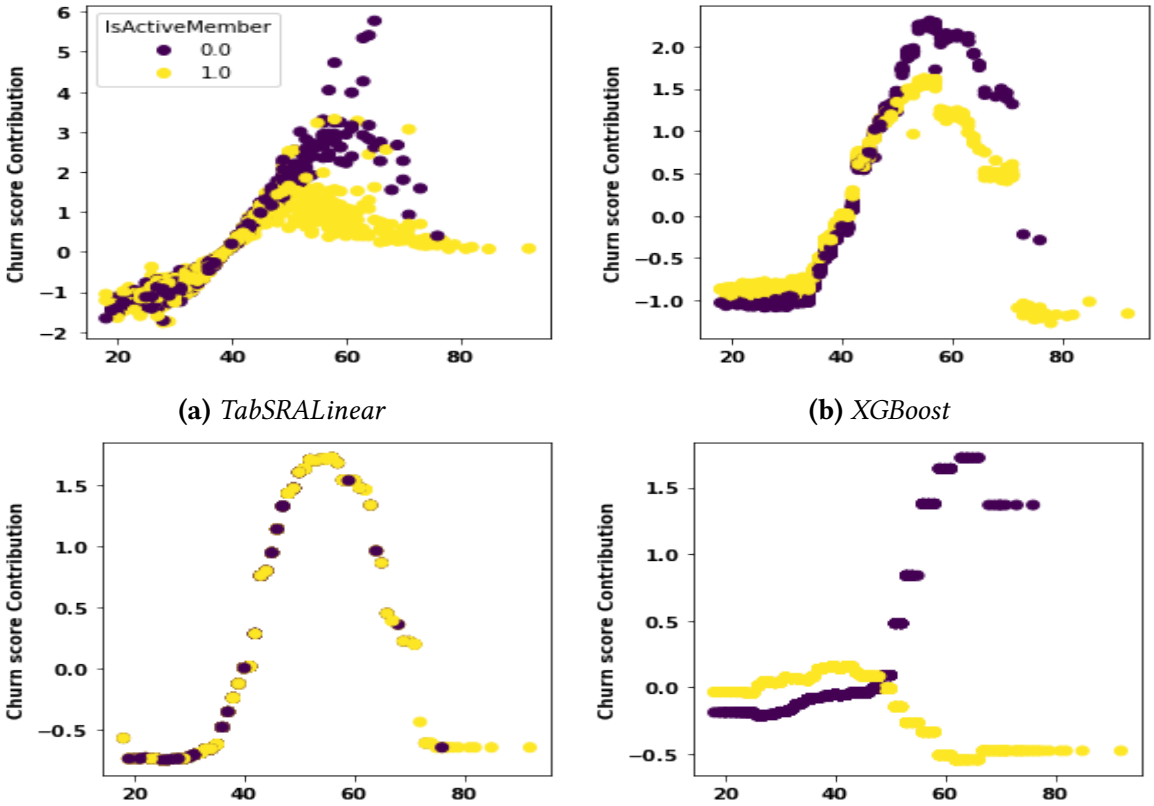


(a) $\mathbf{x}$ **(b)** $\mathbf{o} = \mathbf{a} \odot \mathbf{x}$

Figure A.5: *Illustration of the reinforcement process with the Dense disk: 3000 data points*

A.2.3 Additional results on the applicative case studies

Bank churn modeling



(a) *TabSRALinear* **(b)** *XGBoost*

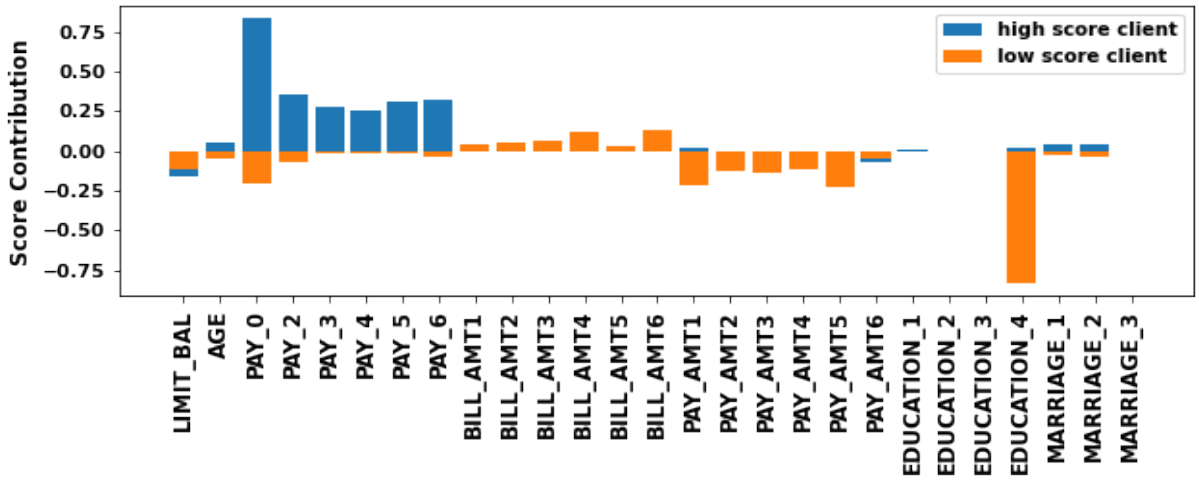
(c) *EBMs* **(d)** *EBM*

Figure A.6: *Bank churn modeling: interaction between the Age and IsActiveMember feature. (A.6b): XGBoost refers to the XGBoost+TreeSHAP feature attribution solution. (A.6c): Indicates the main effect of the Age on the churn score for both EBM_S , EBM and (A.6d) show the pairwise interaction for the latter.*

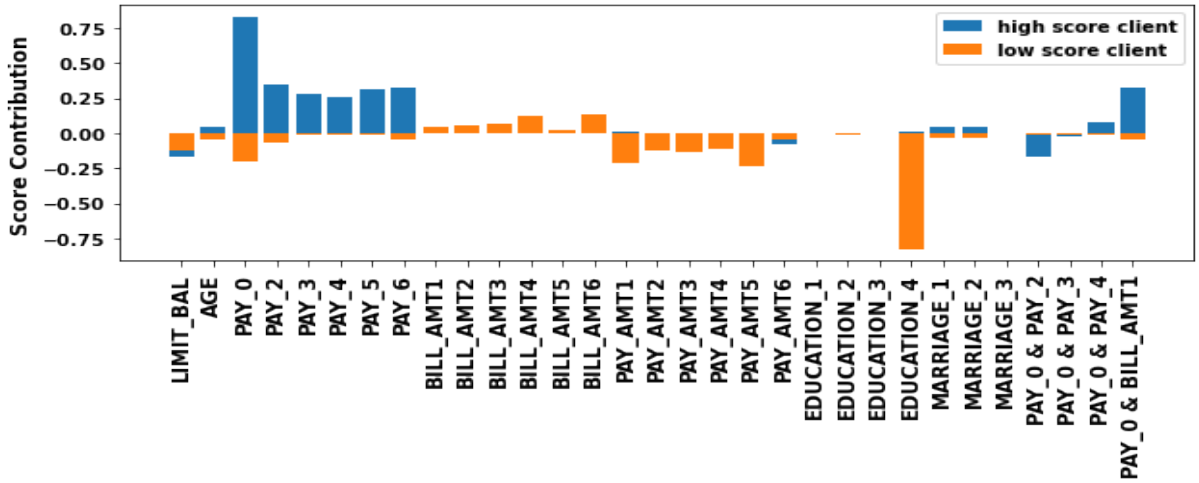
As shown in Fig. A.6, the effect of the *Age* feature on the churn risk is bell-shaped according to TabSRALinear, XGBoost+TreeSHAP, and EBMs. Moreover, there is a strong interaction between the *Age* and *IsActiveMember* features highlighted by the important influence on the churn score of the *Age* (around 60) for non-active members.
To handle this interaction, the GAMs based inherently interpretable solution EBM needed to break it down into main effect and interaction effect, which is not necessary with TabSRALinear. Overall, these findings explain the poor predictive performance of the Logistic Regression (LR) as highlighted in Section 3.4.4.

Credit Card Default

In Section 3.4.4, we demonstrate using the credit card default dataset that TabSRALinear can generate more concise explanations compared to XGBoost+TreeSHAP. As depicted in Fig.



(a) *EBM_S*



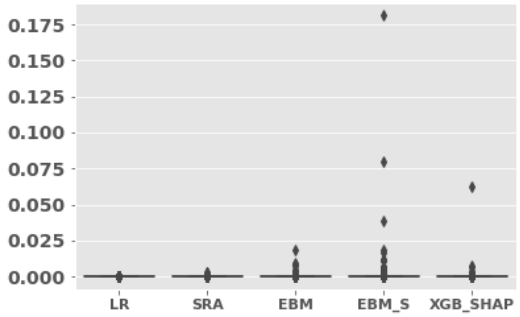
(b) *EBM*

Figure A.7: *Individual prediction understanding for the credit card default dataset.*

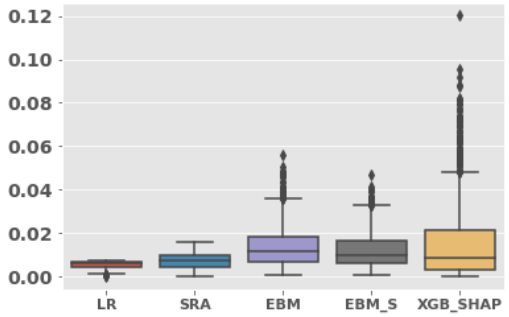
A.7, EBMs also spread contribution among correlated features. As a result, their feature attributions are less sparse compared to those of TabSRALinear, particularly with EBMs having pairwise interactions, which may assign non-zero feature attribution to interaction terms as well as main effects.

A.2.4 Additional results for the robustness study

The output of piecewise constant approximators (for instance EBMs and XGBoost) are generally more sensitive to input perturbation compared to Linear models (LR) and TabSRALinear, as depicted in Fig A.8. We argue that this is due to their flexibility in producing discontinuities or learning irregular functions (Grinsztajn, Oyallon, and Varoquaux [53] and McElfresh, Khandagale, Valverde, Prasad C, Ramakrishnan, Goldblum, and White [55]).



(a) *Credit Card Fraud dataset: 2533 random test points*



(b) *Heloc Fico dataset: 1076 random test points*

Figure A.8: *Change in predictions using input perturbationes (Section 3.4.2). LR = Logistic Regression, SRA=TabSRALinear, XGB_SHAP=XGBoost+TreeSHAP*

A.2.5 Implementation details for the predictive performance evaluation

Search space for hyperparameters

For full-complexity models, the hyperparameter spaces are derived from the previous study [53]. For tree-based models, the number of estimators (trees) is not tuned but rather set to a high value: 250 for Random Forest (RF), 1000 for XGBoost, and CatBoost. For the latter, early stopping is employed with patience 20. Default parameters for tree-based models are ScikitLearn/XGBoost/CatBoost's defaults.

For Neural Nets (NNs), the maximal number of epochs is set to 300 with early stopping and checkpoint (the best model on the validation set is kept). The early stopping round is 40 for MLP, ResNet, TabSRALinear, Linear, 10 for SAINT, and 50 for EBMs. Note that for the model using early stopping, 20% of the training dataset is used as the validation set (different from the validation which is in the test part).
For NNs, we used the Pytorch library (Paszke et al. [173]), the Weights and Biases platform (Biewald [174]) for hyperparameter optimization and experiment tracking.

Table A.1: *Decision Tree (DT)*

Parameter	Distribution	Default
Max depth	[2, 3, 4, 5, 6, 7]	-
Min sample split	[2, 3]	-
Min samples leaf	LogUniformInt [1.5, 50.5]	-

Table A.2: *Linear Models (LR)*

Parameter	Distribution	Default
Learning rate	LogUniform [1e-5,1e-2]	-
Weight decay	LogUniform [1e-6,1e-4]	-
Learning rate scheduler	[True,False]	-
Use bias term	[True]	-
Batch size	[128, 256, 512, 1024]	-

Table A.3: *TabSRALinear*

Parameter	Distribution	Default
Learning rate	LogUniform [1e-5,1e-2]	-
Weight decay	LogUniform [1e-6,1e-4]	-
Learning rate scheduler	[True,False]	-
Use bias term encoders	[True]	-
Use bias term classifier	[True]	-
Dropout	Uniform [0,0.5]	-
N Head	[1,2]	-
Dim Head	[4,8, 12]	-
Batch size	[128, 256, 512, 1024]	-

Table A.4: *EBM_S*

Parameter	Distribution	Default
Learning rate	LogUniform [1e-4,0.7]	-
Max bins	[128,256, 512]	-
Number of interaction terms	[0]	-
Min samples lead	IntUniform [1,100]	-
Max rounds	[20000]	-
Inner bags	[0, 5, 10, 15]	-
Outer bags	[8, 16, 32, 64, 128]	-

A.2.6 Additional results about the predictive performance

Results of hyperpameters optimization using all iterations

In this part, we present the results of hyperparameter tuning using all iterations instead of the use of bootstrapping as described in Section 3.4.1.

As indicated in Table A.13, utilizing all iterations once instead of employing bootstrapping does not change the predictive ranking of algorithms and our conclusions (Section A.2.6).

Table A.5: *EBM*

Parameter	Distribution	Default
Learning rate	LogUniform [1e-4,0.7]	-
Max bins	[128,256, 512]	-
Number of interaction terms	[0, 5, 10, 15, 20, 25]	-
Min samples lead	IntUniform [1,100]	-
Max rounds	[20000]	-
Inner bags	[0, 5, 10, 15]	-
Outer bags	[8, 16, 32, 64, 128]	-

Table A.6: *Random Forest (RF)*

Parameter	Distribution	Default
Max depth	[None, 2, 3, 4] ([0.7, 0.1, 0.1, 0.1])	-
Num estimators	250	
Criterion	[gini, entropy] (classif) [squared_error, absolute_error] (regression)	-
Max features	[sqrt, sqrt, log2, None, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	-
Min samples split	[2, 3] ([0.95, 0.05]	-
Min samples leaf	LogUniformInt[1.5, 50.5]	-
Boostrap	[True, False]	-
Min impurity decrease	[0.0, 0.01, 0.02, 0.05] ([0.85, 0.05, 0.05, 0.05])	-

Table A.7: *XGBoost*

Parameter	Distribution	Default
Max depth	UniformInt[1,11]	-
Num estimators	1000	-
Min child weight	LogUniformInt[1, 1e2]	-
Subsample	Uniform[0.5,1]	-
Learning rate	LogUniform[1e-5,0.7]	-
Col sample by level	Uniform[0.5,1]	-
Col sample by tree	Uniform[0.5, 1]	-
Gamma	LogUniform[1e-8,7]	-
Lambda	LogUniform[1,4]	-
Alpha	LogUniform[1e-8,1e2]	-

Performance per dataset

In this Section, we provide the detailed individual results for each task in the in *Middle-scale benchmark*. R^2 is reported as a predictive performance metric for regression tasks, and the Accuracy is reported for classification tasks.

Table A.8: *CatBoost*

Parameter	Distribution	Default
Max depth	UniformInt[3,11]	-
Num estimators	1000	-
Learning rate	LogUniform[1e-5,0.7]	-
L2 Leaf Reg	LogUniform[1,10]	-

Table A.9: *MLP*

Num layers	UniformInt [1, 8]	4
Layer size	UniformInt [16, 1024]	256
Dropout	[0, 0.5]	0.2
Learning rate	LogUniform [1e-5,1e-2]	1e-3
Category embedding size	UniformInt [64, 512]	128
Learning rate scheduler	[True, False]	True
Batch size	[256, 512, 1024]	512

Table A.10: *ResNet*

Num layers	UniformInt [1, 16]	8
Layer size UniformInt	[64, 1024]	256
Hidden factor	Uniform [1, 4]	2
Hidden dropout	[0, 0.5]	0.2
Residual dropout	Uniform[0, 0.5]	0.2
Learning rate	LogUniform [1e-5, 1e-2]	1e-3
Weight decay	LogUniform [1e-8, 1e-3]	1e-7
Category embedding size	UniformInt [64, 512]	128
Normalization	[batchnorm, layernorm]	batchnorm
Learning rate scheduler	[True, False]	True
Batch size	[256, 512, 1024]	512

Table A.11: *FT Transformer*

Num layers	UniformInt [1, 6]	3
Feature embedding size	UniformInt [64, 512]	192
Residual dropout	Uniform [0, 0.5]	0
Attention dropout	Uniform [0, 0.5]	0.2
FFN dropout	Uniform [0, 0.5]	0.1
FFN factor	Uniform [2/3, 8/3]	4/3
Learning rate	LogUniform [1e-5, 1e-3]	1e-4
Weight decay	LogUniform [1e-6, 1e-3]	1e-5
kv compression	[True, False]	True
kv compression sharing	[headwise, key-value]	headwise
Learning rate scheduler	[True, False]	False
Batch size	[256, 512, 1024]	512

Table A.12: *SAINT*

Num layers	UniformInt [1, 2, 3, 6, 12]	3
Num heads	[2, 4, 8]	4
Dropout	[0, 0.5]	0.2
Layer size	UniformInt [32, 64, 128]	128
Dropout	[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]	0.1
Learning rate	LogUniform[1e-5, 1e-3]	3e-5
Batch size [128, 256, 512, 1024]	512	

Table A.13: *Predictive performance of models across 59 tasks (45 datasets) using all random hyperparameters search iterations. We report the rank over all tasks, the relative test score (Accuracy/R^2) and running time (training+inference) in seconds. MRT: Mean Runing Time*

Model	Rank				Mean Test Score			MRT	
	min	max	mean	median	mean	median	std	mean	median
DT	4.000	12.000	10.525	11	0.869	0.909	0.169	0.282	0.032
EBM_S	1	11	7.686	8	0.931	0.955	0.086	21.101	5.269
EBM	1	10	5.525	5	0.959	0.982	0.067	70.597	18.139
LR	8	12	11.729	12	0.763	0.840	0.230	21.837	20.462
TabSRALinear	2	12	8.008	8	0.909	0.974	0.181	48.167	38.073
MLP	1	12	7.017	8	0.928	0.975	0.140	23.840	16.970
ResNet	2	12	7.102	8	0.911	0.976	0.185	118.734	70.919
SAINT	1	11	5.669	6	0.953	0.982	0.075	253.252	131.435
FT-Transformer	1	10	5.119	5	0.949	0.984	0.096	164.446	82.789
Random Forest	1	10	4.322	4	0.986	0.993	0.019	36.497	7.918
XGBoost	1	10	2.712	2	0.990	0.998	0.021	20.001	12.591
CatBoost	1	9	2.585	2	0.991	1.000	0.020	11.318	3.596

Table A.14: *Regression tasks with **numerical features only**. D1=Ailerons, D2=Bike_Sharing_Demand, D3=Brazilian_houses, D4=MiamiHousing2016, D5=abalone, D6=cpu_act, D7=delays_zurich_transport, D8=diamonds, D9=elevators, D10=house_16H*

Model	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
DT	0.772	0.636	0.982	0.812	0.489	0.967	0.018	0.940	0.688	0.326
EBM_S	0.826	0.652	0.984	0.894	0.512	0.979	0.026	0.944	0.859	0.468
EBM	0.841	0.683	0.990	0.924	0.533	0.982	0.027	0.945	0.887	0.496
LR	0.819	0.280	0.803	0.720	0.476	0.666	0.005	0.929	0.815	0.229
TabSRALinear	0.838	0.644	0.983	0.922	0.514	0.964	0.012	0.942	0.902	0.484
MLP	0.836	0.674	0.994	0.909	**0.576**	0.977	0.013	0.942	0.915	0.481
ResNet	0.838	0.201	0.997	0.913	0.565	0.981	0.011	0.944	0.899	0.489
SAINT	0.571	0.685	0.994	0.921	0.564	0.985	0.021	0.943	**0.917**	0.489
FT-Transformer	0.842	0.679	**0.998**	0.921	0.568	0.980	0.019	0.943	0.915	0.470
Random Forest	0.839	0.687	0.993	0.924	0.551	0.983	**0.031**	0.945	0.837	0.502
XGBoost	0.845	0.692	**0.998**	**0.936**	0.545	**0.986**	0.030	**0.946**	0.904	**0.532**
CatBoost	**0.857**	**0.703**	0.996	**0.936**	0.546	**0.986**	0.028	**0.946**	0.913	0.488

Table A.15: *Regression tasks with **numerical features only**. D11=house_sales, D12=houses, D13=medical_charges, D14=nyc-taxi-green-dec-2016, D15= pol, D16=sulfur, D17=superconduct, D18=wine_quality, D19= yprop_4_1*

Model Model	D11	D12	D13	D14	D15	D16	D17	D18	D19
DT	0.794	0.706	0.978	0.435	0.951	0.783	0.804	0.280	0.026
EBM_S	0.842	0.781	0.979	0.516	0.845	0.733	0.883	0.336	0.048
EBM	0.876	0.817	0.979	0.538	0.923	0.769	0.888	0.392	0.056
LR	0.743	0.674	0.819	0.287	0.706	0.523	0.742	0.240	0.043
TabSRALinear	0.865	0.811	0.979	0.469	0.991	0.817	0.894	0.346	0.023
MLP	0.865	0.815	**0.980**	0.454	0.963	0.843	0.893	0.390	0.014
ResNet	0.866	0.825	0.979	0.469	0.955	0.807	0.892	0.366	0.013
SAINT	0.877	0.825	0.979	0.490	0.995	0.787	0.895	0.371	0.057
FT-Transformer	0.880	0.832	0.979	0.451	0.994	0.859	0.878	0.362	0.045
Random Forest	0.870	0.829	0.979	**0.556**	0.989	0.844	0.908	**0.502**	**0.092**
XGBoost	**0.887**	0.847	0.979	0.551	0.990	0.863	**0.909**	0.490	0.080
CatBoost	**0.887**	**0.850**	0.979	0.536	0.991	**0.877**	0.908	0.495	0.089

Table A.16: *Regression tasks with **heterogeneous features**. D20=Airlines_DepDelay_1M, D21=Allstate_Claims_Severity, D22=Bike_Sharing_Demand, D23=Brazilian_houses, D24=Mercedes_Benz_Greener_Manufacturing, D25=SGEMM_GPU_kernel_performance, D26=abalone, D27=analcatdata_supreme, D28=delays_zurich_transport*

Model	D20	D21	D22	D23	D24	D25	D26	D27	28
DT	0.037	0.384	0.789	0.984	0.574	**1.000**	0.497	0.980	0.056
EBM_S	0.046	0.510	0.744	0.983	0.549	**1.000**	0.514	0.981	0.069
EBM	0.048	0.517	0.924	0.991	0.563	**1.000**	0.537	**0.983**	0.073
LR	0.033	0.482	0.366	0.826	0.533	0.699	0.445	0.740	0.008
TabSRALinear	0.041	0.507	0.927	0.979	0.547	0.999	0.529	0.958	0.052
MLP	0.041	0.514	0.935	0.994	0.558	**1.000**	**0.577**	0.981	0.061
ResNet	0.040	0.512	0.934	0.996	0.567	**1.000**	0.575	0.978	0.057
SAINT	0.045	0.521	0.940	0.994	0.553	**1.000**	0.561	0.979	0.065
FT-Transformer	0.045	0.519	0.933	0.996	0.561	**1.000**	0.559	0.980	0.063
Random Forest	0.045	0.499	0.935	0.993	**0.577**	**1.000**	0.554	0.981	**0.076**
XGBoost	0.048	**0.535**	0.945	**0.997**	0.575	**1.000**	0.554	**0.983**	0.074
CatBoost	**0.050**	**0.535**	**0.949**	0.996	0.576	**1.000**	0.550	0.982	**0.076**

Table A.17: *Regression tasks with **heterogeneous features**. D29=diamonds, D30=house_sales, D31=medical_charges, D32=nyc-taxi-green-dec-2016, D33=particulate-matter-ukair-2017, D34=seattlecrime6, D35=topo_2_1, D36=visualizing_soil*

Model	D29	D30	D31	D32	D33	D34	D35	D36
DT	0.964	0.795	0.978	0.441	0.636	0.180	0.009	**1.000**
EBM_S	0.987	0.853	0.979	0.521	0.670	0.180	0.049	0.935
EBM	0.989	0.885	0.979	0.563	0.679	**0.186**	0.053	0.993
LR	0.952	0.751	0.819	0.318	0.533	0.040	0.000	0.871
TabSRALinear	0.983	0.879	0.978	0.533	0.652	0.061	0.000	0.996
MLP	0.987	0.878	**0.980**	0.470	0.659	0.171	0.025	**1.000**
ResNet	0.987	0.882	0.979	0.486	0.662	0.176	0.019	0.998
SAINT	0.989	0.889	0.979	0.498	0.669	0.180	0.054	**1.000**
FT-Transformer	0.990	0.891	0.979	0.470	0.671	0.179	0.039	**1.000**
Random Forest	0.988	0.875	0.979	0.567	0.673	0.182	**0.070**	**1.000**
XGBoost	**0.991**	**0.897**	0.978	**0.575**	**0.691**	0.185	0.060	**1.000**
CatBoost	**0.991**	**0.897**	0.979	0.560	0.690	**0.186**	0.069	**1.000**

Table A.18: *Classification tasks with **numerical features only**. D37=Bioresponse, D38=Diabetes130US, D39=Higgs, D40=MagicTelescope, D41=MiniBooNE, D42=bank-marketing, D43=california, D44=covertype*

Model	D37	D38	D39	D40	D41	D42	D43	D44
DT	0.680	0.601	0.649	0.788	0.869	0.771	0.839	0.745
EBM_S	0.772	0.605	0.686	0.828	0.915	0.799	0.879	0.752
EBM	0.775	**0.606**	0.707	0.850	0.924	0.803	0.890	0.777
LR	0.735	0.599	0.636	0.768	0.842	0.742	0.831	0.627
TabSRALinear	0.767	0.605	0.679	0.850	0.918	0.789	0.877	0.793
MLP	0.763	0.604	0.685	0.850	0.933	0.789	0.868	0.780
ResNet	0.766	0.604	0.689	0.858	0.936	0.786	0.870	0.790
SAINT	0.758	0.604	0.705	0.847	**0.937**	0.793	0.880	0.801
FT-Transformer	0.748	0.605	0.703	0.857	0.934	0.794	0.885	0.799
Random Forest	**0.794**	0.604	0.707	0.853	0.927	0.797	0.892	0.824
XGBoost	0.792	**0.606**	**0.714**	0.859	**0.937**	0.805	0.901	0.817
CatBoost	0.785	0.605	0.712	**0.860**	**0.937**	**0.806**	**0.904**	**0.830**

Table A.19: *Classification tasks with **numerical features only**. D45=credit, D46=default-of-credit-card-clients, D47=electricity, D48=eye_movements, D49=heloc, D50=house_16H, D51=jannis, D52=pol*

Model	D45	D46	D47	D48	D49	D50	D51	D52
DT	0.752	0.699	0.775	0.566	0.693	0.823	0.715	0.915
EBM_S	0.767	0.713	0.821	0.590	**0.722**	0.870	0.747	0.948
EBM	0.770	0.716	0.829	0.614	0.721	0.877	0.763	0.978
LR	0.706	0.678	0.740	0.556	0.710	0.821	0.724	0.855
TabSRALinear	0.744	0.709	0.792	0.584	0.719	0.873	0.749	0.983
MLP	0.771	0.708	0.807	0.578	0.719	0.877	0.745	0.944
ResNet	0.772	0.706	0.808	0.581	0.718	0.873	0.744	0.948
SAINT	0.763	0.714	0.819	0.583	0.718	**0.888**	0.767	0.979
FT-Transformer	0.775	0.714	0.816	0.584	0.721	0.880	0.763	0.981
Random Forest	0.772	**0.718**	0.859	0.645	0.717	0.881	0.772	0.982
XGBoost	0.773	0.716	**0.868**	**0.662**	0.717	**0.888**	0.777	0.981
CatBoost	**0.776**	**0.718**	0.861	0.642	0.717	0.887	**0.778**	**0.984**

Table A.20: *Classification tasks with **heterogeneous features**. D53=albert, D54=compas-two-years, D55=covertype, D56=default-of-credit-card-clients, D57=electricity, D58=eye_movements, D59=road-safety*

Model	D53	D54	D55	D56	D57	D58	D59
DT	0.637	0.660	0.760	0.698	0.767	0.565	0.727
EBM_S	0.652	0.675	0.776	0.712	0.827	0.596	0.732
EBM	**0.658**	0.672	0.799	0.717	0.838	0.616	0.749
LR	0.635	0.667	0.772	0.679	0.740	0.567	0.697
TabSRALinear	0.643	0.668	0.849	0.711	0.806	0.602	0.748
MLP	0.652	0.678	0.834	0.709	0.819	0.586	0.756
ResNet	0.650	0.676	0.833	0.704	0.824	0.589	0.761
SAINT	0.652	0.674	0.847	0.712	0.830	0.593	0.765
FT-Transformer	0.653	**0.682**	0.858	0.715	0.831	0.590	0.769
Random Forest	0.654	0.679	0.859	0.717	0.876	0.658	0.760
XGBoost	0.656	0.678	0.857	0.714	0.883	**0.660**	0.768
CatBoost	0.656	0.676	**0.873**	**0.719**	**0.884**	0.651	**0.770**