

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES
UNIVERSITÉ DE RENNES

Attaques par Inférence d'Appartenance (MIA)

Une première approche avec le concours *Snake Strikes Back*

Auteurs :

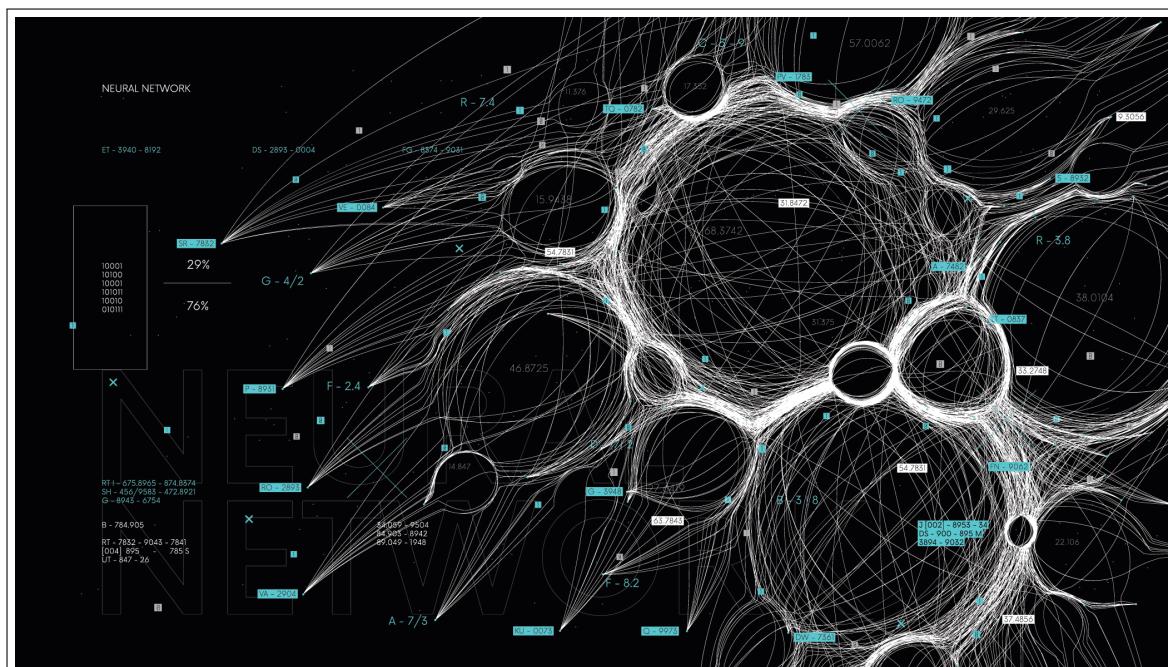
Thomas AUBIN
Selyan DA SILVA
Moussa OUASSOU
Émile PELTIER

Responsable de projet :

Cédric EICHLER

Créateurs de la compétition :

Tristan ALLARD
Mathias BERNARD



Mots-clés : Machine Learning, Confidentialité, Classification, Séries Temporelles, Apprentissage supervisé

Résumé

Le présent rapport a pour objet la synthèse d'un travail mené par un groupe d'étudiants du département Sécurité Informatique de l'INSA Centre Val-de-Loire, dans le cadre de leur participation au concours *Snake Strikes Back* créé par Tristan Allard et Mathias Bernard, chercheurs à l'Université de Rennes. L'objectif du concours est de préciser l'état de l'art sur les protections face aux Attaques par Inférence d'Appartenance (*MIA*) avec une application aux séries temporelles.

Le déroulement de la compétition est rappelé avant de présenter les résultats de l'équipe. Ceux-ci montrent qu'un Réseau Génératif Antagoniste (*GAN*) est sensible à une fuite partielle de ses données d'entraînement si celles-ci sont d'une densité relativement faible (*1000 tuples*). En revanche, une augmentation de cette densité éventuellement couplée à un accès restreint aux données d'entrée du *GAN* entraînent un échec systématique des tentatives d'attaque de l'équipe.

L'attaque du *GAN* est pilotée par l'utilisation de *Shadows Models* dont les données d'entraînement sont connues par l'attaquant. Ainsi, des modèles de classification binaires peuvent être entraînés puis appliquées à des données cible fournies par l'attaquant. La synthèse de l'attaque de ces données est précédée de considérations sur les modèles utilisés et leurs données d'entraînement.

Table des matières

Résumé	1
Introduction	6
Notations	1
I Le concours <i>Snake Strikes Back</i> : position du problème	1
I Contexte et enjeux de la compétition	2
I.1 Principe	2
I.1.1 <i>Taxi Trips (2013-2023)</i> : un dataset public remanié comme donnée d'entrée	2
I.1.2 Objectifs et scoring	2
I.2 Particularités de chacune des 4 tâches	3
I.2.1 Taille de l'échantillon d'entraînement	3
I.2.2 Connaissance <i>a priori</i> des données d'entraînement	3
II Parcours des ressources fournies	4
II.1 L'environnement Snakemake	4
II.2 Les datasets publics	5
II.3 Les datasets synthétiques	6
III DoppelGANger : un générateur de séries temporelles puissant ... mais vulnérable	7
III.1 Particularités du modèle : comparaison avec un GAN classique	7
III.1.1 Fonctionnement global	7
III.1.2 Hyperparamètres choisis pour la compétition	8
III.2 Vers une méthodologie d'attaque	9
II Attaque d'un modèle de Machine Learning par l'utilisation de <i>Shadow Models</i>	11
IV Entraînement de <i>Shadow Models</i> sur des ensembles connus	12
IV.1 Essai préliminaire avec un seul dataset $(E_{T_2, \overline{C}, l_2})$	12
IV.2 Regroupement de plusieurs datasets pour augmenter la taille du dataset d'entraînement $(E_{T_2, \overline{C}, l_{30}})$	13
IV.3 Regroupement d'un nombre important de datasets et nettoyage des données $(E_{T_2, \overline{C}, l_{100}})$	17
IV.3.1 Métriques de comparaison pour piloter le nettoyage	17
IV.3.2 Limite imposée par le temps de calcul	18
IV.3.3 Nettoyage de \mathbb{S}'_{priv_i}	19
IV.4 Construction de données d'entraînement par régression sur les Shadow Models	19
IV.5 Cas particulier des tâches 3 et 4 $(E_{T_3, \overline{C}, l_{60}})$	19
V Méthodes de classification des données	20
V.1 Choix de l'algorithme	21
V.2 Évaluation des classifieurs	21
V.2.1 Choix des métriques	21
V.2.2 Score obtenu pour chaque métrique	22
V.3 Synthèse de l'utilisation des classifieurs	24
VI Synthèse des résultats	25

VI.1	Tâche 1	25
VI.2	Tâche 2	25
VI.3	Tâche 3	25
VI.4	Tâche 4	25
Conclusion		25
III Annexes		1
Annexe 1 : Programmes conçus par l'équipe		2
Annexe 2 : Framework du projet		6
Annexe 3 : Chronologie et contributions		11
IV Bibliographie		12

Table des figures

I.1	Exemples de lignes de $\mathbb{S}_{Pub_{1 2}}$, $\mathbb{S}_{Pub_{3 4}}$ et \mathbb{S}_{Pri_i}	3
II.1	Distribution des données des datasets $\mathbb{S}_{Pub_{T_1-2}}$ et $\mathbb{S}_{Pub_{T_3-4}}$, par jour.	5
II.2	Distribution des données des datasets \mathbb{S}_{Synth_i} , par jour.	6
III.1	Architecture du DoppelGanger donnant les concepts clés et les extensions aux approches type Gan canoniques	8
III.2	Éléments d'une attaque par <i>Shadow Models</i> pour un problème de classification	10
IV.1	Distribution des datasets privés créés par l'équipe - $\mathfrak{H}(\mathbb{S}'_{Priv_{T_2}}) \mid (E_{T_2, \overline{C}, l_{30}})$	13
IV.2	Distribution des datasets synthétiques générés par les datasets privés - $\mathfrak{H}(\mathbb{S}'_{Synth_{i_{T_2}}}) \mid (E_{T_2, \overline{C}, l_{30}})$	14
IV.3	$\mathfrak{D}(\mathbb{S}_{Synth_{T_2}}, \mathbb{S}'_{Synth_i})$ pour $(E_{T_2, \overline{C}, l_{30}})$	15
IV.4	$\mathfrak{H}(\mathbb{S}'_{Etr_{i_{T_1}}}, \mathbb{S}'_{Synth_{i_{T_1}}}) \mid (E_{T_1, \overline{C}, l_{200}})$	16
IV.5	$\mathfrak{D}(\mathbb{S}_{Synth_{T_2}}, \mathbb{S}'_{Synth_i})$ pour $(E_{T_2, \overline{C}, l_{100}})$	17
IV.6	$\mathfrak{D}(\mathbb{S}_{Synth_{T_1}}, \mathbb{S}'_{Synth_i})$ pour $(E_{T_1, \overline{C}, l_{200}})$	18
IV.7	Durée totale d'une classification en fonction de $l(\mathbb{S}_{etr})$	18
V.1	Taux de succès de différents modèles de classification, comparaison entre le DoppelGanger et d'autres générateurs	20
V.2	Score AUC pour une classification naïve et une classification parfaite	21

Liste des tableaux

III.1	Valeurs assignées aux hyperparamètres du modèle pour la compétition	8
V.1	Moyenne harmonique du couple Précision-Rappel de différents classificateurs selon plusieurs configurations d'entraînement	22
V.2	Aire sous la courbe ROC des différents classificateurs	22
V.3	Matrices de confusion de différents classificateurs selon plusieurs configurations d'entraînement .	23
VI.1	Meilleurs scores obtenus sur chaque tâche	25
VI.2	Liste exhaustive des librairies Python utilisées par le projet	10
VI.3	Évaluations par les pairs et auto-évaluation	11

Table des Équations

I.1	Formule simplifiée du scoring de la compétition	2
IV.1	Critère fondamental pour construire \mathbb{S}'_{Pri}	12
IV.2	Hypothèse fondamentale pour construire \mathbb{S}'_{Pri}	12
IV.3	Divergence de Divergence de Kullback-Leibler entre deux distributions de probabilités	17
IV.4	Divergence de Divergence de Jensen–Shannon entre deux distributions de probabilités	17
V.2	Matrice de confusion idéale après normalisation des proportions	21
V.3	Score de précision d'une classification	21
V.4	Score de rappel d'une classification	21
V.5	Score f_1 d'une classification	21

Table des éléments de code

VI.1 Fonctions permettant de classifier les fichiers <code>targetsTask</code>	2
VI.2 Fonctions permettant de créer et d'entraîner des classifieurs	3
VI.3 Fonctions permettant de créer les données d'entraînement des classifieurs	5

Introduction

La prolifération des technologies dites d’Intelligence Artificielle, en particulier des modèles génératifs, soulève des questions inédites de sécurité et de confidentialité des données. En effet, la génération de données synthétiques proches de la réalité suppose une proximité élevée entre les données synthétiques disponibles publiquement, et les données d’entrée des modèles génératifs, *a fortiori* privées. Ces questions peuvent se révéler particulièrement préoccupantes lorsque les données utilisées présentent une sensibilité élevée (*l’emploi d’IA génératives à des fins médicales est par exemple en cours de démocratisation*).

Ainsi, le Machine Learning traditionnel nécessite désormais une expertise sur des disciplines émergentes, à la croisée entre IA et cybersécurité¹ : c’est le cas du Machine Learning Antagoniste (*Adversarial Machine Learning, [1]*). Ce champ de recherche a pour but l’étude des attaques contre les algorithmes de Machine Learning et les protections contre celles-ci.

Parmi ces attaques, un exemple remarquable est celui des Attaques par Inférence de Machine (*Membership Inference Attacks, [28], [27]*). Le principe d’une telle attaque est de reconstituer des données réelles à partir de données produites par des modèles génératifs, lesquels devant être supposément protégés contre ce procédé.

C’est dans ce contexte qu’a été créé *Snake Strikes Back*, deuxième édition d’une compétition encore en version beta. Celle-ci fixe comme objectif la prédiction d’appartenance de séries temporelles aux données d’entrée d’un Réseau Antagoniste Génératif (*GAN, [10]*).

Nous donnons en premier lieu une présentation sommaire du cadre de la compétition. Notre approche du problème est ensuite déclinée en plusieurs composantes : création de données d’entrée pour des Shadow Models, évaluation de ces données d’entrée, entraînement et évaluation de classificateurs binaires. Le dernier chapitre établit la synthèse des résultats obtenus. Les annexes présentent le Framework utilisé par le groupe, les bibliothèques Python (écrites par le groupe ou non), ainsi qu’une rétrospective du projet (*chronologie et évaluation par les pairs*).

1. Certaines entreprises parlent de *MLSecOps*

Notations

Pour l'ensemble du rapport, les notations suivantes sont adoptées.

- T_i , $i \in [1, 4]$ désigne une des 4 des tâches de la compétition.
- \mathbb{S} désigne un ensemble de données comprises entre 0 et 1 structurées en 7 classes. En particulier :
 - $\mathbb{S}_{Pub_{T_1-2}}$ et $\mathbb{S}_{Pub_{T_3-4}}$ sont les datasets publics de la compétition.
 - $\mathbb{S}_{Priv_{T_i}}$ est un dataset privé inaccessible par les attaquants.
 - $\mathbb{S}'_{Priv_{T_i}}$ est un dataset obtenu en concaténant tous les datasets d'entraînement des Shadow Models créés par l'équipe.
 - $\mathbb{S}_{Etr_{T_i}}$ est un dataset utilisé pour entraîner un classifieur. Il est composé pour moitié de $\mathbb{S}'_{Priv_{T_i}}$ et pour moitié de lignes de $\mathbb{S}_{Pub_{T_{i-1}+1}}$ disjointes de $\mathbb{S}'_{Priv_{T_i}}$. Chaque ligne a une classe supplémentaire, booléenne, fonction de l'appartenance ou non à $\mathbb{S}'_{Priv_{T_i}}$.
- \mathbb{C} est un ensemble du même format et de taille 100. Les attaquants doivent déterminer si chacune de ces lignes est un membre ou non du dataset privé.
- \mathcal{G} est l'abréviation de "DoppelGanger", le modèle génératif que l'on cherche à attaquer.
- \mathcal{C} est un algorithme de classification visant à attribuer la classe 0 ou 1 à chaque tuple d'un \mathbb{S} donné, cet attribut désignant l'appartenance ou non au dataset public.
- $E_{T_i, C, l_n}, E_{T_i, \bar{C}, l_n}$ désigne une expérience complète de classification. En particulier :
 - T_i est la tâche abordée.
 - l_n signifie que les classifieurs sont entraînés sur $n \times 1000$ lignes.
 - $C|\bar{C}$ indique si les données d'entrée sont proches des données attaquées ("Clean" ou non).
- $\mathfrak{H}(\mathbb{S})$ est une visualisation de la distribution statistique de chaque classe de \mathbb{S} .
- $\mathfrak{D}(\mathbb{S}_1, \mathbb{S}_2)$ est la divergence de Jensen-Shannon entre deux datasets.
- S_{T_i} désigne un score obtenu sur la tâche i .

Première partie

Le concours *Snake Strikes Back* :
position du problème

Chapitre I

Contexte et enjeux de la compétition

I.1 Principe

Un modèle génératif de type GAN est présenté, dénoté \mathcal{G} . À l'inverse des GAN classiques généralement utilisés pour générer des images, celui-ci génère des séries temporelles, c'est-à-dire des ensembles ordonnés de données mesurées à intervalles réguliers.

Le but de la compétition est de trouver les éventuelles faiblesses du GAN en matière de confidentialité, c'est-à-dire s'il est possible de connaître ses données d'entraînement à partir de ses seules données de sortie (*dites données "synthétiques"*) \mathbb{S}_{Synth} dans le cas d'une étude "black-box" ou de ces mêmes données et du modèle lui-même pour une étude "white-box".

I.1.1 *Taxi Trips (2013-2023)* : un dataset public remanié comme donnée d'entrée

Le cas considéré ici est tiré d'un dataset disponible publiquement présentant 23 caractéristiques de quelque 200 millions de voyages en taxi dans la ville de Chicago, entre 2013 et 2023.

Pour la compétition, on extrait de ces données un nouveau dataset comportant les revenus journaliers d'un taxi sur deux semaines. Des opérations de normalisation et de lissage aux extrêmes sont conduites mais non présentées ici.

Enfin, le dataset présenté aux équipes comporte 1 million de lignes et 7 colonnes correspondant aux 7 premiers jours, les revenus de la deuxième semaine étant reportés sur une nouvelle ligne.

I.1.2 Objectifs et scoring

Les équipes se voient confier 4 tâches $T_i, i \in [1; 4]$ dont la particularité est expliquée plus bas. Pour chacune de ces 4 tâches, 100 lignes issues des deux datasets publics $\mathbb{S}_{Pub_{T_1, T_2}}$ et $\mathbb{S}_{Pub_{T_3, T_4}}$, sont regroupées dans un ensemble "cible" que l'on appellera $\mathbb{C}_i, i \in [1; 4]$.

L'objectif de l'équipe est de déterminer si chaque ligne appartient effectivement aux datasets d'entraînement $\mathbb{S}_{Pri_{T_i}}$ de \mathbb{G} .

Le score de chaque tâche est basée sur la différence entre le taux de vrais et de faux positifs, la formule complète étant détaillée dans [2].

$$S_{T_i} = \tau_{tp} - \tau_{fp} \tag{I.1}$$

I.2 Particularités de chacune des 4 tâches

I.2.1 Taille de l'échantillon d'entraînement

Le nombre de lignes varie d'un facteur 10 entre les tâches 1-3 et 2-4. Ainsi :

- \mathbb{S}_{Pri_1} et \mathbb{S}_{Pri_3} contiennent 10000 lignes
- \mathbb{S}_{Pri_2} et \mathbb{S}_{Pri_4} contiennent 1000 lignes

I.2.2 Connaissance *a priori* des données d'entraînement

Les lignes de \mathbb{S}_{Pri_1} et \mathbb{S}_{Pri_2} sont des copies exactes des lignes de $\mathbb{S}_{Pub_{1|2}}$.

Les tâches 3 et 4 sont beaucoup plus subtiles, car les lignes de \mathbb{S}_{Pri_3} et \mathbb{S}_{Pri_4} ne correspondent pas exactement à celles de $\mathbb{S}_{Pub_{3|4}}$. Chaque ligne est en fait une concaténation des quatre premiers jours d'une ligne de $\mathbb{S}_{Pub_{3|4}}$ et de trois jours n'appartenant pas à ce dataset, et par conséquent hors de la connaissance de l'attaquant.

d1	d2	d3	d4	d5	d6	d7

(a) Pour $\mathbb{S}_{Pub_{1|2}}$

d1	d2	d3	d4	d5	d6	d7

(b) Pour $\mathbb{S}_{Pri_{1,2}}$

d1	d2	d3	d4	d5	d6	d7

(c) Pour $\mathbb{S}_{Pub_{3|4}}$

d4	d5	d6	d7	d8	d9	d10

(d) Pour $\mathbb{S}_{Pri_{3,4}}$

FIGURE I.1 – Exemples de lignes de $\mathbb{S}_{Pub_{1|2}}$, $\mathbb{S}_{Pub_{3|4}}$ et \mathbb{S}_{Pri_i}

Chapitre II

Parcours des ressources fournies

II.1 L'environnement **Snakemake**

Snakemake est un framework permettant de modifier des échelles de données et de créer des fichiers par pipeline. Similaire à **GNU-Make**, cet outil conçu pour la bio-informatique est utilisé ici pour créer et compresser le dataset de Chicago.

Le fichier de configuration, appelé **Snakefile**, comprend une vingtaine d'étapes allant du téléchargement des données jusqu'à la sortie des ensembles \mathbb{S}_{Synth_i} par \mathcal{G} . À cet égard il est crucial de comprendre finement son fonctionnement, car il doit être manipulé si l'on veut créer des *Shadow Models* (*voir IV*).

Une fois que la commande **Snakemake** est terminée, l'équipe peut commencer à travailler avec un dossier comprenant notamment :

- Le dossier **data** comprenant les datasets \mathbb{S}_{Pub} et \mathbb{S}_{Synth} . Les formats utilisés (**.parquet**, **.npz**, **.pckl**) sont facilement convertissables en dataframes par **pandas**.
- Les scripts nécessaires au fonctionnement de \mathcal{G}
- Un certain nombre de fichiers de config **.yaml**

Pour des raisons évidentes, les datasets \mathbb{S}_{Pri} sont hachés.

Le concours étant encore en version *beta*, l'équipe a rencontré de nombreux problèmes lors de l'installation des données fournies. Il est donc préférable à ce stade de développer d'utiliser un environnement Linux et de ne télécharger le dataset initial qu'une fois pour toute l'équipe.

II.2 Les datasets publics

Comme expliqué précédemment, le dataset prend la forme de 7-tuplets de valeurs entre 0 et 1. On peut alors répartir les valeurs par intervalles pour chaque classe du tuple et évaluer leur quantité sur chacun de ces intervalles. Cette représentation visuelle élémentaire sera notée $\mathfrak{H}(\mathbb{S})$ et est donnée ci-dessous.

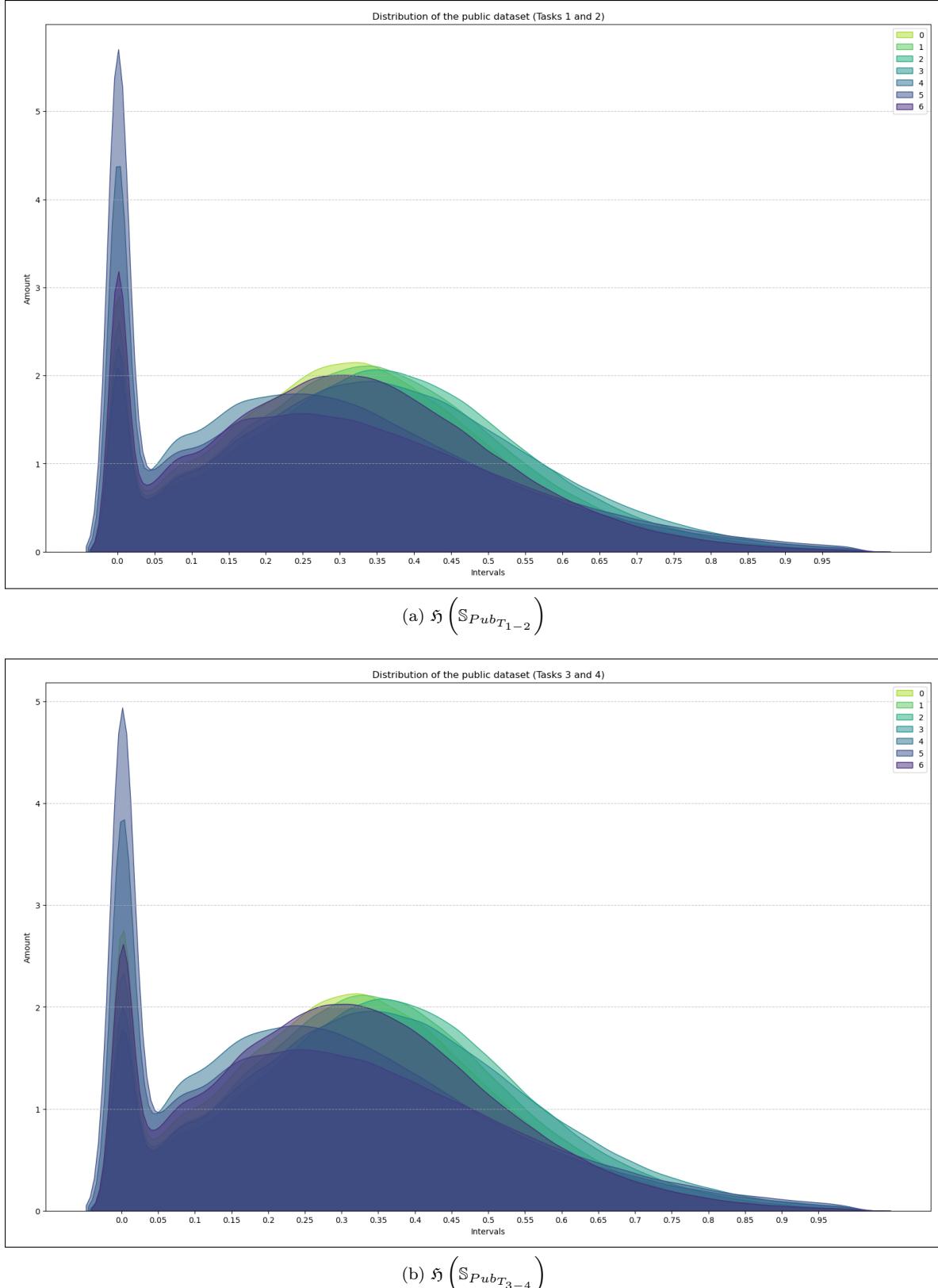


FIGURE II.1 – Distribution des données des datasets $\mathbb{S}_{PubT_{1-2}}$ et $\mathbb{S}_{PubT_{3-4}}$, par jour.

II.3 Les datasets synthétiques

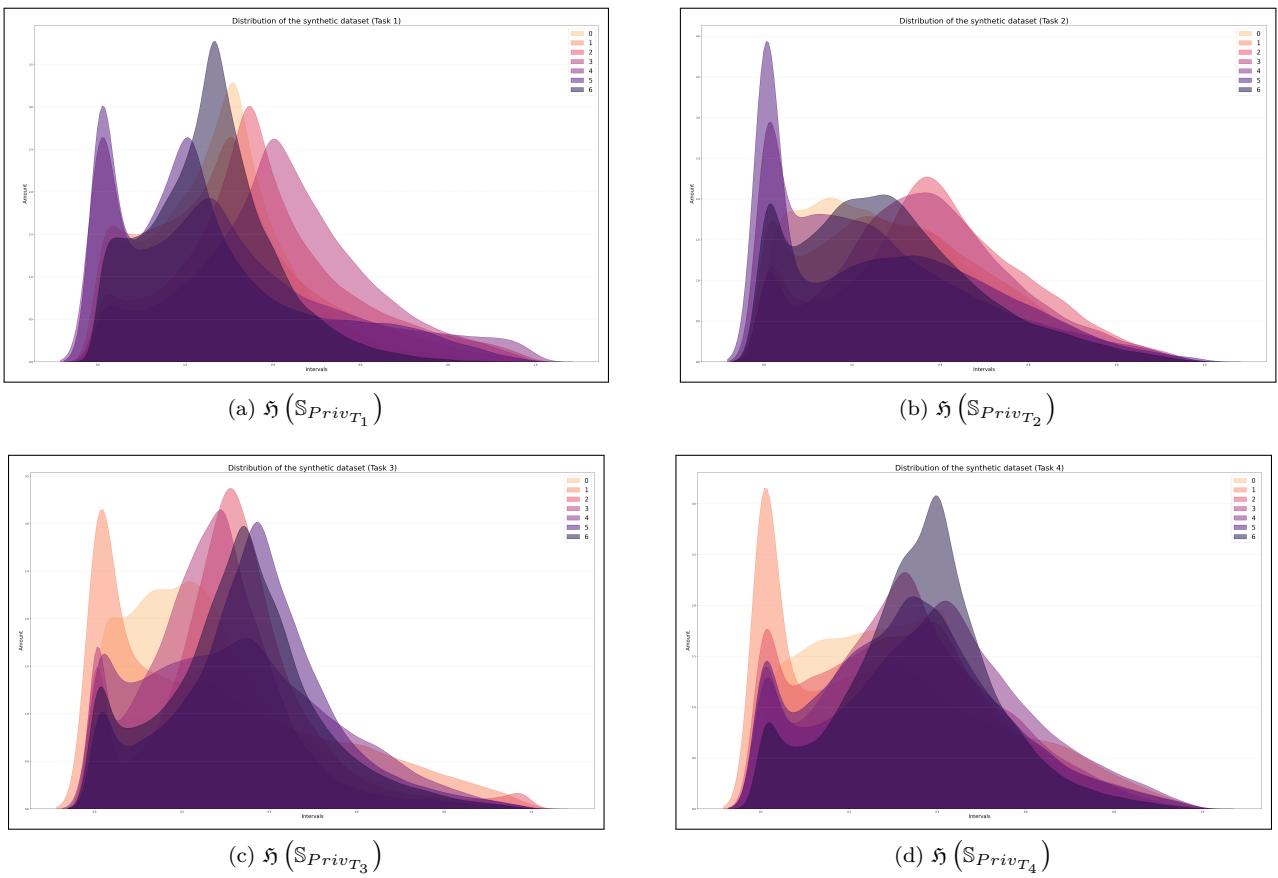


FIGURE II.2 – Distribution des données des datasets \mathbb{S}_{Synth_i} , par jour.

Chapitre III

DoppelGANger : un générateur de séries temporelles puissant ... mais vulnérable

Ce chapitre s'appuie principalement sur l'article publié par l'équipe ayant développé le modèle DoppelGANger ([17]).

III.1 Particularités du modèle : comparaison avec un GAN classique

III.1.1 Fonctionnement global

La principale contrainte reposant sur le GAN est la génération de séries temporelles. En effet l'architecture traditionnelle du GAN (*MLP, Multi-Layer Perceptron*) n'est pas adaptée à la génération de telles données. L'architecture du DoppelGanger est donc une variante de RNN appelée LSTM (long-term short-term memory).

Le fonctionnement du DoppelGanger, schématisé plus bas, est résumé comme suit par ses créateurs :

1. Capture des corrélations entre les métadonnées et les mesures, en utilisant un discriminateur auxiliaire
2. Ajout de métadonnées factices capturant les valeurs minimales et maximales pour chaque échantillon généré pour remédier au problème d'effondrement des modes¹ concernant les mesures.
3. Utilisation d'un générateur RNN par batchs pour capturer les corrélations temporelles et synthétiser de longues séries temporelles représentatives.

1. production d'échantillons très similaires ou identiques, couvrant uniquement une partie limitée (un mode) de la distribution des données réelles, et ne représentant pas la diversité complète des données d'origine

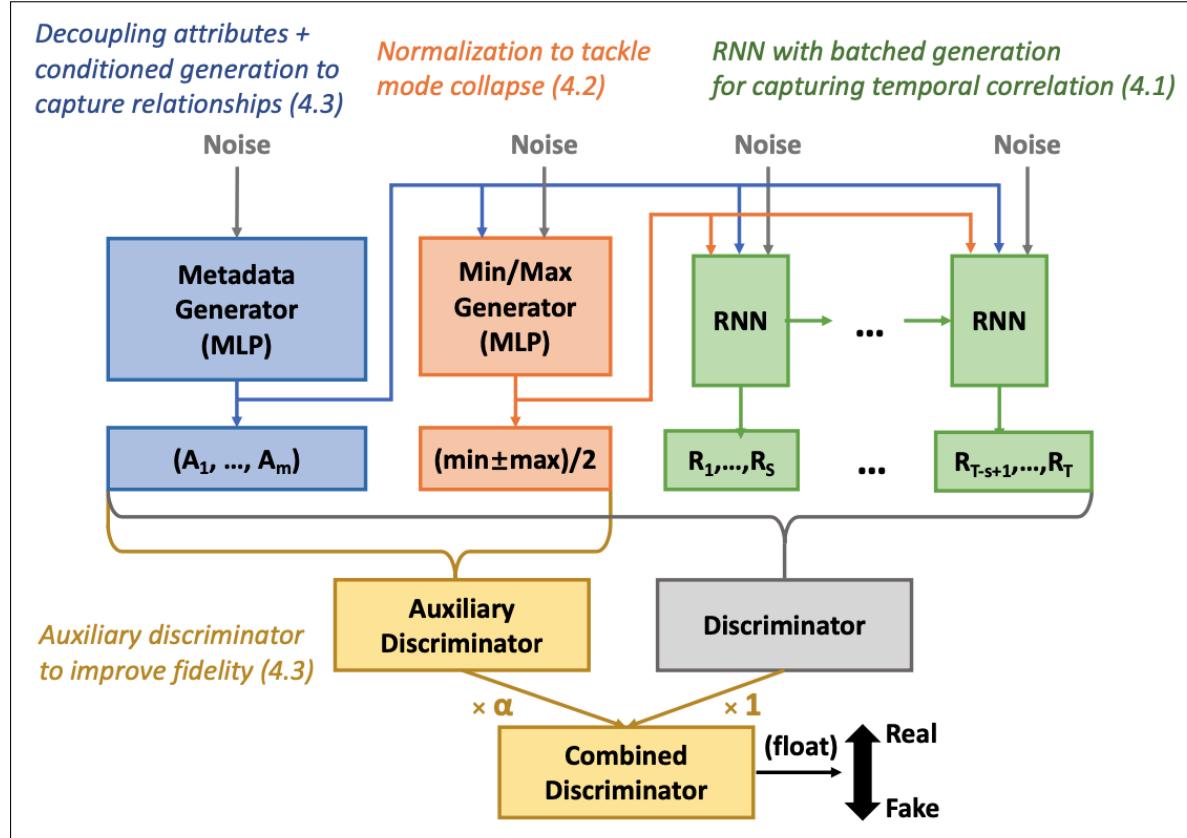


FIGURE III.1 – Architecture du DoppelGanger donnant les concepts clés et les extensions aux approches type Gan canoniques

III.1.2 Hyperparamètres choisis pour la compétition

Le modèle de la compétition n'inclut pas de métadonnées dans ses entrées et sorties, pour des raisons de simplicité.

Paramètre	Valeur
Taille du batch	200
Taux d'apprentissage du générateur	$6 \cdot 10^{-3}$
Taux d'apprentissage du discriminateur	$5 \cdot 10^{-3}$
Nombre de réseaux cachés du générateur	5
Taille des réseaux cachés du générateur	75
Nombre d'epochs	1000 (pour les tâches 1 et 3) ou 5000 (pour les tâches 2 et 4)

TABLE III.1 – Valeurs assignées aux hyperparamètres du modèle pour la compétition

III.2 Vers une méthodologie d'attaque

Une fois les ressources appropriées par l'équipe, il est possible de déterminer une première approche du problème, en trouvant un compromis entre exploitation des ressources disponibles, adaptation à des néophytes en Machine Learning, temps et efficacité de la méthode de résolution.

L'avantage principal dont l'équipe dispose est que \mathcal{G} est mis à disposition. Il est alors possible de conduire de nombreuses expérimentations en le prenant comme point de départ.

Bien que les scripts soient intégralement fournis, l'équipe adopte une approche essentiellement "black-box" par souci de simplicité. La méthode d'attaque peut alors être résumée comme suit :

1. Sélection de faux dataset privé \mathbb{S}'_{Pri_i}
2. Génération de faux datasets synthétiques \mathbb{S}'_{Synth_i} en entraînant \mathcal{G} sur \mathbb{S}'_{Pri_i}
3. Entraînement d'un algorithme de classification \mathcal{C} sur un dataset d'entraînement \mathbb{S}'_{Etr} dont l'appartenance des lignes à \mathbb{S}'_{Pri_i} est connue
4. Classification de \mathbb{C} par l'algorithme

Cette méthode utilisant un avatar du générateur initial est appelée **attaque par Shadow Model**. Celle-ci suppose de déterminer intelligemment \mathbb{S}'_{Pri_i} d'une part, et \mathcal{C} d'autre part. C'est l'objet de la partie suivante qui consacre un chapitre à chacun de ces points, puis résume leur efficacité sur chaque tâche.

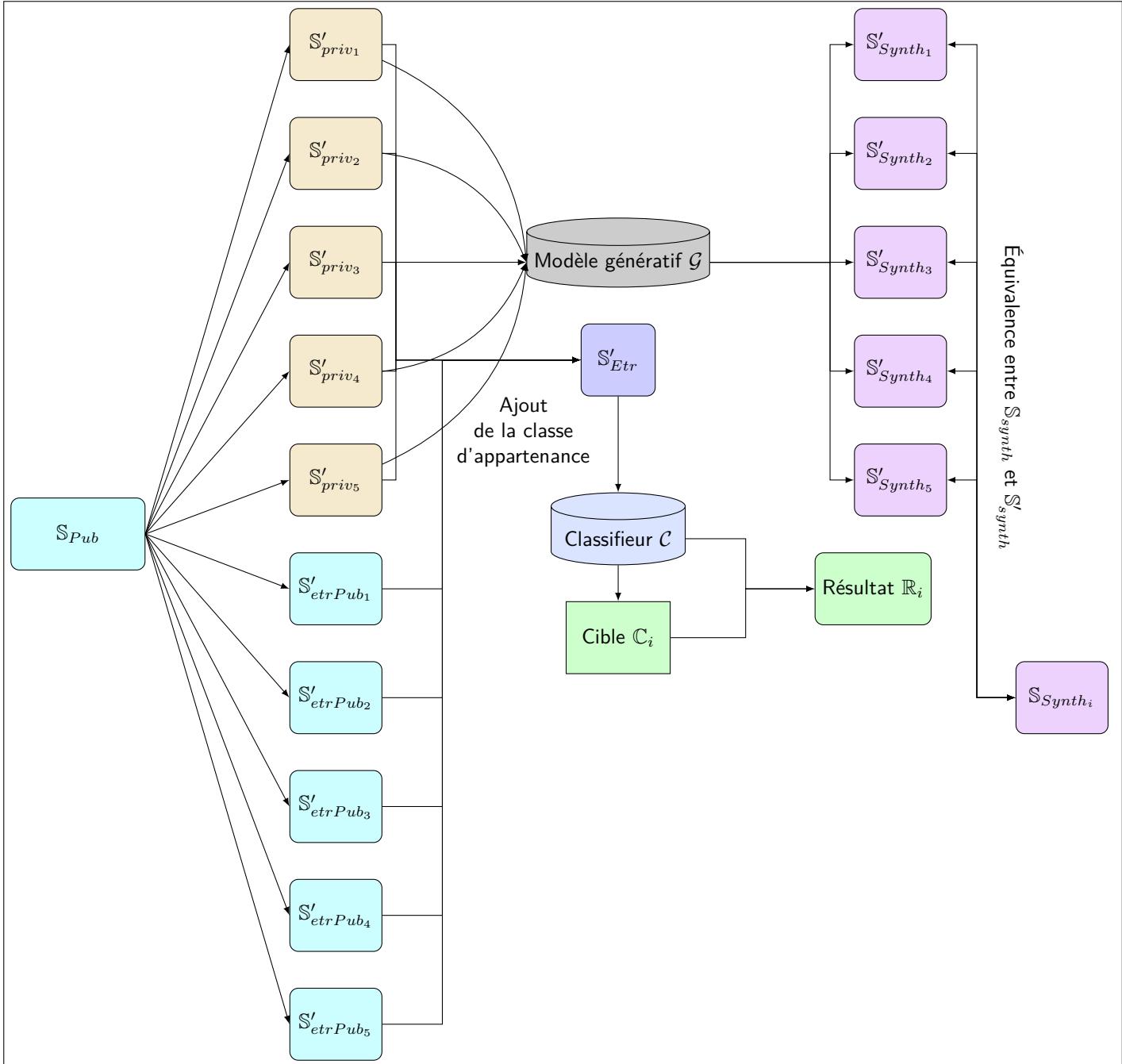


FIGURE III.2 – Éléments d'une attaque par *Shadow Models* pour un problème de classification

Les expérimentations présentées plus bas découlent, sauf mention explicite du contraire, de l'étude de la tâche 2 (*petit dataset d'entraînement et connaissance a priori de l'ensemble des données utilisées*). La portabilité de celles-ci sur une autre tâche n'a été que partiellement abordée.

Deuxième partie

Attaque d'un modèle de Machine Learning par l'utilisation de *Shadow Models*

Chapitre IV

Entraînement de *Shadow Models* sur des ensembles connus

Le Shadow Model n'est efficace que sous la condition d'adopter un comportement similaire à celui du modèle initial.

Ce chapitre a donc pour but la détermination d'un ensemble \mathbb{S}'_{Pri_i} , composé du plus grand nombre possible de datasets d'entraînement de \mathcal{G} notés \mathbb{S}'_{Pri_i} générant \mathbb{S}'_{Synth_i} tel que :

$$\begin{aligned}\mathbb{S}'_{Pri} &= \sum_i \mathbb{S}'_{Pri_i} \\ \forall i; \mathbb{S}'_{Pri_i} &\approx \mathbb{S}_{Pri}\end{aligned}\tag{IV.1}$$

Rappelons que \mathbb{S}_{Pri} n'est pas connu *a priori*.

Ainsi, afin de garantir l'équivalence entre les deux modèles, un critère de sélection est adopté : **l'équipe fait l'hypothèse forte que pour une paire de modèles, une similarité entre les données de sortie suffit à prouver une similarité entre les données d'entraînement..** Ainsi :

$$\mathbb{S}'_{Pri_i} \approx \mathbb{S}_{Pri} \iff \mathbb{S}'_{Synth_i} \approx \mathbb{S}_{Synth}\tag{IV.2}$$

Ainsi, au-delà de la performance du modèle de classification, deux paramètres essentiels doivent retenir l'attention :

- La taille du dataset d'entraînement $l(\mathbb{S}'_{priv})$
- La similarité entre \mathbb{S}'_{Synth_i} et \mathbb{S}_{Synth}

Les sections suivantes proposent plusieurs approches avec différentes prises en compte de ces paramètres.

IV.1 Essai préliminaire avec un seul dataset (E_{T_2, \bar{C}, l_2})

Cette expérience sert avant tout à construire un framework d'étude et à appréhender les différentes composantes du problème. C'est celle-ci qui a été présentée lors de la restitution à mi-parcours du projet.

L'objectif est d'établir un point de départ sur lequel articuler des réflexions plus poussées. On choisit donc d'entraîner le classifieur sur un seul dataset $\mathbb{S}'_{priv}|l(\mathbb{S}'_{priv} = 1000)$. Ce dataset privé est obtenu en changeant la seed générative du DoppelGanger.

Cette première expérience permet d'obtenir $S_{T_2} = 0,47$. Ce score largement perfectible est partiellement du à un mauvais classifieur, mais son principal problème est un nombre trop faible de lignes. Les expériences suivantes cherchent donc à augmenter $l(\mathbb{S}'_{priv})$ en créant plus de Shadow Models.

IV.2 Regroupement de plusieurs datasets pour augmenter la taille du dataset d'entraînement ($E_{T_2, \bar{C}, l_{30}}$)

Cette opération est similaire à la première, mais on cherche à maximiser $l(\mathbb{S}'_{priv})$ pour faciliter l'entraînement du classifieur. Ainsi on construit 15 Shadows Models constitués chacun d'une paire $\mathbb{S}'_{priv_i} \rightarrow \mathbb{S}'_{synth_i}$.

Nb : On obtient \mathbb{S}'_{priv_i} en changeant la seed générative de \mathcal{G} .



FIGURE IV.1 – Distribution des datasets privés créés par l'équipe - $\mathfrak{H}\left(\mathbb{S}'_{Priv_{iT_2}}\right) \mid \left(E_{T_2, \bar{C}, l_{30}}\right)$

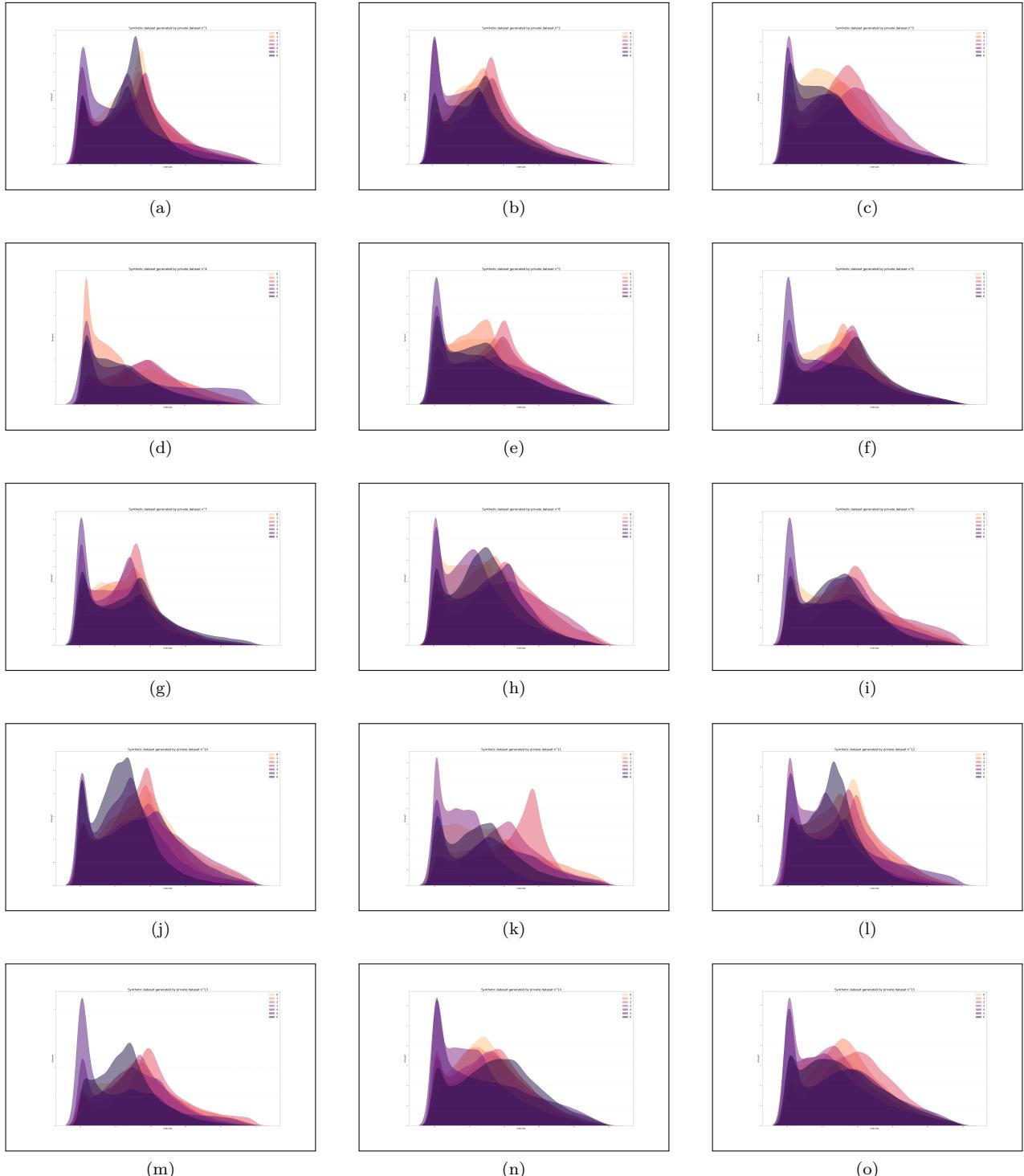


FIGURE IV.2 – Distribution des datasets synthétiques générés par les datasets privés - $\mathfrak{H}(\mathbb{S}'_{Synth_{iT_2}}) \mid (E_{T_2, \overline{C}, l_{30}})$

Ce changement d'échelle permet d'augmenter les performances du classifieur (*voir V*) sans pour autant se soucier davantage de la qualité des données fournies. Une approche naïve envisagée en début de projet était de comparer à l'oeil les distributions, et d'exclure arbitrairement les \mathbb{S}'_{priv_i} dont l'allure paraissait impropre à l'inclusion dans l'entraînement du classifieur. Une telle approche engendre de moins bons résultats. Ce problème s'éclaircit quand on visualise la divergence ^a des données utilisées avec $\mathbb{S}_{synth_{T_2}}$: en réalité tous les datasets sont relativement éloignés de celui que l'on cherche à reproduire ! Il faut donc affiner le critère de sélection si l'on veut discriminer certains d'entre eux.

a. voir section suivante pour le choix de la métrique



FIGURE IV.3 – $\mathfrak{D} \left(\mathbb{S}_{synth_{T_2}}, \mathbb{S}'_{synth_i} \right)$ pour $(E_{T_2, \bar{C}, l_{30}})$

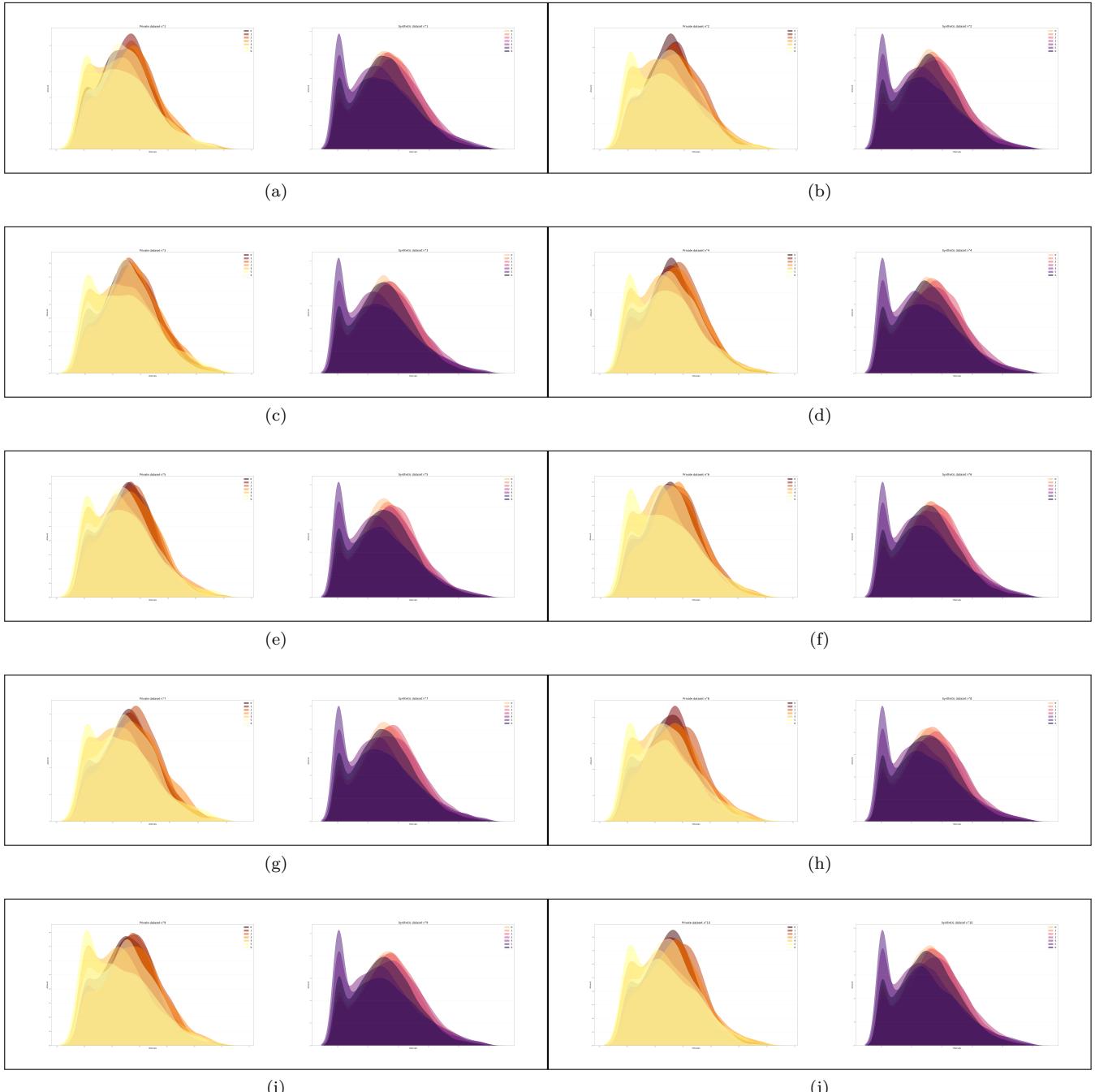


FIGURE IV.4 – $\mathfrak{H} \left(\mathbb{S}'_{Etr_{iT_1}}, \mathbb{S}'_{Synth_{iT_1}} \right) \mid \left(E_{T_1, \overline{C}, l_{200}} \right)$

IV.3 Regroupement d'un nombre important de datasets et nettoyage des données ($E_{T_2, \bar{C}, l_{100}}$)

Les résultats précédents montrent que l'augmentation de $l(\mathbb{S}'_{priv})$ est inefficace, (*voire contre-productive*) au-delà d'un certain seuil. On reprend alors les équations IV.1 et IV.2 et on cherche à construire \mathbb{S}'_{priv} avec des considérations plus fines.

IV.3.1 Métriques de comparaison pour piloter le nettoyage

En substance, il est aisément d'émettre la réflexion suivante : "Si les datasets synthétiques générés par nos modèles sont similaires aux datasets synthétiques donnés pour la compétition, alors on peut donner les données d'entraînement des modèles en entrée d'un classifieur et espérer raisonnablement qu'il soit adapté à l'attaque de \mathcal{G} ". En revanche, il est beaucoup plus difficile de vérifier la première partie de cette assertion.

Pour ce faire, on peut assimiler les séries temporelles à une distribution statistique (*à l'instar de notre procédé pour visualiser les données*), et les normaliser pour approximer une distribution probabiliste. Ainsi, il est possible d'utiliser un certain nombre de métriques pour apprécier les similitudes entre ces distributions. Deux d'entre elles ont été retenues : la divergence de Kullback-Leibler ([8]) et la divergence de Jensen-Shannon ([12]).

Leurs formules sont rappelées ci-dessous.

$$\mathfrak{D}(P||Q) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \quad (\text{IV.3})$$

$$\mathfrak{JSD}(P||Q) = \frac{1}{2}\mathfrak{D}(P||M) + \frac{1}{2}\mathfrak{D}(Q||M), \quad | M = \frac{1}{2}(P+Q) \quad (\text{IV.4})$$

La divergence de Jensen-Shannon possède des propriétés de symétrie et de finitude des valeurs qui rendent son utilisation plus facile que la divergence de Kullback-Leibler. C'est donc celle-ci que l'on utilisera et cherchera à minimiser (*une divergence nulle signifiant une exactitude entre les deux ensembles*).

On notera \mathfrak{D} la divergence de Jensen-Shannon par la suite. Comme un dataset est décomposé en 7 distributions, le calcul d'une divergence pour un dataset donne en sortie un 7-uplet. On peut donc visualiser la divergence d'une expérience entière (*ie un grand nombre de datasets*) avec la même représentation que précédemment :

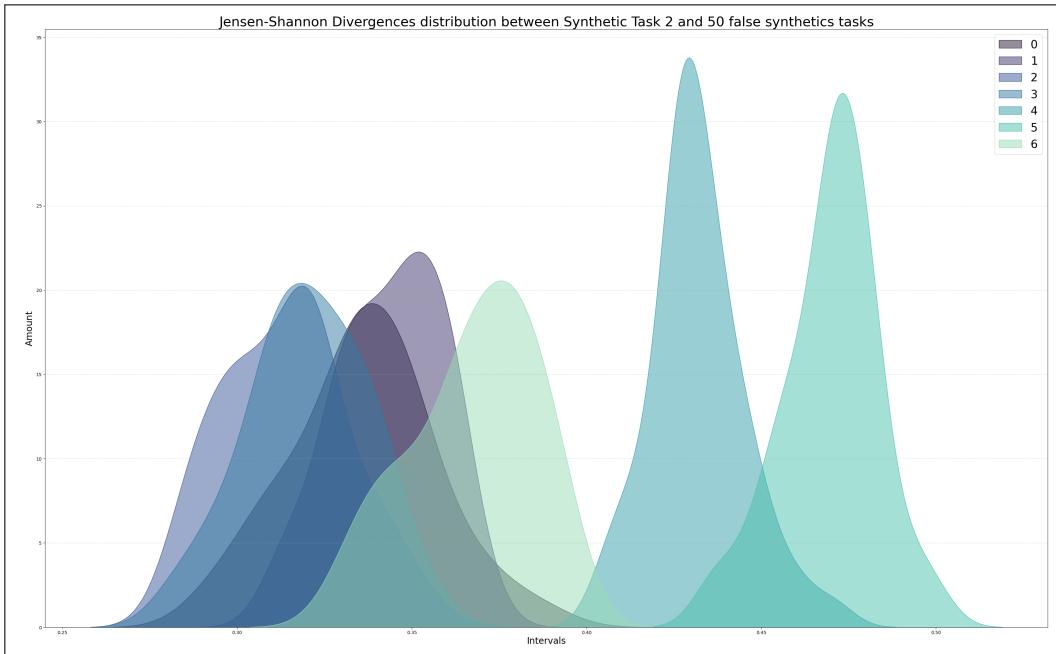


FIGURE IV.5 – $\mathfrak{D}(\mathbb{S}_{synth_{T_2}}, \mathbb{S}'_{synth_i})$ pour $(E_{T_2, \bar{C}, l_{100}})$

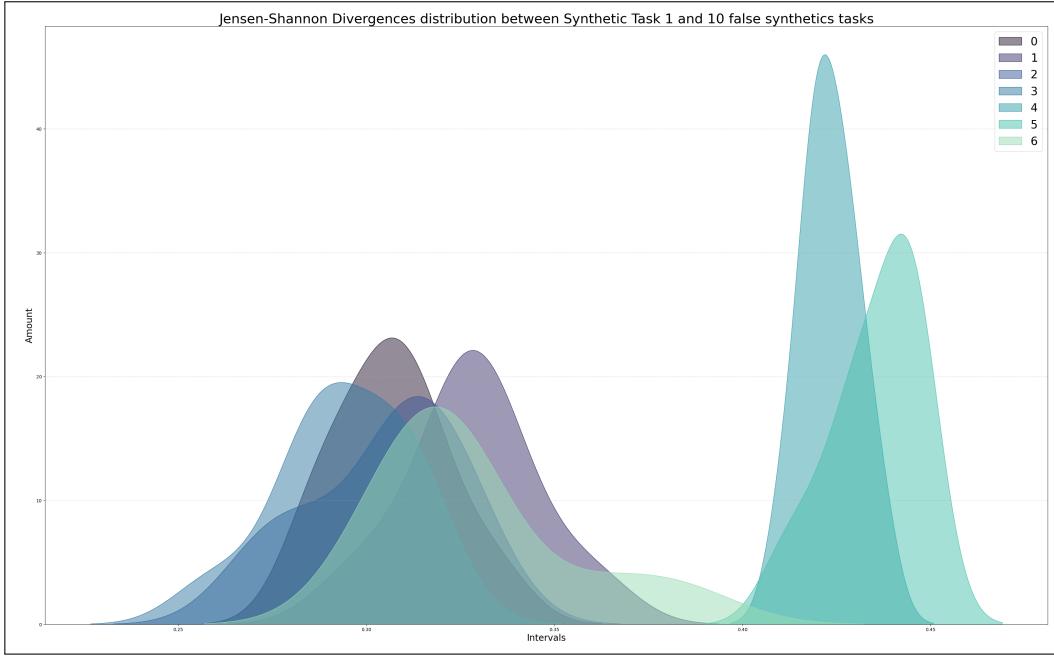


FIGURE IV.6 – $\mathfrak{D} \left(\mathbb{S}_{synthT_1}, \mathbb{S}'_{synth_i} \right)$ pour $(E_{T_1, \bar{C}, l_{200}})$

IV.3.2 Limite imposée par le temps de calcul

Un résultat inattendu de l'étude est la durée de calcul non négligeable qu'engendre une augmentation de $l(\mathbb{S}'_{priv})$. Ainsi, quand $l(\mathbb{S}'_{priv}) \approx 10^5$, $\mathcal{C}(\mathbb{C}) \approx 48h$. Un rapide relevé des temps de calcul laisse supposer une complexité quadratique pour les méthodes de classification utilisées, ce qui empêche de toute manière une classification basée seulement sur la quantité de données générées.

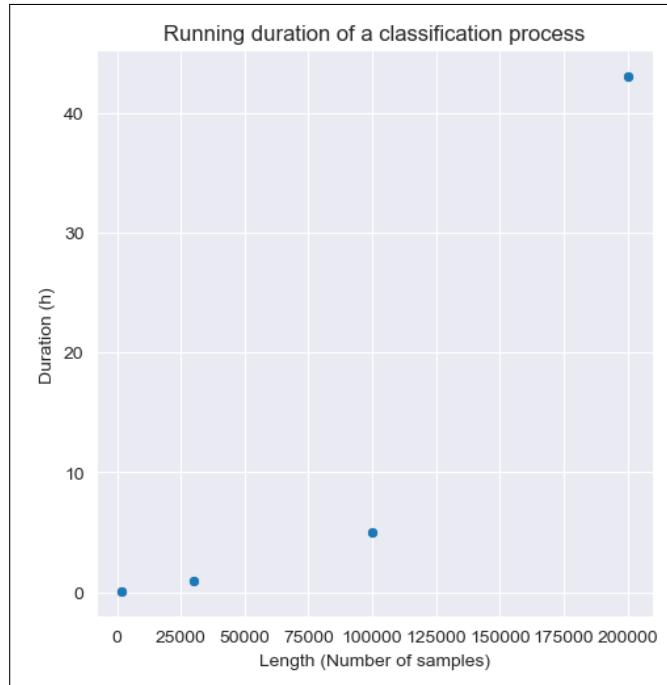


FIGURE IV.7 – Durée totale d'une classification en fonction de $l(\mathbb{S}_{etr})$

IV.3.3 Nettoyage de \mathbb{S}'_{priv_i}

Il s'agit de la partie la plus délicate de l'étude. En effet, on a réussi à générer de nombreux \mathbb{S}'_{priv_i} ¹ et à apprécier la divergence des données générées avec les données générées par le modèle attaqué, mais rien n'est à ce stade envisagé pour diminuer cette divergence.

La principale difficulté est que comme vu plus haut (IV.2), aucun \mathbb{S}'_{priv_i} généré n'est sensiblement plus divergent que ses homologues. Par ailleurs, la divergence varie d'une classe à l'autre : que faire d'un \mathbb{S}'_{priv_i} très divergent pour la classe 1 mais peu divergent pour la classe 4 ?

Aucune réponse à cette question non triviale n'a malheureusement été implémentée par l'équipe. Cependant, quelques pistes de réflexion ont été émises : si l'on génère un très grand nombre de \mathbb{S}'_{priv_i} (*par exemple quelques centaines*), on doit être en mesure de fixer un seuil de divergence pour lequel on est capable d'exclure suffisamment de \mathbb{S}'_{priv_i} pour respecter ce seuil, sans diminuer drastiquement $l(\mathbb{S}'_{priv})$. On peut définir plusieurs expériences : en fixant ce seuil pour la divergence moyenne des 7 classes, ou bien en considérant que son dépassement pour une seule classe invalide tout le dataset. On peut également imaginer reconstituer de nouveaux \mathbb{S}'_{priv_i} à partir de colonnes non-divergentes issues de plusieurs datasets. Cette méthode aurait probablement augmenté les scores de l'équipe, mais laisse à penser qu'elle aurait vite atteint ses limites. En effet, on remarque sur la figure IV.3 que la possibilité de générer des \mathbb{S}'_{priv_i} avec des divergences très faibles n'est pas prouvée, en particulier pour les classes 4 et 5.

IV.4 Construction de données d'entraînement par régression sur les Shadow Models

Une autre approche très élégante mais exigeante tant en puissance de calcul qu'en compétences en Machine Learning aurait été de ne pas se limiter à une classification et de considérer la recherche d'une relation mathématique concrète entre \mathbb{S}'_{priv_i} et \mathbb{S}'_{synth_i} . Rien n'empêche *a priori* d'utiliser un ou plusieurs algorithmes de régression pour construire cette relation, et si celle-ci est inversible, elle permet de fabriquer de faux \mathbb{S}'_{priv_i} très peu divergents du modèle observé. Ces datasets seraient alors optimaux pour entraîner les classifieurs.

Notons que cette méthode est hautement spéculative en l'état. De plus, elle est susceptible de ne pas être transposable à T_2 et T_4 car les données générées par \mathcal{G} quand il est entraîné sur 1000 lignes sont tout à fait différentes de ses données d'entraînement, ce qui restreint intuitivement la possibilité de déterminer cette relation (*cf figures IV.1 et IV.2*).

IV.5 Cas particulier des tâches 3 et 4 ($E_{T_3, \bar{C}, l_{60}}$)

La fabrication des \mathbb{S}'_{priv_i} est plus délicate ici car \mathcal{G} est entraîné sur un dataset partiellement connu.

Une étape importante supplémentaire importante est l'intégration d'un modèle LSTM pour compléter les données manquantes (jours 8-10). Cette méthode permet d'obtenir un dataset complet, essentiel pour entraîner les classifieurs. En effet, le modèle LSTM est entraîné sur les jours 1-7 des données publiques, et ses prédictions sont ajoutées aux données pour constituer un ensemble plus robuste. Cette étape améliore la diversité et la qualité des données disponibles pour les Shadow Models.

1. Cette partie est d'ailleurs semi-automatisée avec un script `Bash` écrit par l'équipe

Chapitre V

Méthodes de classification des données

Ce chapitre a pour but la sélection d'une ou plusieurs techniques de classification visant à attribuer la valeur 0 ou 1 à une classe que l'on créera et ajoutera aux ensembles \mathbb{S}_{etr} .

Pour confirmer les propos tenus ici, l'équipe se base principalement sur les pages Wikipedia des modèles concernés ([26, 6, 22, 29]) et sur la littérature d'introduction au Machine Learning ([5, 11]).

L'article expliquant le fonctionnement et l'intérêt du modèle DoppelGanger ([17]) est un autre guide pour la réflexion. On peut en effet en extraire cette figure :

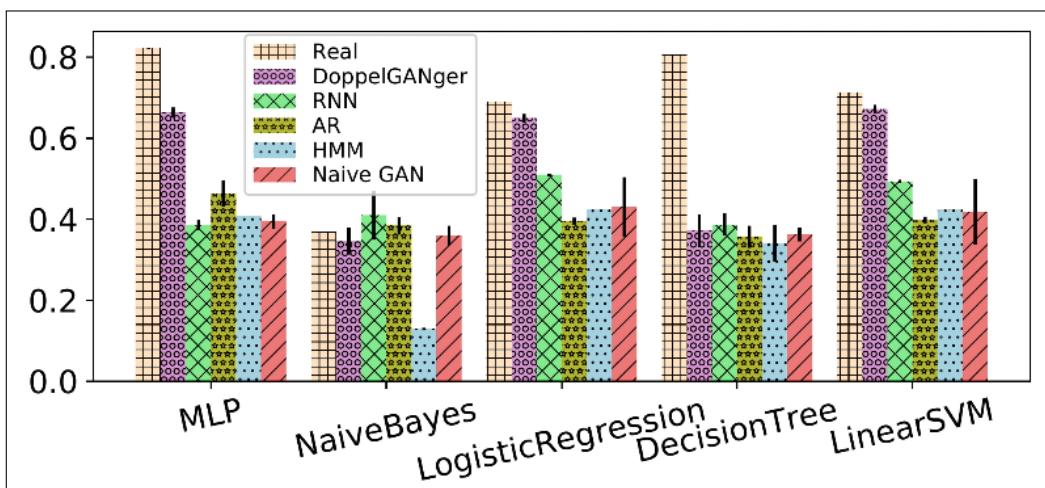


FIGURE V.1 – Taux de succès de différents modèles de classification, comparaison entre le DoppelGanger et d'autres générateurs

Une détermination rigoureuse d'un modèle de classification voudrait que l'on étudie également le meilleur ratio de membres et de non-membres. (*autrement dit de lignes labellées "0" et "1"*). On fait cependant l'hypothèse que les tâches \mathbb{C}_{T_i} comportent autant de membres que de non-membres.

Ainsi, on définit :

$$\mathbb{S}_{etr} = \mathbb{S}'_{pub} + \mathbb{S}'_{priv} \quad \mid \quad l(\mathbb{S}'_{pub}) = l(\mathbb{S}'_{priv}) \quad (V.1)$$

Avec \mathbb{S}'_{pub} composé de lignes choisies aléatoirement dans \mathbb{S}_{pub} mais n'appartenant pas à \mathbb{S}'_{priv} .

V.1 Choix de l'algorithme

Le meilleur modèle à utiliser pour classifier \mathbb{C}_{T_i} dépend de nombreux paramètres. Une longue partie du projet a été consacrée à la détermination de ce modèle en amont de son application à la compétition. Le retour d'expérience montre qu'il est en réalité préférable d'utiliser de nombreux modèles et de comparer leurs performances sur des cas concrets. En effet, les classificateurs proposés par `scikit-learn` sont conçus pour être utilisés en série, indépendamment de leur fonctionnement interne, et leur temps de calcul est raisonnable.

Les modèles retenus sont ceux présentés dans les tableaux V.1, V.2 et V.3¹.

Pour des raisons de simplicité, l'équipe n'a pas utilisé de réseaux de neurones pour la classification des données. Toutefois, on peut également justifier ce choix en considérant que pour un problème de classification binaire de séries temporelles, les algorithmes traditionnels de Machine Learning sont *a priori* efficaces.

V.2 Évaluation des classifieurs

V.2.1 Choix des métriques

Il est important de choisir plusieurs paramètres pour apprécier finement le comportement des classifieurs. Le premier utilisé par l'équipe est la matrice de confusion, que l'on normalise vis-à-vis de la taille du dataset de test. La matrice de confusion idéale est donc :

$$M_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (\text{V.2})$$

On calcule également les score de précision et de rappel pour tenir compte de l'influence de faux positifs et de faux négatifs respectivement :

$$\text{precision} = \frac{1}{1 + \frac{\tau_{fp}}{\tau_{tp}}} \quad (\text{V.3})$$

$$\text{rappel} = \frac{1}{1 + \frac{\tau_{fn}}{\tau_{tp}}} \quad (\text{V.4})$$

On calcule de plus le score f_1 , moyenne harmonique de la précision et du rappel.

$$f_1 = \frac{2 \times \tau_{tp}}{2 \times \tau_{tp} + \tau_{fp} + \tau_{fn}} \quad (\text{V.5})$$

Enfin, on calcule l'aire sous la courbe ROC de la classification.

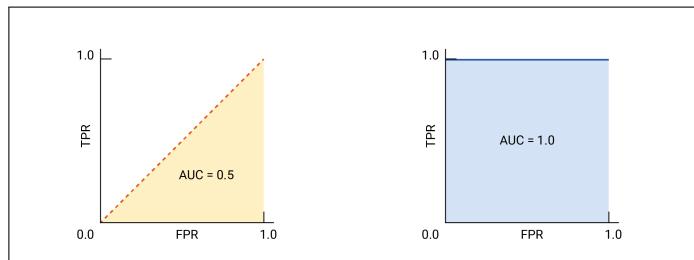


FIGURE V.2 – Score AUC pour une classification naïve et une classification parfaite

1. Un classifieur dit "empileur" prend les prédictions produites par d'autres modèles pour faire sa propre prédiction d'une cible ou d'un label. ([11], Chapitre 3)

V.2.2 Score obtenu pour chaque métrique

Modèle / Expérience	$E_{T_1, \bar{C}, l_{200}}$	E_{T_2, \bar{C}, l_2}	$E_{T_2, \bar{C}, l_{30}}$	$E_{T_2, \bar{C}, l_{100}}$	$E_{T_3, C}$	$E_{T_4, C}$
Régression logistique	0,50	X	0,56	0,50		
Arbre de décision	0,83	X	0,93	0,85		
Recherche des K plus proches voisins	0,63	X	0,79	0,64		
Bayésien naïf (<i>modèle gaussien</i>)	0,51	0,67	0,57	0,51		
Machine à vecteurs de support	0,50	X	0,74	0,53		
Forêt aléatoire	0,83	X	0,95	0,85		
Amplification de gradient	0,60	X	0,86	0,73		
Empilement	0,83	X	0,95	0,85		

TABLE V.1 – Moyenne harmonique du couple Précision-Rappel de différents classificateurs selon plusieurs configurations d’entraînement

Modèle / Expérience	$E_{T_1, \bar{C}, l_{200}}$	E_{T_2, \bar{C}, l_2}	$E_{T_2, \bar{C}, l_{30}}$	$E_{T_2, \bar{C}, l_{200}}$	$E_{T_3, C}$	$E_{T_4, C}$	$E_{T_4, \bar{C}}$
Régression logistique	0,500	X	0,58	0,50			
Arbre de décision	0,499	X	0,78	0,85			
Recherche des K plus proches voisins	0,501	X	0,80	0,64			
Bayésien naïf (<i>modèle gaussien</i>)	0,501	X	0,60	0,51			
Machine à vecteurs de support	0,501	X	0,82	0,53			
Forêt aléatoire	0,503	X	0,93	0,85			
Amplification de gradient	0,502	X	0,92	0,73			
Empilement	0,498	X	0,88	0,85			

TABLE V.2 – Aire sous la courbe ROC des différents classificateurs

Modèle / Expérience	$E_{T_1, \bar{C}, l_{200}}$	$E_{T_2, \bar{C}, l_{12}}$	$E_{T_2, \bar{C}, l_{30}}$	$E_{T_2, \bar{C}, l_{100}}$	$E_{T_3, \bar{C}}$	$E_{T_4, C}$
Régression logistique	$\begin{bmatrix} 0,31 & 0,19 \\ 0,32 & 0,20 \end{bmatrix}$	X	$\begin{bmatrix} 0,30 & 0,20 \\ 0,24 & 0,26 \end{bmatrix}$	$\begin{bmatrix} 0,31 & 0,19 \\ 0,32 & 0,19 \end{bmatrix}$		
Arbre de décision	$\begin{bmatrix} 0,26 & 0,24 \\ 0,26 & 0,24 \end{bmatrix}$	X	$\begin{bmatrix} 0,3 & 0,2 \\ 0,24 & 0,26 \end{bmatrix}$	$\begin{bmatrix} 0,25 & 0,24 \\ 0,26 & 0,25 \end{bmatrix}$		
Recherche des K plus proches voisins	$\begin{bmatrix} 0,25 & 0,25 \\ 0,25 & 0,25 \end{bmatrix}$	X	$\begin{bmatrix} 0,35 & 0,15 \\ 0,13 & 0,37 \end{bmatrix}$	$\begin{bmatrix} 0,25 & 0,24 \\ 0,25 & 0,26 \end{bmatrix}$		
Bayésien naïf (<i>modèle gaussien</i>)	$\begin{bmatrix} 0,32 & 0,18 \\ 0,32 & 0,18 \end{bmatrix}$	X	$\begin{bmatrix} 0,31 & 0,19 \\ 0,25 & 0,26 \end{bmatrix}$	$\begin{bmatrix} 0,32 & 0,17 \\ 0,33 & 0,18 \end{bmatrix}$		
Machine à vecteurs de support	$\begin{bmatrix} 0,27 & 0,23 \\ 0,27 & 0,23 \end{bmatrix}$	X	$\begin{bmatrix} 0,37 & 0,13 \\ 0,13 & 0,37 \end{bmatrix}$	$\begin{bmatrix} 0,30 & 0,19 \\ 0,31 & 0,2 \end{bmatrix}$		
Forêt aléatoire	$\begin{bmatrix} 0,26 & 0,24 \\ 0,26 & 0,24 \end{bmatrix}$	X	$\begin{bmatrix} 0,42 & 0,08 \\ 0,08 & 0,42 \end{bmatrix}$	$\begin{bmatrix} 0,26 & 0,23 \\ 0,27 & 0,23 \end{bmatrix}$		
Amplification de gradient	$\begin{bmatrix} 0,25 & 0,25 \\ 0,25 & 0,25 \end{bmatrix}$	X	$\begin{bmatrix} 0,42 & 0,08 \\ 0,09 & 0,41 \end{bmatrix}$	$\begin{bmatrix} 0,26 & 0,23 \\ 0,26 & 0,25 \end{bmatrix}$		
Empilement	$\begin{bmatrix} 0,20 & 0,21 \\ 0,29 & 0,3 \end{bmatrix}$	X	$\begin{bmatrix} 0,41 & 0,08 \\ 0,08 & 0,42 \end{bmatrix}$	$\begin{bmatrix} 0,20 & 0,3 \\ 0,20 & 0,31 \end{bmatrix}$		

TABLE V.3 – Matrices de confusion de différents classificateurs selon plusieurs configurations d’entraînement

V.3 Synthèse de l'utilisation des classifieurs

La phase de classification de l'attaque est globalement un succès. Les expériences montrent que pour une quantité suffisante de données d'entraînement, plusieurs classifieurs obtiennent des résultats satisfaisants, illustrés par leur métriques d'évaluation.

L'utilisation d'un grand nombre de modèles est intéressante car peu contraignante d'un point de vue de sa programmation² et éloquente en résultats. En effet, les performances des modèles varient d'une expérience à l'autre. On dégage quand-même des tendances avec une domination nette des arbres de décision, des forêts aléatoires, de l'amplification de gradient et plus ponctuellement de l'empilement de classifieurs³, infirmant les suppositions initiales selon lesquelles un bayésien naïf ou une régression logistique pouvait suffir, ces modèles plus simples étant largement éprouvés lorsque l'on augmente $l(\mathbb{S}_{etr})$. Pour l'ensemble des modèles, Le temps de calcul ne dépasse pas quelques heures si on fixe $l(\mathbb{S}_{etr}) = 100000$, longueur pour laquelle on constate de toute façon une stagnation, voire une baisse des performances.

À cet égard, il aurait été intéressant de dresser des courbes d'apprentissage⁴ afin de déterminer la taille limite $l(\mathbb{S}_{etr})$ pour laquelle les performances n'augmentent plus.

Cette phase présente néanmoins un écueil important : la précision des modèles, définie comme la capacité à ne pas prédire comme positif un négatif, est pour certaines expérimentations relativement faible. On obtient donc des classifieurs identifiant facilement tous les membres, mais peu fiables sur la prédiction des non-membres, avec un taux important de faux positifs. D'après la définition du scoring (*I.1*) de la compétition, ce problème est donc un frein non négligeable à l'augmentation de \mathbb{S}_{T_i}

2. Avec `scikit-learn`, on peut utiliser une boucle itérative sur le nombre de modèles que l'on souhaite. Voir le chapitre 3 de [11] pour l'algorithme remanié par l'équipe.

3. Ce résultat est par ailleurs en accord avec l'état de l'art du Machine Learning.

4. [11], Chapitre 3, section 7.4

Chapitre VI

Synthèse des résultats

Tâche	$s(T_1)$	$s(T_2)$	$s(T_3)$	$s(T_4)$
Score	0.52	0.56	0.50	0.50

TABLE VI.1 – Meilleurs scores obtenus sur chaque tâche

VI.1 Tâche 1

La taille des données d'entrée rend la classification particulièrement longue si l'on utilise des modèles réce®nts. Ainsi, l'équipe n'a réalisé qu'une classification, sur les mêmes hypothèses que la tâche 2. Le score obtenu est plutôt faible, ce qui suggère qu'une augmentation de la taille des données d'entraînement est requise.

VI.2 Tâche 2

On choisit d'aborder cette tâche en premier en raison de la petite taille de son dataset privé. En effet, on peut supposer que la variation d'un paramètre aura une plus forte sensibilité sur un petit ensemble. Sur un nombre raisonnable de datasets, les performances des classifieurs sont excellentes, sans que l'on ait étudié en détail la qualité des données d'entraînement. L'amélioration de cette qualité aurait sans doute permis d'augmenter significativement le score.

VI.3 Tâche 3

Cette tâche n'a pas été abordée car elle nécessite de longs temps de calcul et un retour d'expérience de la tâche 4, impossible compte tenu du stade du projet auquel nous l'avons étudiée.

VI.4 Tâche 4

Abordée très tardivement, cette tâche complexe aurait nécessité une prise de recul importante pour savoir comment exploiter les quelques résultats que nous avons pu en tirer. En effet, l'ensemble à classifier est différent des ensembles des tâches 1 et 2.

Conclusion

La compétition *Snake Strikes Back* constitue une entrée en matière intéressante pour des débutants en Machine Learning. Le problème posé est simple : les données sont normalisées, complètes, exclusivement numériques, et la classification est binaire. Pourtant, elle révèle que plusieurs approches sont possibles et qu'aucune ne permet de mettre complètement en échec le DoppelGanger.

L'équipe a connu un lancement difficile en mettant plusieurs semaines à télécharger les données nécessaires à la compétition. Ce retard pris au début du projet s'est inscrit en parallèle d'une découverte totale du Machine Learning et de difficultés de coordination sur les différents aspects du projet. De ce fait, de nombreuses hypothèses restent en suspens. En particulier, il est regrettable de ne pas avoir concrétisé les propositions exprimées dans les parties IV.3, IV.4 et IV.5. Une autre explication aux scores obtenus est la mauvaise utilisation des métriques d'évaluation des classifieurs. En effet celles-ci auraient mérité un croisement plus important pour étudier plus en profondeur la forte présence de faux positifs.

Enfin, sous l'hypothèse d'un lancement moins tardif et d'une montée en compétences plus rapide, l'équipe aurait probablement obtenu de meilleurs résultats en considérant des approches par Deep Learning. Il est notamment dommage de s'être restreint à une approche black-box alors que la compétition permettait une approche white-box, avec un accès complet aux scripts du DoppelGanger. Par ailleurs, l'utilisation de réseaux de neurones aurait pu être envisagée pour définir plus finement les Shadow Models, et définir de nouveaux modèles de classification. Cette approche aurait nécessité un approfondissement de la bibliographie, principalement pour l'article [17].

Troisième partie

Annexes

Annexe 1 : Programmes conçus par l'équipe

L'ensemble des programmes est disponible sur le Github de l'équipe. Les principales fonctions sont rappelées ici à titre informatif.

https://github.com/ThomInsa/MIA-SnakeStrikesBack-INSA_CVL.git

```
1 from attackDopel import *
2
3 toClass1 = targets_Task1.drop(['Unnamed: 0', 'index'], axis=1)
4 toClass2 = targets_Task2.drop(['Unnamed: 0', 'index'], axis=1)
5 toClass3 = targets_Task3.drop(['Unnamed: 0', 'index'], axis=1)
6 toClass4 = targets_Task4.drop(['Unnamed: 0', 'index'], axis=1)
7
8 def export_TargetsClassification(pathToResultsFolder, taskNumber, classifiers_collection,
9     predictions, title=""):
10    separator = "\n"
11    resultDateTime = dt.datetime.today().strftime('%Y-%m-%d-%H-%M-%S')
12    resultName = "results_Task" + str(taskNumber) + "_" + resultDateTime + ".txt"
13    resultPath = pathToResultsFolder + "Task " + str(taskNumber) + "/" + resultName
14    resultTitle = title + "Results of classifiers for Task" + str(taskNumber) + "(" +
15    resultDateTime + ")"
16    resultsFile = open(resultPath, 'w')
17
18    resultsFile.writelines(resultTitle)
19    resultsFile.writelines(separator)
20
21    modelIndex = 0
22    for model in classifiers_collection:
23        resultsFile.writelines(model.__class__.__name__ + " : \n----\n")
24        for i in range(len(predictions[modelIndex])):
25            resultsFile.writelines(str(predictions[modelIndex][i]) + "\n")
26        modelIndex += 1
27
28    return 0
29
30 def classifyTarget(taskNumber, classifier_collection):
31     predictions = []
32     toClass = targetToClassifyFromInt(taskNumber)
33
34     for model in classifier_collection:
35         predictions.append(model.predict(toClass.values))
36
37     return predictions
38
39 def targetToClassifyFromInt(taskNumber):
40     match taskNumber:
41         case 1:
42             return toClass1
43         case 2:
44             return toClass2
45         case 3:
46             return toClass3
47         case 4:
48             return toClass4
49         case _:
50             return "Incorrect task number"
```

Listing VI.1 – Fonctions permettant de classifier les fichiers targetsTask

```

1 import numpy as np
2 from sklearn import model_selection
3 from sklearn.dummy import DummyClassifier
4 from sklearn.linear_model import (
5     LogisticRegression,
6 )
7 from sklearn.metrics import precision_score, confusion_matrix
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.neighbors import (
10     KNeighborsClassifier,
11 )
12 from sklearn.naive_bayes import GaussianNB
13 from sklearn.svm import SVC
14 from sklearn.ensemble import (
15     RandomForestClassifier,
16 )
17 import xgboost
18 from mlxtend.classifier import (
19     StackingClassifier
20 )
21
22 import pandas as pd
23 import datetime as dt
24
25 def newClassifiersDatabase():
26     classifiersDict = {'Classifier':[], 'Name':[], 'f1':[], 'precision':[], 'confusionMatrix': []
27     , 'AUC':[], 'STD':[]}
28     return pd.DataFrame(classifiersDict)
29
30 def trainModels(modelsToUse, X_train, y_train):
31     classifiersCollection = []
32     for model in modelsToUse:
33         print("Training model : " + str(model.__name__))
34         cls = model()
35         cls.fit(X_train.values, y_train.values)
36         classifiersCollection.append(cls)
37     return classifiersCollection
38
39 def trainStackingClassifier(classifiersCollection, X_train, y_train):
40     stack = StackingClassifier(
41         classifiers=classifiersCollection,
42         meta_classifier=LogisticRegression(),
43     )
44     print("Training model : StackingClassifier")
45     stack.fit(X_train.values, y_train.values)
46     return stack
47
48 def appendStackingClassifier(classifiersCollection, modelsCollection, X_train, y_train):
49     stack = StackingClassifier(
50         classifiers=classifiersCollection,
51         meta_classifier=LogisticRegression(),
52     )
53     modelsCollection.append(stack)
54     stack.fit(X_train.values, y_train.values)
55     classifiersCollection.append(stack)
56     return classifiersCollection
57
58 def createCollection_Predictions(classifiersCollection, X_test):
59     predictionsCollection = []
60     for model in classifiersCollection:
61         prediction = pd.DataFrame(model.predict(X_test))
62         prediction.rename(columns={"0": "isMember"})
63         predictionsCollection.append(prediction)
64     return predictionsCollection
65
66 def createCollection_PrecisionScore(y_test, predictionsCollection):
67     precisionCollection = []
68     index = 0
69     for prediction in predictionsCollection:
70         precisionCollection.append(precision_score(y_test.values, predictionsCollection[index]))
71         index += 1
72     return precisionCollection
73
74 def createCollection_ROC(classifiersCollection, X, y, X_train, y_train, value = "mean"):
75     rocCollection = []

```

```

75     index = 0
76     for classifier in classifiersCollection:
77         s = model_selection.cross_val_score(classifier, X, y, scoring='roc_auc', cv=
78             model_selection.KFold(n_splits=10, random_state=42, shuffle=True))
79         if value == "mean":
80             rocCollection.append(s.mean())
81         elif value == "std":
82             rocCollection.append(s.std())
83         else:
84             print("Invalid value, choose mean or std for ROC computation")
85         return 1
86     index += 1
87     return rocCollection
88
89 def createCollection_ConfusionMatrices(y_test, predictionsCollection):
90     matricesCollection = []
91     for prediction in predictionsCollection:
92         matrix = confusion_matrix(y_test, prediction)
93         matrix = np.round(matrix / len(prediction), decimals=2)
94         matricesCollection.append(matrix)
95     return matricesCollection
96
97 def createClassifiersMetricsDB(classifiersCollection, X, y, precisionsCollection,
98     matricesCollection, ROCMCollection, ROCSCollection):
99     classifiersDB = newClassifiersDatabase()
100    index = 0
101    for classifier in classifiersCollection:
102        newRow = (classifier, classifier.__class__.__name__, classifier.score(X,y),
103        precisionsCollection[index], str((matricesCollection[index]).tolist()), ROCMCollection[
104        index], ROCSCollection[index])
105        classifiersDB.loc[len(classifiersDB)] = newRow
106        index += 1
107    return classifiersDB
108
109 def saveClassifiersMetricsDB(classifiersDB, directory_path, file_name):
110     now = dt.datetime.today().strftime('%Y-%m-%d-%H-%M-%S')
111     new_file_name = file_name + f"classifiersPerfs_{now}.csv"
112     file_path = f"{directory_path}{new_file_name}"
113     (classifiersDB.drop(columns=['Classifier'])).to_csv(file_path, index=False)

```

Listing VI.2 – Fonctions permettant de créer et d'entraîner des classifieurs

```

1 import os
2 import pandas as pd
3
4 from data.teamExperiments.attackDopel import npzPrivateToDataframe, publicData_Tasks12,
5     publicData_Tasks34
6
7 def createFinalTrainingSet(nonMembersPart, membersPart):
8     addMembershipToSubset(nonMembersPart, 0)
9     addMembershipToSubset(membersPart, 1)
10    trainingSet = pd.concat([nonMembersPart, membersPart])
11    return trainingSet
12
13 def makeMemberPart(privateDataset_Path, clean = False):
14     members_clean = pd.DataFrame()
15     members_unclean = pd.DataFrame()
16     setToExclude = []
17     for file in os.listdir(privateDataset_Path):
18         # print(file)
19         if file.endswith(".npz"):
20             members_unclean = pd.concat([members_unclean, npzPrivateToDataframe(os.path.join(
21                 privateDataset_Path, file))], ignore_index=True)
22             if file not in setToExclude:
23                 members_clean = pd.concat([members_clean, npzPrivateToDataframe(os.path.join(
24                     privateDataset_Path, file))], ignore_index=True)
25     if clean:
26         return members_clean
27     else:
28         return members_unclean
29
30
31 def makeNonMemberPart(memberPart, taskNumber):
32     nonMembers = pd.DataFrame()
33     if taskNumber == 1 or taskNumber == 2:
34         allNonMembers = pd.concat([publicData_Tasks12, memberPart, memberPart]).drop_duplicates(keep=False)
35     elif taskNumber == 3 or taskNumber == 4:
36         allNonMembers = pd.concat([publicData_Tasks34, memberPart, memberPart]).drop_duplicates(keep=False)
37     else:
38         print("Task number not recognized")
39         return 1
40     while len(nonMembers) < len(memberPart):
41         rowToAdd = allNonMembers.sample()
42         nonMembers = pd.concat([nonMembers, rowToAdd])
43     return nonMembers
44
45 def makeSubsetFromDataset(dataFrame, subsetstartIndex, subset endIndex_excluded):
46     # Make small datasets from the original dataset
47     return dataFrame.iloc[subsetstartIndex:subset endIndex_excluded].copy()
48
49 def addMembershipToSubset(subset, isMember):
50     newSubset = subset.copy()
51     if isMember == 0 or isMember == 1:
52         subset["isMember"] = isMember
53         return newSubset
54     else:
55         print("Second parameter must be 0 or 1. Your dataset hasn't changed")
56         return newSubset

```

Listing VI.3 – Fonctions permettant de créer les données d’entraînement des classifieurs

Annexe 2 : Framework du projet

Outils utilisés

La plupart des manipulations du projet ont été réalisées sur Dataspell, un IDE développé par Jetbrains permettant l'utilisation conjointe de notebooks Jupyter, de scripts Python, de différents formats de fichier (csv, npz, parquet) et d'outils pour le BigData.

La création des Shadow Models exige de se trouver dans un environnement Linux. La distribution utilisée est Mint 22.1. Les scripts créés par l'équipe pour automatiser cette création sont écrits en Bash.

Librairies Python

Librairie	Version	Build	Source
anyio	4.1.0	pyhd8ed1ab_0	conda-forge
appdirs	1.4.4	pypi_0	pypi
appnope	0.1.4	pyhd8ed1ab_0	conda-forge
argon2-cffi	21.3.0	pyhd8ed1ab_0	conda-forge
argon2-cffi-bindings	21.2.0	py311h3336109_5	conda-forge
argparse-dataclass	2.0.0	pypi_0	pypi
arrow	1.3.0	pyhd8ed1ab_0	conda-forge
asgiref	3.8.1	pypi_0	pypi
asttokens	2.4.1	pyhd8ed1ab_0	conda-forge
async-lru	1.0.3	pyhd8ed1ab_0	conda-forge
attrs	24.2.0	pyh71513ae_0	conda-forge
babel	2.16.0	pyhd8ed1ab_0	conda-forge
backcall	0.2.0	pyh9f0ad1d_0	conda-forge
beautifulsoup4	4.12.3	pyha770c72_0	conda-forge
bleach	6.2.0	pyhd8ed1ab_0	conda-forge
brotli	1.1.0	h00291cd_2	conda-forge
brotli-bin	1.1.0	h00291cd_2	conda-forge
brotli-python	1.1.0	py311hd89902b_2	conda-forge
bzip2	1.0.8	hfdf4475_7	conda-forge
ca-certificates	2024.12.14	h8857fd0_0	conda-forge
cached-property	1.5.2	hd8ed1ab_1	conda-forge
cached_property	1.5.2	pyha770c72_1	conda-forge
certifi	2024.12.14	pyhd8ed1ab_0	conda-forge
cffi	1.17.1	py311h137bacd_0	conda-forge
charset-normalizer	3.4.0	pyhd8ed1ab_0	conda-forge
colorama	0.4.6	pyhd8ed1ab_0	conda-forge
comm	0.2.2	pyhd8ed1ab_0	conda-forge
conda-inject	1.3.2	pypi_0	pypi
configargparse	1.7	pypi_0	pypi
connection-pool	0.0.3	pypi_0	pypi
contourpy	1.3.1	py311h4e34fa0_0	conda-forge
cycler	0.12.1	pyhd8ed1ab_0	conda-forge
cyrus-sasl	2.1.27	hf9bab2b_7	conda-forge
datrie	0.8.2	pypi_0	pypi
debugpy	1.8.9	py311hc356e98_0	conda-forge
decorator	5.1.1	pyhd8ed1ab_0	conda-forge

defusedxml	0.7.1	pyhd8ed1ab_0	conda-forge
dill	0.3.9	pypi_0	pypi
django	5.1.3	pypi_0	pypi
docutils	0.21.2	pypi_0	pypi
dpath	2.2.0	pypi_0	pypi
entrypoints	0.4	pyhd8ed1ab_0	conda-forge
exceptiongroup	1.2.2	pyhd8ed1ab_0	conda-forge
executing	2.3.2	pyhd8ed1ab_0	conda-forge
flit-core	3.10.1	pyhd8ed1ab_0	conda-forge
fonttools	4.55.0	py311ha3cf9ac_0	conda-forge
fqdn	1.5.1	pyhd8ed1ab_0	conda-forge
freetype	2.12.1	h60636b9_2	conda-forge
gitdb	4.0.11	pypi_0	pypi
gitpython	3.1.43	pypi_0	pypi
gputaskscheduler	0.1.0	dev_0	<develop>
h11	0.13.0	pyhd8ed1ab_0	conda-forge
h2	4.1.0	pyhd8ed1ab_0	conda-forge
hpack	4.0.0	pyh9f0ad1d_0	conda-forge
httpcore	1.0.7	pyh29332c3_1	conda-forge
httpx	0.27.2	pyhd8ed1ab_0	conda-forge
humanfriendly	10.0	pypi_0	pypi
hyperframe	6.0.1	pyhd8ed1ab_0	conda-forge
icu	73.2	hf5e326d_0	conda-forge
idna	3.10	pyhd8ed1ab_0	conda-forge
immutables	0.21	pypi_0	pypi
importlib-metadata	8.5.0	pyha770c72_0	conda-forge
importlib_resources	6.4.5	pyhd8ed1ab_0	conda-forge
ipykernel	6.29.5	pyh57ce528_0	conda-forge
ipython	8.11.0	pyhd1c38e8_0	conda-forge
ipywidgets	8.1.5	pyhd8ed1ab_0	conda-forge
isoduration	20.11.0	pyhd8ed1ab_0	conda-forge
jedi	0.19.2	pyhff2d567_0	conda-forge
jinja2	3.1.4	pyhd8ed1ab_0	conda-forge
joblib	1.4.2	pyhd8ed1ab_0	conda-forge
json5	0.10.0	pyhd8ed1ab_0	conda-forge
jsonpointer	3.0.0	py311h6eed73b_1	conda-forge
jsonschema	4.23.0	pyhd8ed1ab_0	conda-forge
jsonschema-specifications	2024.10.1	pyhd8ed1ab_0	conda-forge
jsonschema-with-format-nongpl	4.23.0	hd8ed1ab_0	conda-forge
jupyter	1.1.1	pyhd8ed1ab_0	conda-forge
jupyter-lsp	2.2.5	pyhd8ed1ab_0	conda-forge
jupyter_client	8.6.3	pyhd8ed1ab_0	conda-forge
jupyter_console	6.6.3	pyhd8ed1ab_0	conda-forge
jupyter_core	5.7.2	pyh31011fe_1	conda-forge
jupyter_events	0.10.0	pyhd8ed1ab_0	conda-forge
jupyter_server	2.14.2	pyhd8ed1ab_0	conda-forge
jupyter_server_terminals	0.5.3	pyhd8ed1ab_0	conda-forge
jupyterlab	4.3.4	pyhd8ed1ab_0	conda-forge
jupyterlab_pygments	0.3.0	pyhd8ed1ab_1	conda-forge
jupyterlab_server	2.27.3	pyhd8ed1ab_0	conda-forge
jupyterlab_widgets	3.0.13	pyhd8ed1ab_0	conda-forge
keras	2.13.1	pyhd8ed1ab_0	conda-forge
kiwisolver	1.4.7	py311hf2f7c97_0	conda-forge
krb5	1.21.3	h37d8d59_0	conda-forge
lcms2	2.16	ha2f27b4_0	conda-forge
lerc	4.0.0	hb486fe8_0	conda-forge
libblas	3.9.0	25_osx64_openblas	conda-forge
libbrotlicommon	1.1.0	h00291cd_2	conda-forge
libbrotlidec	1.1.0	h00291cd_2	conda-forge
libbrotlienc	1.1.0	h00291cd_2	conda-forge
libcblas	3.9.0	25_osx64_openblas	conda-forge

libcxx	19.1.3	hf95d169_0	conda-forge
libdeflate	1.22	h00291cd_0	conda-forge
libedit	3.1.20191231	h0678c8f_2	conda-forge
libffi	3.4.2	h0d85af4_5	conda-forge
libgfortran	5.0.0	13_2_0_h97931a8_3	conda-forge
libgfortran5	13.2.0	h2873a65_3	conda-forge
libiconv	1.17	hd75f5a5_2	conda-forge
libjpeg-turbo	3.0.0	h0dc2134_1	conda-forge
liblapack	3.9.0	25_osx64_openblas	conda-forge
libllvm18	18.1.8	h9ce406d_2	conda-forge
libntlm	1.4	h0d85af4_1002	conda-forge
libopenblas	0.3.28	openmp_hbf64a52_1	conda-forge
libpng	1.6.44	h4b8f8c9_0	conda-forge
libsodium	1.0.20	hfdf4475_0	conda-forge
sqlite	3.47.0	h2f8c449_1	conda-forge
libtiff	4.7.0	h583c2ba_1	conda-forge
libwebp-base	1.4.0	h10d778d_0	conda-forge
libxcb	1.17.0	hf1f96e2_0	conda-forge
libxgboost	2.1.3	cpu_h63c3a07_0	conda-forge
libxml2	2.12.7	hc603aa4_3	conda-forge
libzlib	1.3.1	hd23fc13_2	conda-forge
llvm-openmp	19.1.3	hf78d878_0	conda-forge
markupsafe	3.0.2	py311h8b4e8a7_0	conda-forge
matplotlib	3.9.2	py311h6eed73b_2	conda-forge
matplotlib-base	3.9.2	py311h8b21175_2	conda-forge
matplotlib-inline	0.1.7	pyhd8ed1ab_0	conda-forge
mistune	3.0.2	pyhd8ed1ab_0	conda-forge
mlxtend	0.23.3	pypi_0	pypi
multiprocess	0.70.17	pypi_0	pypi
munkres	1.1.4	pyh9f0ad1d_0	conda-forge
nbclient	0.10.0	pyhd8ed1ab_0	conda-forge
nbconvert-core	7.16.4	pyhd8ed1ab_1	conda-forge
nbformat	5.10.4	pyhd8ed1ab_0	conda-forge
ncurses	6.5	hf036a51_1	conda-forge
nest-asyncio	1.6.0	pyhd8ed1ab_0	conda-forge
notebook	7.3.2	pyhd8ed1ab_0	conda-forge
notebook-shim	0.2.4	pyhd8ed1ab_0	conda-forge
numpy	1.26.4	py311hc43a94b_0	conda-forge
openjpeg	2.5.2	h7310d3a_0	conda-forge
openldap	2.6.8	hcd2896d_0	conda-forge
openssl	3.4.0	hd471939_0	conda-forge
overrides	7.7.0	pyhd8ed1ab_0	conda-forge
packaging	24.2	pyhff2d567_1	conda-forge
pandas	2.2.3	py311haeb46be_1	conda-forge
pandocfilters	1.5.0	pyhd8ed1ab_0	conda-forge
parso	0.8.4	pyhd8ed1ab_0	conda-forge
pathos	0.3.3	pypi_0	pypi
patsy	1.0.1	pyhff2d567_0	conda-forge
pexpect	4.9.0	pyhd8ed1ab_0	conda-forge
pickleshare	0.7.5	py_1003	conda-forge
pillow	11.0.0	py311h1f68098_0	conda-forge
pip	24.3.1	pyh8b19718_0	conda-forge
pkgutil-resolve-name	1.3.10	pyhd8ed1ab_1	conda-forge
plac	1.4.3	pypi_0	pypi
platformdirs	4.3.6	pyhd8ed1ab_0	conda-forge
pox	0.3.5	pypi_0	pypi
ppft	1.7.6.9	pypi_0	pypi
prometheus_client	0.21.0	pyhd8ed1ab_0	conda-forge
prompt-toolkit	3.0.48	pyha770c72_0	conda-forge
prompt_toolkit	3.0.48	hd8ed1ab_0	conda-forge
psutil	6.1.0	py311h1314207_0	conda-forge

pthread-stubs	0.4	h00291cd_1002	conda-forge
ptyprocess	0.7.0	pyhd3deb0d_0	conda-forge
pulp	2.9.0	pypi_0	pypi
pure_eval	0.2.3	pyhd8ed1ab_0	conda-forge
py-xgboost	2.1.3	cpu_pyhd3de297_0	conda-forge
pyarrow	18.0.0	pypi_0	pypi
pycparser	2.22	pyhd8ed1ab_0	conda-forge
pygments	2.18.0	pyhd8ed1ab_0	conda-forge
pyobjc-core	10.3.1	py311hd6939f8_1	conda-forge
pyobjc-framework-cocoa	10.3.1	py311hd6939f8_1	conda-forge
pyparsing	3.2.0	pyhd8ed1ab_1	conda-forge
pysocks	1.7.1	pyha2e5f31_6	conda-forge
python	3.11.0	he7542f4_1_cpython	conda-forge
python-dateutil	2.9.0.post0	pyhff2d567_0	conda-forge
python-fastjsonschema	2.20.0	pyhd8ed1ab_0	conda-forge
python-json-logger	2.0.7	pyhd8ed1ab_0	conda-forge
python-tzdata	2024.2	pyhd8ed1ab_0	conda-forge
python_abi	3.11	5_cp311	conda-forge
pytz	2024.1	pyhd8ed1ab_0	conda-forge
pyyaml	6.0.2	py311h3336109_1	conda-forge
pyzmq	26.2.0	py311h4d3da15_3	conda-forge
qhull	2020.2	h3c5361c_5	conda-forge
readline	8.2	h9e318b2_1	conda-forge
referencing	0.35.1	pyhd8ed1ab_0	conda-forge
requests	2.32.3	pyhd8ed1ab_0	conda-forge
rretry	0.11.8	pypi_0	pypi
rfc3339-validator	0.1.4	pyhd8ed1ab_0	conda-forge
rfc3986-validator	0.1.1	pyh9f0ad1d_0	conda-forge
rpds-py	0.21.0	py311h3b9c2be_0	conda-forge
scikit-learn	1.5.2	py311ha1d5734_1	conda-forge
scipy	1.14.1	py311hed734c1_1	conda-forge
seaborn	0.13.2	hd8ed1ab_2	conda-forge
seaborn-base	0.13.2	pyhd8ed1ab_2	conda-forge
send2trash	1.8.3	pyh31c8845_0	conda-forge
setuptools	75.5.0	pyhff2d567_0	conda-forge
six	1.16.0	pyh6c4a22f_0	conda-forge
smart-open	7.0.5	pypi_0	pypi
smmmap	5.0.1	pypi_0	pypi
snakemake	8.25.3	pypi_0	pypi
snakemake-interface-common	1.17.4	pypi_0	pypi
snakemake-interface-executor-plugins2	0.1.0	pypi_0	pypi
snakemake-interface-report-plugins1.0	0.1.0	pypi_0	pypi
snakemake-interface-storage-plugins3.0	0.1.0	pypi_0	pypi
sniffio	1.3.1	pyhd8ed1ab_0	conda-forge
soupsieve	2.5	pyhd8ed1ab_1	conda-forge
sqlparse	0.5.2	pypi_0	pypi
stack_data	0.6.2	pyhd8ed1ab_0	conda-forge
statsmodels	0.14.4	py311h0034819_0	conda-forge
tabulate	0.9.0	pypi_0	pypi
tensorflow-addons	0.23.0.dev0	pypi_0	pypi
terminado	0.18.1	pyh31c8845_0	conda-forge
threadpoolctl	3.5.0	pyhc1e730c_0	conda-forge
throttler	1.2.2	pypi_0	pypi
time	1.8	h10d778d_0	conda-forge
tinycss2	1.4.0	pyhd8ed1ab_0	conda-forge
tk	8.6.13	h1abcd95_1	conda-forge
tomli	2.1.0	pyhff2d567_0	conda-forge
tornado	6.4.1	py311h3336109_1	conda-forge
tqdm	4.67.0	pyhd8ed1ab_0	conda-forge
traitlets	5.14.3	pyhd8ed1ab_0	conda-forge
typeguard	2.13.3	pypi_0	pypi

types-python-dateutil	2.9.0.20241003	pyhff2d567_0	conda-forge
typing	3.10.0.0	pyhd8ed1ab_1	conda-forge
typing_extensions	4.12.2	hd8ed1ab_0	conda-forge
typing_extensions	4.12.2	pyha770c72_0	conda-forge
typing_utils	0.1.0	pyhd8ed1ab_0	conda-forge
tzdata	2024b	hc8b5060_0	conda-forge
unicodedata2	15.1.0	py311h1314207_1	conda-forge
uri-template	1.3.0	pyhd8ed1ab_0	conda-forge
urllib3	2.2.3	pyhd8ed1ab_0	conda-forge
wcwidth	0.2.13	pyhd8ed1ab_0	conda-forge
webcolors	24.8.0	pyhd8ed1ab_0	conda-forge
webencodings	0.5.1	pyhd8ed1ab_2	conda-forge
websocket-client	1.8.0	pyhd8ed1ab_0	conda-forge
wheel	0.45.0	pyhd8ed1ab_0	conda-forge
widgetsnbextension	4.0.13	pyhd8ed1ab_0	conda-forge
xgboost	2.1.3	cpu_pyhac85b48_0	conda-forge
xorg-libxau	1.0.11	h00291cd_1	conda-forge
xorg-libxdmcp	1.1.5	h00291cd_0	conda-forge
xz	5.2.6	h775f41a_0	conda-forge
yaml	0.2.5	h0d85af4_2	conda-forge
yte	1.5.4	pypi_0	pypi
zeromq	4.3.5	h7130eaa_7	conda-forge
zipp	3.21.0	pyhd8ed1ab_0	conda-forge
zstandard	0.23.0	py311hdf6fcfd6_1	conda-forge
zstd	1.5.6	h915ae27_0	conda-forge

TABLE VI.2 – Liste exhaustive des librairies Python utilisées par le projet

Annexe 3 : Chronologie et contributions

Chronologie

Avant la soutenance intermédiaire

Le travail réalisé en amont de la soutenance a principalement été l'appropriation du sujet et des différentes données de la compétition.

Après la soutenance intermédiaire

Les expériences ont par la suite été multipliées, en changeant la taille des données d'entraînement du classifieur et en évaluant la différence de comportement des Shadow Models par rapport au modèle initial. De nombreuses autres hypothèses ont été émises (exploration des tâches 3 et 4, construction des données d'entraînement par régression sur les données initiales), mais aucun cap clair n'a été défini, ne permettant pas de privilégier une piste et d'aboutir à de nouvelles conclusions.

Évaluations par les pairs et auto-évaluations

Les membres présents sur une ligne reçoivent les notes attribuées par les membres dans les colonnes associées.

Membre	Thomas	Selyan	Moussa	Émile
Thomas	16	16	16	16
Selyan	15	14	15	14
Moussa	13	13,5	13	13
Émile	13	13,5	13	13

TABLE VI.3 – Évaluations par les pairs et auto-évaluation

Quatrième partie

Bibliographie

Fondamentaux de mathématiques, de programmation et de Machine Learning

- [5] Chloé-Agathe AZENCOTT. *Introduction au Machine Learning*. (2nd). InfoSup. Dunod, fév. 2022.
- [11] Matt HARRISON. *Machine Learning - Les Fondamentaux. Exploiter des données structurées en Python*. O'Reilly, 2020.
- [13] Benjamin JOURDAIN. *Probabilités et statistiques pour l'ingénieur*. Jan. 2018.
- [19] *Machine Learning*. Page Wikipedia du Machine Learning. Nov. 2024. URL : https://en.wikipedia.org/wiki/Machine_learning.
- [25] Simon J.D. PRINCE. *Understanding Deep Learning*. MIT, 2024.

Sur le Machine Learning Antagoniste (*Adversarial Machine Learning*)

- [1] *Adversarial Machine Learning*. Page Wikipedia de l'Adversarial Machine Learning. Nov. 2024. URL : https://en.wikipedia.org/wiki/Adversarial_machine_learning#Adversarial_attacks_and_training_in_linear_models.
- [2] Tristan ALLARD et Mathias BERNARD. « Snakes Strikes Back ». In : (oct. 2024).
- [4] AUTHOR. *Membership inference attacks from first principles*. How published. Some note. Month Year. URL : <https://www.youtube.com/watch?v=1CNxfhMlk-A>.
- [10] *Generative adversarial network*. Page Wikipedia du modèle GAN. Nov. 2024. URL : https://en.wikipedia.org/wiki/Generative_adversarial_network.
- [17] Zinan LIN et al. « Using GANs for Sharing Networked Time Series Data : Challenges, Initial Promise, and Open Questions ». In : (jan. 2021). Présentation du modèle DoppelGANger. URL : <https://arxiv.org/abs/1909.13403>.
- [27] Reza SHOKRI. *Membership Inference Attacks against Machine Learning Models*. Vidéo de vulgarisation du papier du même nom. Mai 2017. URL : <https://www.youtube.com/watch?v=rDm1n2gceJY&t=53s>.
- [28] Reza SHOKRI et al. « Membership Inference Attacks Against Machine Learning Models ». In : () .

Ressources diverses pour le Machine Learning

- [3] Tatev ASLANYAN. *Machine Learning in 2024 – Beginner’s Course*. Fév. 2024. URL : <https://www.youtube.com/watch?v=bmmQA8A-yUA&t=1769s>.
- [6] *Classification naïve bayésienne*. Page Wikipedia du modèle bayésien naïf. Août 2024. URL : https://fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne.
- [7] *Découvrez les cellules à mémoire interne : les LSTM*. Jan. 2025. URL : <https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814656-decouvrez-les-cellules-a-memoire-interne-les-lstm>.
- [8] *Divergence de Kullback-Leibler*. Page Wikipedia de la Divergence de Kullback-Leibler. Sept. 2024. URL : https://fr.wikipedia.org/wiki/Divergence_de_Kullback-Leibler.
- [9] *Gaussian Naive Bayes Explained With Scikit-Learn*. Tutoriel pour construire un classifieur bayésien naïf. Nov. 2023. URL : <https://builtin.com/artificial-intelligence/gaussian-naive-bayes>.
- [12] *Jensen-Shannon divergence*. Page Wikipedia de la Divergence de Jensen-Shannon. Oct. 2024. URL : https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence#Applications.
- [14] LEARNDATAA. *81 Scikit-learn 78 Supervised Learning 56 Naive Bayes classifiers*. Youtube. 2021. URL : <https://www.youtube.com/watch?v=9Tmnr4L5Z0Q>.
- [15] *Les réseaux de neurones récurrents : des RNN simples aux LSTM*. Oct. 2019. URL : <https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm>.
- [16] *LES RNN, LES LSTM, LES GRU ET ELMO*. Déc. 2019. URL : <https://lbourdois.github.io/blog/nlp/RNN-LSTM-GRU-ELMO/>.
- [18] *Long Short Term Memory (LSTM) : de quoi s’agit-il ?* Oct. 2023. URL : <https://datascientest.com/long-short-term-memory-tout-savoir>.
- [20] MARKOVML. *Comparing and Evaluating Datasets : A Simplified Guide*. 24 nov. 2024. URL : <https://www.markovml.com/blog/compare-datasets>.
- [21] Boris MEINARDUS. *How I’d learn ML in 2024 (if I could start over)*. Youtube. 2024. URL : <https://www.youtube.com/watch?v=gUmagAluXpk>.
- [22] *Méthode des k plus proches voisins*. Page Wikipedia anglophone de la recherche des plus proches voisins. Mars 2024. URL : https://fr.wikipedia.org/wiki/M%C3%A9thode_des_k_plus_proches_voisins.
- [24] *Overfitting*. Page Wikipedia de l’Overfitting. Nov. 2024. URL : https://en.wikipedia.org/wiki/Overfitting#Machine_learning.
- [26] *Régression logistique*. Page Wikipedia de la régression logistique. Nov. 2024. URL : https://fr.wikipedia.org/wiki/R%C3%A9gression_logistique.
- [29] *Training, validation, and test data sets*. Page Wikipedia rappelant la différence entre les datasets d’entraînement et de test. Nov. 2024. URL : https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets.
- [30] *Understanding LSTM and its diagrams*. Mars 2016. URL : <https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714>.