

# RAPPORT DE PROJET

TER

---

## Étude sur la Génération de données, numériques, textuelles, visuelles.

---



Temouden Kaotar

Lehouaoui Sara

M1 Informatique IASD

[gitlab link](#)

Encadrant :

Pascal Poncelet

Année 2021/2022

*Nous tenons à remercier notre encadrant  
M. Pascal Poncelet de nous avoir accompagné  
durant la totalité de ce projet.*

# Sommaire

<b>1 Présentation du sujet de TER</b>	<b>4</b>
1.1 Introduction . . . . .	4
<b>2 Données numériques</b>	<b>7</b>
2.1 Classification des données réelles . . . . .	7
2.2 Génération de données à partir de moyenne et écart type . . . . .	9
2.2.1 Évaluation de la prédiction . . . . .	9
2.2.2 Évaluation dans l'apprentissage . . . . .	10
2.3 Utilisation de SMOTE . . . . .	11
2.3.1 Évaluation de la prédiction . . . . .	11
2.3.2 Évaluation dans l'apprentissage . . . . .	12
2.4 Utilisation de SDV . . . . .	12
2.4.1 Évaluation de la prédiction . . . . .	13
2.4.2 Évaluation dans l'apprentissage . . . . .	13
2.5 Discussion . . . . .	14
<b>3 Données Images</b>	<b>15</b>
3.1 Rappel sur les CNN et les réseaux antagonistes génératifs . . . . .	15
3.1.1 Architectures classiques des réseaux de neurones . . . . .	15
3.1.2 Réseau neuronal convolutif CNN . . . . .	16
3.1.3 Réseau Antagoniste Génératif GAN . . . . .	17
3.2 MNIST : Ensemble de données de chiffres manuscrits . . . . .	18
3.3 Discussion . . . . .	19

<b>4</b>	<b>Données textuelles</b>	<b>21</b>
4.1	Rappel sur les réseaux neuronaux récurrents et les réseaux LSTM . . . . .	21
4.1.1	Recurrent neural network RNN . . . . .	21
4.1.2	Les réseaux LSTM . . . . .	23
4.2	Classification des données réelles . . . . .	24
4.3	Génération des données textuelles avec LSTM . . . . .	25
4.3.1	Évaluation du modèle . . . . .	26
4.3.2	Discussion . . . . .	27
<b>5</b>	<b>Gestion de projet</b>	<b>28</b>
5.1	Cahier des charges . . . . .	28
5.2	Outils utilisés . . . . .	28
<b>6</b>	<b>Conclusion</b>	<b>30</b>
6.1	Pour aller plus loin . . . . .	30
6.2	Notre expérience avec ce TER . . . . .	30

# Partie 1

## Présentation du sujet de TER

### 1.1 Introduction

L'apprentissage automatique est une technologie révolutionnaire qui constitue actuellement un aspect essentiel de nombreuses industries en plein essor. En effet, l'utilisation de méthodes statistiques permet aux algorithmes de s'entraîner et à effectuer des classifications ou des prévisions, ce qui induit la découverte d'informations essentielles dans le cadre de projets d'exploration des données.

Ces informations permettent ensuite de prendre des décisions dans les applications et les entreprises, et ont idéalement un impact sur les principales métriques de croissance. Par exemple grâce à la performance de la classification des images, on arrive à mieux classer des images de série de cancer du poumon qu'un être humain d'après [cet article](#).

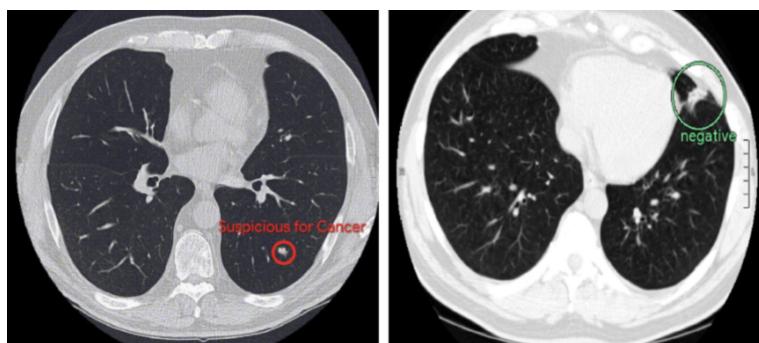


FIGURE 1.1 – Détection du cancer de poumon par l'IA (Source : venturebeat.com)

Le modèle d'apprentissage en profondeur a été utilisé pour prédire si un patient a un cancer du poumon, générer un score de risque de malignité du cancer du poumon du patient et identifier l'emplacement du tissu malin dans les poumons.

On se situe dans ce projet dans un cadre de classification supervisée, l'objectif de la classification supervisée est principalement de définir des règles permettant de classer des objets dans des classes à partir de variables qualitatives ou quantitatives caractérisant ces objets. Prenons un exemple où on considère une base de données avec quatre attributs ciel, température, vent et sport. Pour chacun des objets on a une étiquette sport avec une valeur positive voulant dire

que la personne peut sortir faire du sport selon le climat ou comme valeur négative. Dans le premier cas, nous avons un jeu de données équilibré avec 5 tuples portant la valeur positive pour sport et 5 autres tuples avec la valeur négative.

Ciel	Température	Vent	Sport
Ensoleillé	25°	Faible	OUI
Ensoleillé	26°	Faible	OUI
Couvert	20°	Faible	OUI
Ensoleillé	20°	Faible	OUI
Couvert	23°	Faible	OUI

(a) Tuples positifs

Ciel	Température	Vent	Sport
Pluie	25°	Fort	NON
Pluie	26°	Fort	NON
Couvert	20°	Fort	NON
Pluie	20°	Fort	NON
Pluie	23°	Fort	NON

(b) Tuples négatifs

FIGURE 1.2 – Jeu de données équilibré

On va apprendre nos données à partir d'un modèle de classification puis on va être capable de prédire si une personne pourra faire du sport ou non selon le climat. Intuitivement, on a un bon modèle avec une bonne classification.

Malheureusement, on se retrouve parfois dans le cas suivant où on a des tuples d'une classe plus que l'autre :

Ciel	Température	Vent	Sport
Ensoleillé	25°	Faible	OUI
Ensoleillé	26°	Faible	OUI
Couvert	20°	Faible	OUI
Ensoleillé	20°	Faible	OUI
Couvert	23°	Faible	OUI
Couvert	20°	Faible	OUI
Ensoleillé	20°	Faible	OUI
Ensoleillé	22°	Faible	OUI
Ensoleillé	22°	Faible	OUI

(a) Tuples positifs

Ciel	Température	Vent	Sport
Pluie	25°	Fort	NON

(b) Tuples négatifs

FIGURE 1.3 – Jeu de données déséquilibré

Le modèle va forcément bien apprendre quand faire du sport et mal apprendre quand ne pas faire parce qu'il n'y a pas assez d'éléments, ce n'est donc pas généralisable.

Notre problématique repose donc sur ce contexte, c'est à dire avoir des classes très déséquilibrées. Par ailleurs, on sait qu'en classification, avoir des classes déséquilibrées aura des impacts forts sur le résultat de la classification comme intuitivement on peut le percevoir.

Afin de résoudre ce problème, nous avons exploré différentes solutions :

- Demander à un expert, dans certains cas ce n'est pas faisable, on a pas d'autres données de ce type.
- Diminuer la classe Sport, la classe avec le plus de tuples, cette solution n'est pas optimale car des données vont être supprimées.
- Dupliquer les données déjà présents, dans notre exemple on a une seule donnée, ça ne sera donc pas intéressant de recopier la même donnée 9 fois.

- Génération des données, qui sera donc la solution qui convient le mieux pour pallier aux classes déséquilibrées.

Dans le cadre de ce TER, on s'intéresse de plus près à cette dernière solution, on veut générer des données de différents types : numériques, images et textuelles.

Vous pouvez retrouver le code du TER dans le lien suivant [Lien gitlab](#)

Le reste du rapport est organisé de la manière suivante, la première section concerne les données numériques, pour cela on a choisi le jeu de données iris, on travaillera sur trois techniques différentes de génération de données numérique, puis on évaluera la qualité des modèles en prédiction et en apprentissage.

Au cours de la deuxième section, nous verrons comment générer des données images de type MNIST (images de chiffres) à partir des GANs et évalueront la qualité du modèle.

Enfin, lors de la dernière partie, nous nous approfondirons sur les réseaux neuronaux, les LSTM pour générer des données textuelles à partir des dataset portant sur les opinions.

## Partie 2

# Données numériques

Dans cette partie, on s'intéresse à la génération des données numériques, pour cela nous allons mener différentes expérimentations avec un jeu de données traditionnel en apprentissage automatique qui s'appelle 'IRIS'. Ce jeu de données se compose de 50 échantillons de chacune des trois espèces des fleurs d'iris (Iris setosa, Iris virginica et Iris versicolor).

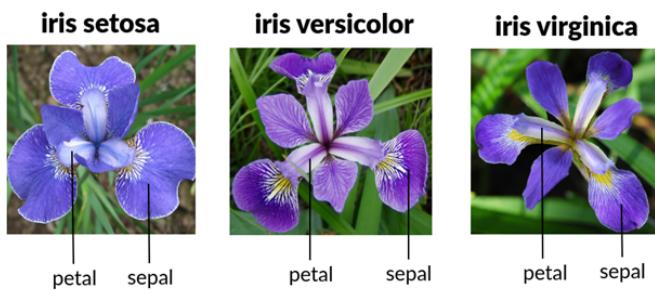


FIGURE 2.1 – Espèces de fleur d'iris (Source : medium.com)

Au cours de cette section, nous allons aborder différentes manières de générer des données numériques : à partir de la moyenne et écart type, à partir de SMOTE et de SDV. A la fin de chaque expérimentation, nous proposons des discussions en vue d'évaluer la qualité des données générées.

### 2.1 Classification des données réelles

Notre objectif dans cette section est d'apprendre un vrai classifieur sur des vraies données réelles, pour cela nous utilisons le jeu de données IRIS dont le descriptif est le suivant :

	Sepal length	Sepal width	Petal length	Petal width	Class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
:	:	:	:	:	:
150	5.9	3.0	5.1	1.8	virginica

FIGURE 2.2 – Quelques lignes du jeu de données Iris (Source : ResearchGate.net)

La figure 2.2 illustre le jeu de données iris où nous avons 4 caractéristiques : longueur pétales, largeur pétales, longueur sépales, largeur sépales.

A partir de ce jeu de données, nous avons chercher le meilleur classifier possible.

Afin d'évaluer la qualité de la performance de ces modèles, on a utilisé les différentes mesures classiques :

- **accuracy** : Elle permet de connaître le nombre de prédictions positifs bien effectuées. Plus elle est élevée, plus le modèle de Machine Learning minimise le nombre de Faux Positif.  
Quand la précision est haute, cela veut dire que la majorité des prédictions positives du modèle sont des positifs bien prédit.
- **recall** : Le recall permet de savoir le pourcentage de positifs bien prédit par notre modèle. En d'autres termes c'est le nombre de positifs bien prédit (Vrai Positif) divisé par l'ensemble des positifs (Vrai Positif + Faux Négatif).
- **f1 score** : Bien qu'ils soient utiles, ni l'accuracy ni le recall ne permettent d'évaluer entièrement un modèle de Machine Learning. Séparément c'est deux métrique sont inutiles :
  - si le modèle prédit tout le temps « positif », le recall sera élevé
  - au contraire, si le modèle ne prédit jamais « positif », la précision sera élevée

On aura donc des métriques qui nous indiquent que notre modèle est efficace alors qu'il sera au contraire plus naïf qu'intelligent.

Heureusement pour nous, une métrique permettant de combiner la précision et le recall existe : le F1 Score. Le F1 Score permet d'effectuer une bonne évaluation de la performance de notre modèle. Il se mesure ainsi :

Plus notre F1 Score est élevé, plus notre modèle est performant.

- **matrice de confusion** : La matrice de confusion est un outil qui permet de savoir à quel point le modèle est « confus », ou qu'il se trompe.  
C'est une matrice qui mesure la qualité d'un système de classification, il s'agit d'un tableau avec en colonne les différents cas réels et en ligne les différents cas d'usage prédits.

On peut observer ci-dessous, la matrice de confusion et les différentes mesures mentionnées auparavant pour le modèle initial du jeu de données iris :

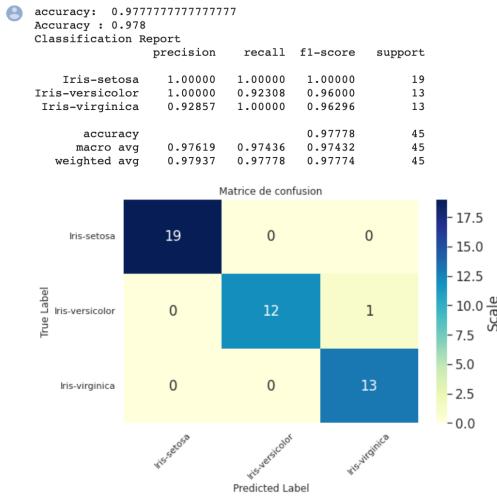


FIGURE 2.3 – Matrice de confusion du modèle initial

En effet, on a retenu le modèle avec la meilleure accuracy (0.98) qui nous permettra de bien prédire les données. C'est avec ce modèle là que nous allons nous comparer par la suite.

## 2.2 Génération de données à partir de moyenne et écart type

### 2.2.1 Évaluation de la prédiction

Dans la distribution des données réelles de la figure 2.4(a) ci-dessous, on remarque qu'il y a une distinction entre les trois classes : l'espèce Setosa en bleue est facilement différenciable et que Virginica (vert) et Versicolore (orange) sont assez proches donc plus difficilement classifiables. Pour se rapprocher de notre objectif d'évaluer la qualité des résultats, il serait plus intéressant de choisir l'une de ces 2 classes.

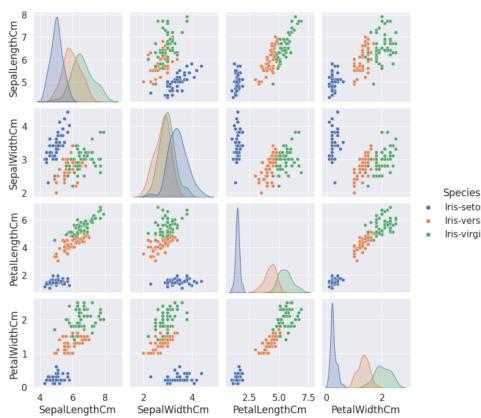
On s'est donc plutôt penché sur l'espèce Virginica, on a retiré 30 iris de cette classe de notre jeu de données initial, pour essayer par la suite, de les générer.

On a d'abord commencé par prédire les données à partir de la moyenne et l'écart-type appliquées sur nos données initiales pour avoir une certaine estimation.

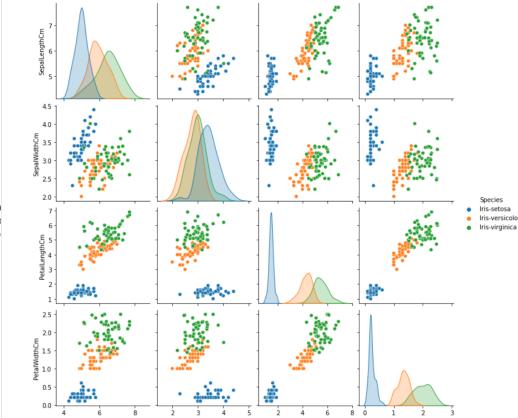
Après avoir remarqué que les colonnes suivent une loi normale, on a relevé les variables mean et standard variation correspondant respectivement à la moyenne et l'écart type.

A partir des intervalles calculés, on a pu générer des données comprises dans l'intervalle.

On peut observer la distribution dans le sns pairplot ci dessous à droite(b). On a 3 couleurs différentes pour chaque classe d'Iris, chaque graphique de la matrice (excepté les diagonales) présente les données selon deux critères (features). Les diagonales représentent la distribution marginale des vecteurs en fonction de chaque caractéristique.



(a) sns pairplot real data



(b) sns pairplot Moyenne et Écart-type

FIGURE 2.4 – Distribution de données réelles et générées par moyenne et écart-type

La courbe de la longueur des sépales des virginicas est légèrement plus basse dans les données générées que dans les vraies données. La courbe de la largeur des sépales générée est un peu plus à droite que la courbe initiale. Concernant les pétales, la courbe de la longueur des virginicas générées est un peu plus haute et la largeur plus basse que la courbe sur les données initiales.

## 2.2.2 Évaluation dans l'apprentissage

Pour évaluer la qualité du modèle dans l'apprentissage nous avons

accuracy: 0.9777777777777777

Accuracy : 0.978

Classification Report

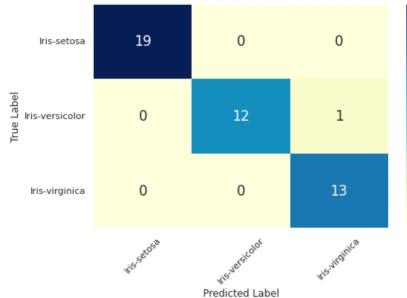
	precision	recall	f1-score	support
Iris-setosa	1.00000	1.00000	1.00000	19
Iris-versicolor	1.00000	0.92308	0.96000	13
Iris-virginica	0.92857	1.00000	0.96296	13
accuracy		0.97778	45	
macro avg	0.97619	0.97436	0.97432	45
weighted avg	0.97937	0.97778	0.97774	45

accuracy : 0.978

Classification Report

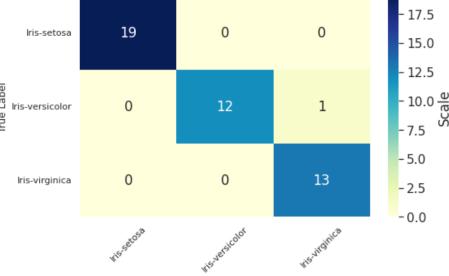
	precision	recall	f1-score	support
Iris-setosa	1.00000	1.00000	1.00000	19
Iris-versicolor	1.00000	0.92308	0.96000	13
Iris-virginica	0.92857	1.00000	0.96296	13
accuracy		0.97778	45	
macro avg	0.97619	0.97436	0.97432	45
weighted avg	0.97937	0.97778	0.97774	45

Matrice de confusion



(a) Matrice de confusion Real Data

Matrice de confusion



(b) Matrice de confusion Moyenne et écart type

FIGURE 2.5 – Matrices de confusion des données réelles et générées par moyenne et écart-type

Les matrices de confusion sont très similaires, et l'accuracy est légèrement différente.

## 2.3 Utilisation de SMOTE

SMOTE, Synthetic Minority Oversampling Technique, est une méthode de suréchantillonnage des observations minoritaires. Pour éviter de réaliser un simple clonage des individus minoritaires, SMOTE se base sur un principe simple : générer de nouveaux individus minoritaires qui ressemblent aux autres. Cela permet de densifier de façon plus homogène la population d'individus minoritaires.

Pour en savoir plus sur cette technique vous pouvez vous référer au lien suivant : [SMOTE](#)

### 2.3.1 Évaluation de la prédiction

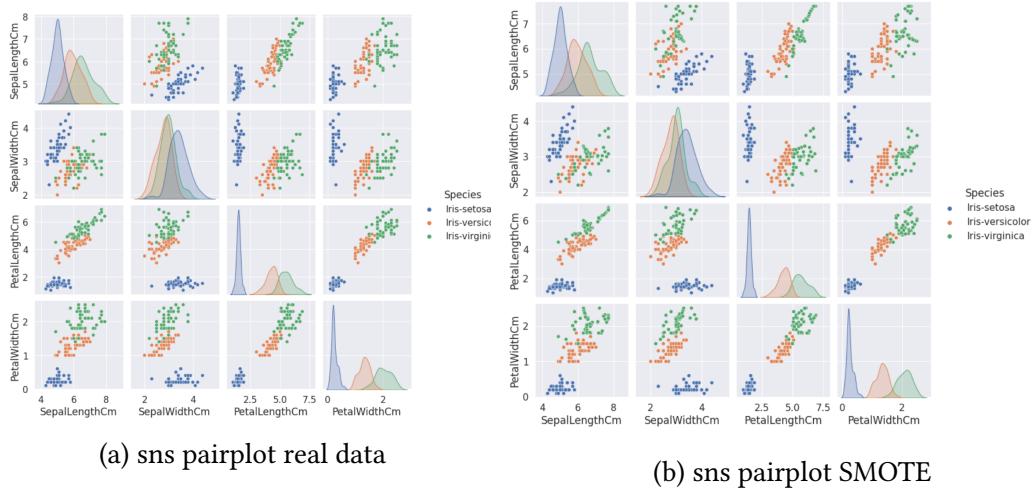
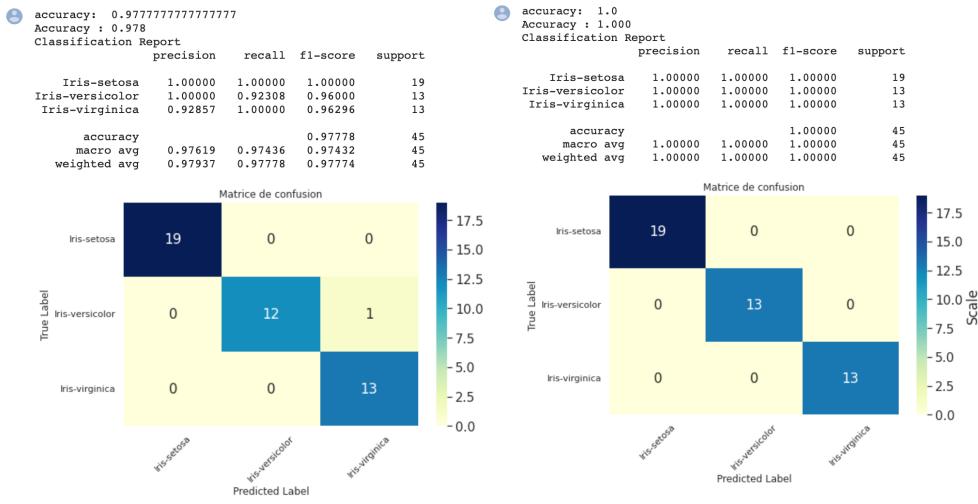


FIGURE 2.6 – Distribution de données réelles et générées par SMOTE

On remarque que la distribution est presque similaire, la largeur des sépales et la longueur des sépales des virginica sont légèrement plus grandes dans le SMOTE. On remarque aussi que la pente de la longueur des sépales forme un angle droit, elle ne descend pas en ligne droite comme dans l'original. La courbe de la longueur des pétales est plus étendue et celle de la largeur des pétales est plus haute dans le SMOTE que dans les données initiales.

### 2.3.2 Évaluation dans l'apprentissage



(a) Matrice de confusion Real Data

(b) Matrice de confusion SMOTE

FIGURE 2.7 – Matrices de confusion des données réelles et générées par SMOTE

La matrice de confusion est différente d'un grain, on peut dire que c'est une bonne prédiction.

## 2.4 Utilisation de SDV

SDV (Synthetic Data Vault) est un package Python permettant de générer des données synthétiques basées sur l'ensemble de données fourni. En outre, les données générées auraient les mêmes propriétés de format et les mêmes statistiques que le jeu de données fourni. SDV génère des données synthétiques en appliquant des techniques mathématiques et des modèles d'apprentissage automatique tels que le modèle de deep learning. Même si les données contiennent plusieurs types de données et des données manquantes, SDV s'en chargera, nous n'avons donc qu'à fournir les données.

Pour en savoir plus sur cette technique vous pouvez vous référer au lien suivant : [SDV](#)

### 2.4.1 Évaluation de la prédiction

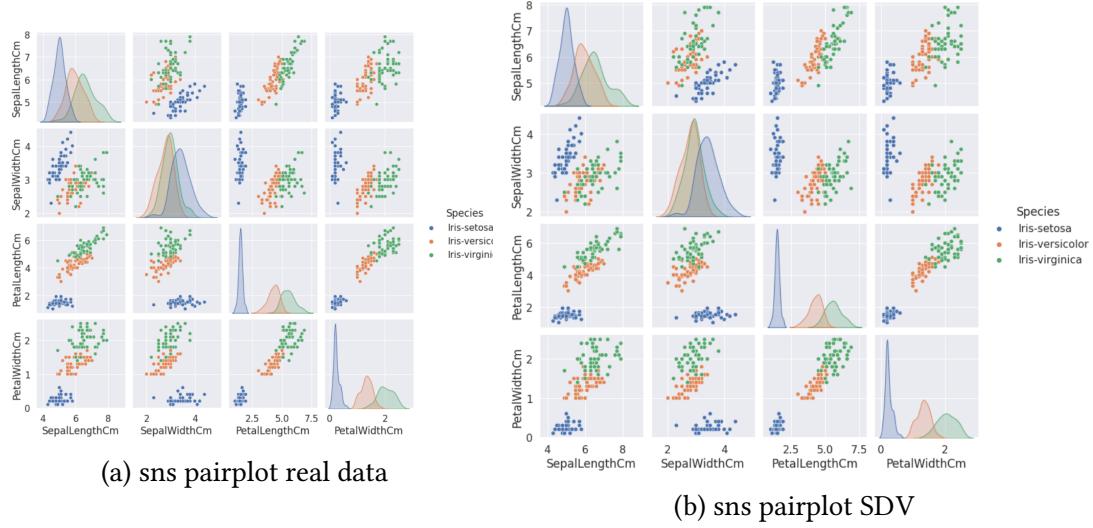


FIGURE 2.8 – Distribution de données réelles et générées par SDV

### 2.4.2 Évaluation dans l'apprentissage

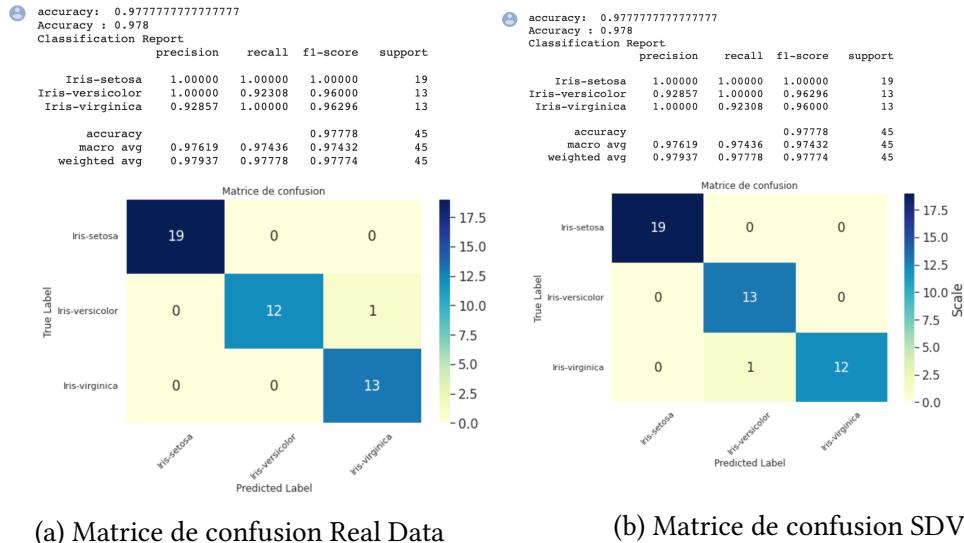


FIGURE 2.9 – Matrices de confusion des données réelles et générées par SDV

On observe que la pente de la courbe de la longueur des sépales n'est pas très droite comparée à celle des données initiales, la largeur des sépales est assez similaire. La courbe de la longueur des pétales est un peu plus pointu et la courbe de la largeur des sépales est plus arrondie que les données initiales.

## 2.5 Discussion

Notre objectif était de voir si nous pouvions générer des données numériques et nous avons pu constater au cours de ces expérimentations avec les trois méthodes différentes que nous étions capables, effectivement, de générer des données numériques assez similaires aux données réelles. En effet, dans l'évaluation en prédiction et l'évaluation en apprentissage les scores entre données réelles et générées sont assez similaires donc ça montre qu'on a pu remplir notre objectif.

Par contre, en raison des valeurs rapprochées entre les scores des différentes approches, il est très difficile de déterminer laquelle des méthodes est meilleure ou moins bien que l'autre. Cela est dû au manque de variation des données et ainsi, si on veut et cibler la meilleure méthode de génération de données numériques on pourra réaliser des expérimentations sur des données plus variées sur la chaîne de traitement mise en place.

Notre sujet principal se porte sur la génération de différents types de données et pour avancer sur celui ci et réaliser nos objectifs, nous avons mis en place une chaîne de traitement pour les données numériques qui est adaptée aux différents jeux de données et qui permettra alors aux personnes intéressées de s'approfondir sur le sujet.

# Partie 3

## Données Images

Dans cette partie on va s'intéresser à des données de type image pour cela nous allons utiliser des réseaux de convolutions mais également des réseaux antagonistes génératifs. Par la suite nous allons effectuer différentes expérimentations menées sur jeux de données MNIST et nous analyserons les résultats obtenus.

### 3.1 Rappel sur les CNN et les réseaux antagonistes génératifs

#### 3.1.1 Architectures classiques des réseaux de neurones

Un réseau de neurones est constitué d'une couche d'entrée, de couches cachées et d'une couche de sortie. Il y a plusieurs choix d'architecture à faire, notamment le nombre de couches (profondeur du réseau), le nombre de neurones par couche (largeur du réseau) et le type de connexion entre les couches.

Une fois l'architecture définie, il faut choisir une fonction de coût, une fonction de sortie et les fonctions d'activation des couches cachées.

**Apprentissage** L'apprentissage des paramètres d'un réseau de neurones se fait en 2 temps. Une première phase appelée **forward pass** où les données en entrée sont transmises dans le réseau. L'erreur est donc calculée avec la fonction de coût. Les paramètres du réseau sont ensuite mis à jour de façon à minimiser l'erreur. Il s'agit de la **backward pass** où l'erreur est minimisée par rétropagation du gradient de la fonction de coût.

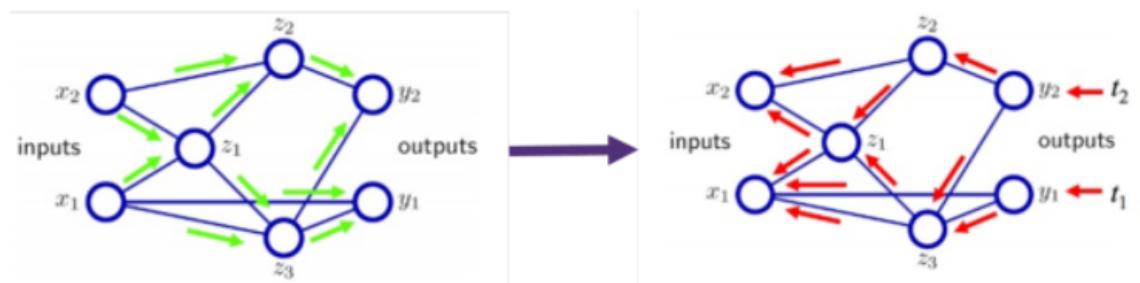


FIGURE 3.1 – Forward pass backward pass (Source : Researchgate.net)

### 3.1.2 Réseau neuronal convolutif CNN

Un réseau de neurones convolutifs est un type de réseau de neurones acycliques dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les réseaux convolutifs sont particulièrement utilisés sur des données ayant une structure de 'grille' comme les images ou les séries temporelles. Le nom convolutif indique que le réseau utilise l'opération de convolution qui est un cas particulier d'opération linéaire. L'opération de convolution peut être définie comme suit :  $s(t) = (x * t) = \int x(a)w(t - a)da$ .

On travaille souvent avec des convolutions discrètes qu'on peut définir comme suit :  $s(t) = (x * t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a)$ .

Les convolutions sont choisies pour ce type d'applications car elles présentent certaines caractéristiques :

- **Interactions sparses** A l'instar des réseaux de neurones classiques où chaque unité de sortie interagit avec toutes les unités d'entrée, les réseaux de neurones convolutifs peuvent dégager des variables importantes sans faire interagir toutes les unités d'entrée. Par exemple, en traitement d'image, on est capable de détecter des contours sur des images ayant plusieurs milliers ou millions de pixels avec des noyaux qui n'occupent que quelques dizaines de paramètres. Ceci permet de réduire le besoin de stockage en mémoire de paramètres et aussi d'améliorer les performances.
- **Partage de paramètres** On n'a pas besoin d'apprendre un set de paramètres par position, mais uniquement un seul set de paramètres. Ceci fait des convolutions un outil bien plus efficace que la simple multiplication matricielle.

En plus des couches de convolution, on rajoute des couches de pooling qui consiste à remplacer la sortie à l'issue de la phase de convolution et d'activation d'une certaine position par une statistique sommaire de la zone environnante. Par exemple, le max-pooling remplace les sorties par le maximum sur un rectangle avoisinant. Ceci permet de rendre la représentation des données invariante aux petites translations. Par exemple, pour détecter des sourcils sur un visage, on n'a pas besoin de connaître exactement leur emplacement.

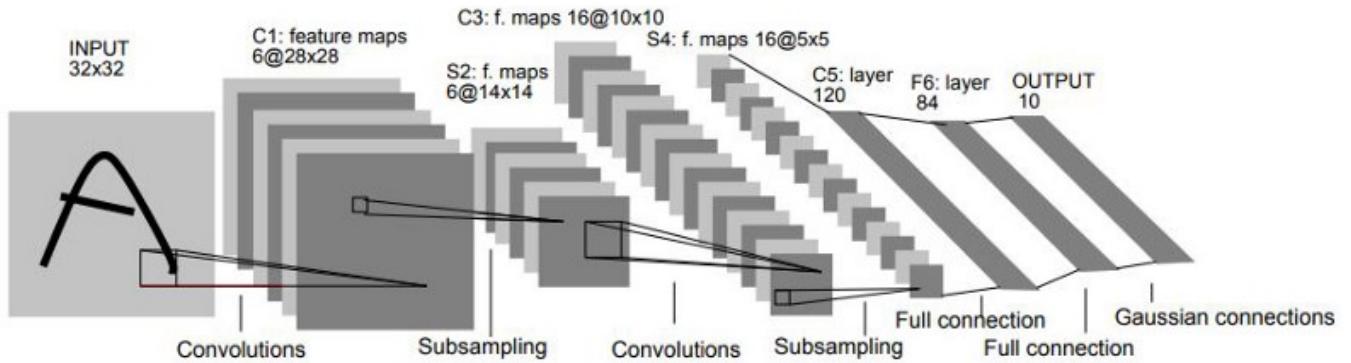


FIGURE 3.2 – Architecture classique d'un réseau convolutif (Source : Researchgate.net)

### 3.1.3 Réseau Antagoniste Génératif GAN

Un GAN ou Generative Adversarial Network (réseau antagoniste génératif) est une technique d'intelligence artificielle permettant de créer des imitations parfaites d'images ou autres données.

Elle repose sur la mise en compétition de deux réseaux : "générateur" et "discriminateur". Le générateur est un type de réseau neuronal convolutif dont le rôle est de créer de nouvelles données sans que l'on puisse déterminer leur véracité.

De son côté, le discriminateur est un réseau neuronal « déconvolutif » qui détermine l'authenticité des données ou s'il fait ou non partie d'un ensemble de données.

Ainsi, le générateur produit des données de meilleure qualité tandis que le discriminateur détecte de mieux en mieux les fausses.

La figure 3.1 schématisé le fonctionnement d'un GAN.

On donne en entrée au réseau génératif un bruit de grande dimension afin de produire des objets. Ces objets générés sont ensuite transmis au discriminateur, aux côtés des véritables données du jeu. Le discriminateur détermine l'authenticité de la sortie, on peut alors appliquer la rétro-propagation.

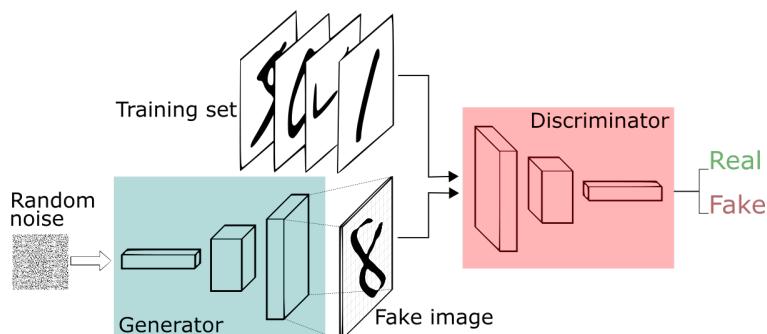


FIGURE 3.3 – Figure 3.1 - Schéma représentatif des GANs (Source Wiki)

## 3.2 MNIST : Ensemble de données de chiffres manuscrits

L'ensemble de données MNIST est un acronyme qui signifie l'ensemble de données modifié de l'Institut national des normes et de la technologie.

Il s'agit d'un ensemble de données de 70 000 petites images carrées en niveaux de gris de  $28 \times 28$  pixels de chiffres simples écrits à la main entre 0 et 9.

La première étape consiste à définir le modèle discriminateur. Il s'agit d'une classification binaire avec 2 couches convolutionnelles, il contient des couches de convolution 2D *conv2D*, des couches intermédiaires avec activation *LeakyReLu*.

Une couche *Dropout* pour éteindre une fraction des neurones au hasard afin d'éviter le sur-apprentissage.

La couche *Flatten* pour recaler la sortie du dropout au bon format pour la couche suivante et une couche de sortie *Dense* avec activation *sigmoid* pour s'assurer que les valeurs de sortie sont dans la plage souhaitée de [0,1].

Le modèle est mis à jour par lots *batches*, notamment avec une collection d'échantillons réels et une collection d'échantillons générés. Lors de la formation, l'*epoch* est définie comme un passage à travers l'ensemble de données de formation.

Le modèle générateur est chargé de créer de nouvelles images fausses mais plausibles de chiffres manuscrits.

Pour cela, il prend un point de l'espace latent en entrée et produit une image carrée en niveaux de gris.

On a une couche *dense* en tant que première couche cachée (hidden layer) qui a suffisamment de nœuds pour représenter une version basse résolution de l'image de sortie.

La couche *Conv2DTranspose* sert pour le processus de suréchantillonnage (upsampling) de l'image basse résolution vers une version haute résolution de l'image, appelé aussi déconvolution.

Encore une fois, on a la couche intermédiaire *LeakyReLu* et la couche de sortie *conv2D* avec activation *sigmoid*.

Le modèle générateur doit être entraîné, ses pondérations sont mises à jour en fonction des performances du modèle discriminateur. On a alors un nouveau modèle composite qui combine le générateur et le discriminateur afin d'entraîner le générateur.

Dans la figure 3.2 on a le résultat du modèle sauvegardé après 100 époques :



FIGURE 3.4 – 100 images MNIST générées par GAN après 100 époques

### 3.3 Discussion

Le modèle de générateur final peut être utilisé de manière autonome, on charge le modèle à partir du fichier créé puis on génère 25 images afin de les comparer avec les images du vrai jeu de données.

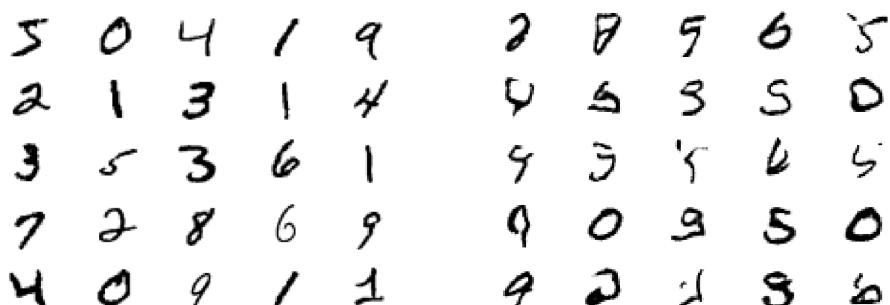


FIGURE 3.5 – Comparaison du jeu de données MNIST avec les données images générées

On remarque que les images produites sont assez similaires avec les images réelles, on peut observer par exemple les images du 5, 0 et 9 sont reproduites très clairement.

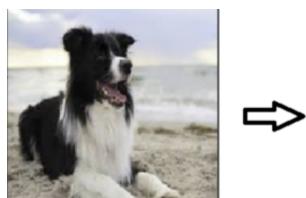
Cependant, on observe des images qui ne correspondent pas vraiment aux chiffres de notre base de données initiale mais sont peu fréquentes.

Par exemple l'image qui se trouve au milieu de la Figure b ressemble à un 5 "avec une apostrophe", ou bien l'image à droite de cette dernière qui ressemble à la lettre "l".

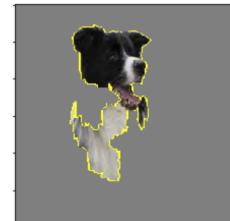
Pourtant lors de la prédiction, l'image est classée dans la bonne classe, il sera donc intéressant d'analyser quels sont les éléments qui ont été retenus dans l'image pour pouvoir classer dans la bonne classe et pour ça nous pouvons utiliser LIME (Local Interpretable Model-agnostic Explanations) [LIME](#) dont l'objectif est de mettre en évidence les patterns qui lui ont permis de classer.

Le concept du LIME repose sur la segmentation de l'image initiale du dataset en plusieurs sous images, tel que chaque segment sera traité comme une caractéristique et chaque caractéristique représente la zone segmentée correspondante.

Les contributions des segments de l'image sont en fait les coefficients de caractéristique de la régression linéaire de substitution, dans l'exemple de [l'article](#), l'image du Border Collie, on repère 5 caractéristiques pour prédire l'animal, en d'autres termes, notre modèle dépend fortement des 5 zones segmentées qui reconnaissent le Border Collie dans l'image. Voici quelques images qui montrent clairement les étapes de la prédiction de l'animal Border Collie :



(a) Segmentation de l'image



(b) Caractéristique principale de l'image

FIGURE 3.6 – Fonctionnement du LIME (Source : medium.com)

On peut alors imaginer que c'est exactement le même procédé dans notre jeu de données MNIST : il existe certaines caractéristiques qui permettent de prédire l'image du chiffre 5 par exemple, et donc malgré la bonne prédiction de cette image, on pourra retrouver une divergence visuelle.

En conclusion, le résultat est plutôt bon car l'entraînement a bien été fait sur un ensemble de pixels de chaque image, ce qui rend les images générées qui sont différentes du dataset initial acceptables.

# Partie 4

## Données textuelles

La génération de texte est l'une des grandes tendances actuelles du machine learning. Elle peut être considérée comme une caractéristique très importante des outils basés sur l'IA. Dans ce scénario, les systèmes d'IA apprennent en absorbant des milliards de mots extraits d'internet et génèrent du texte en réponse à diverses requêtes. Ces IA peuvent ensuite être utilisées pour un large éventail de tâches, résumé un texte, Chatbots, question répondant, les moteurs de recherche, en passant par la possibilité de discuter avec des personnages historiques.

Dans cette partie, on s'intéresse à créer un modèle de génération de texte en se basant sur l'architecture des réseaux LSTM. Les motivations sont multiples comme la génération de faux avis par exemple. L'objectif est double :

- Générer du texte de bonne qualité qui soit compréhensible et éligible pour un humain.
- Générer du texte qui exprime une opinion (positive ou négative).

Pour le premier cas, la meilleure approche reste encore de faire évaluer le texte par un humain. En ce qui concerne l'opinion, on pourra comme dans la partie précédente utiliser un classifieur de texte afin de déterminer l'opinion des textes que nous avons générés.

Nous verrons par la suite que même si les phrases ne sont pas toujours bien construites grammaticalement parlant, elles pourront être classées par le classifieur et donc servir à étendre un jeu de donnée ou même à entraîner un autre classifieur.

### 4.1 Rappel sur les réseaux neuronaux récurrents et les réseaux LSTM

#### 4.1.1 Recurrent neural network RNN

On a vu dans la partie précédente les différents types de neurones, dans cette partie on s'intéresse à des réseaux qui sont un peu différents : les réseaux récurrents.

Un réseau de neurones récurrents est un réseau de neurones présentant des connexions récurrentes. Il est constitué d'unités interconnectées interagissant non-linéairement et pour

lequel il existe au moins un cycle dans la structure. Ce type de réseau de neurones est adapté pour le traitement des données séquentielles. Ils sont aussi adaptés pour des données d'entrée de taille variable.

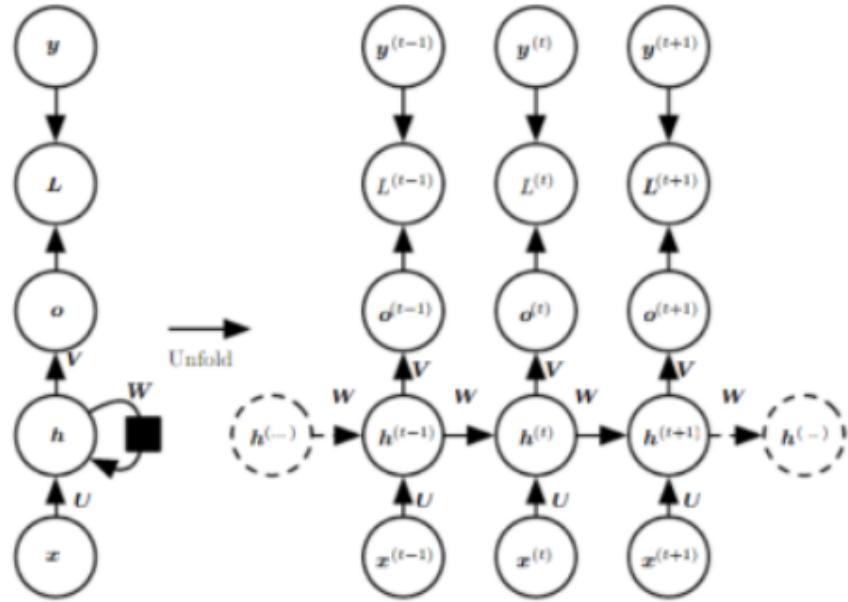


FIGURE 4.1 – Réseau de neurones récurrent sous format compact et "déplié" (Source : datascientiststoday.net)

#### Vanishing gradient

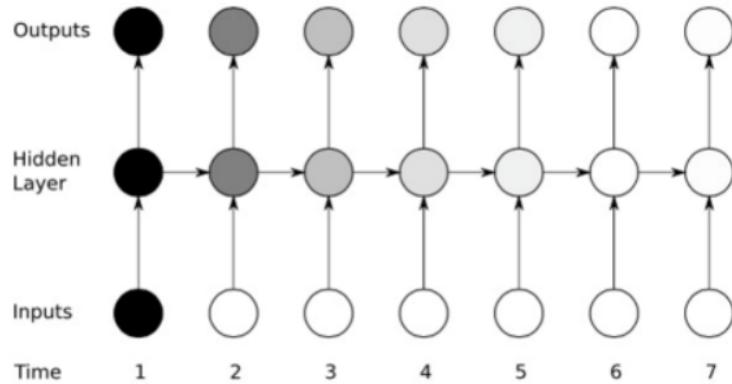


FIGURE 4.2 – Illustration du Vanishing gradient (Source : researchgate.net)

On peut démontrer mathématiquement que la composition successive de fonctions au sein d'un réseau de neurones récurrents fait tendre le gradient vers 0. Ceci rend le réseau incapable de modéliser des dépendances à long terme.

### 4.1.2 Les réseaux LSTM

On peut détourner ce problème en remplaçant les unités du réseau (neurones) par des cellules **LSTM** : **Long Short Terme Memory**. Il s'agit d'une cellule mémoire read-write qui **apprend** à mémoriser et à oublier. A chaque itération, la cellule met à jour sa **mémoire** et son **état**. On peut observer l'architecture de la cellule LSTM plus en détail dans la figure suivante :

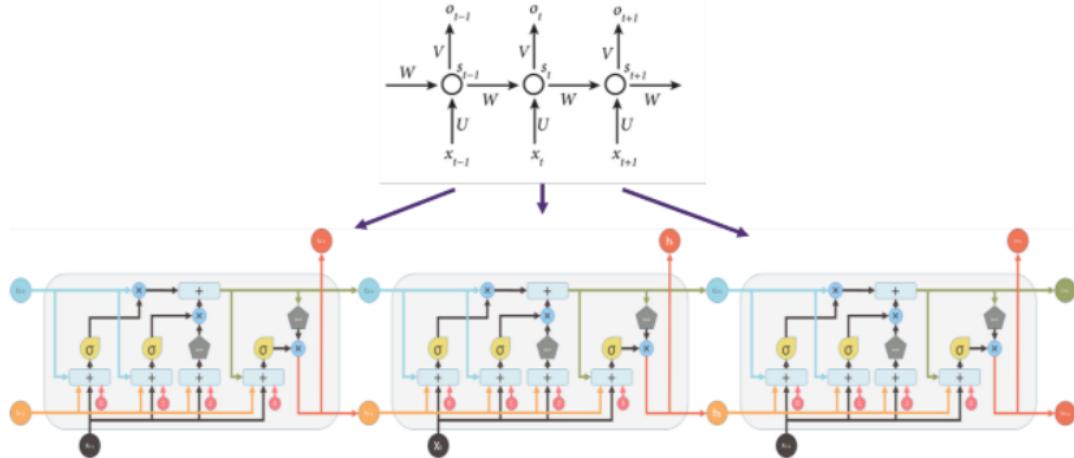


FIGURE 4.3 – Long Short Term Memory (Source : medium.com)

Une cellule LSTM est obtenue en combinant les composantes suivantes :

- **Pipeline de mémoire** : Partie de la cellule responsable de la mise à jour de l'état
- **Forget gate** : Partie de la cellule responsable de la prise en compte de l'état précédent
- **Memory gate** : Partie de la cellule responsable de la mise à jour de la mémoire
- **Output gate** : Partie de la cellule responsable du calcul de la sortie

La figure suivante montre les différentes composantes citées :

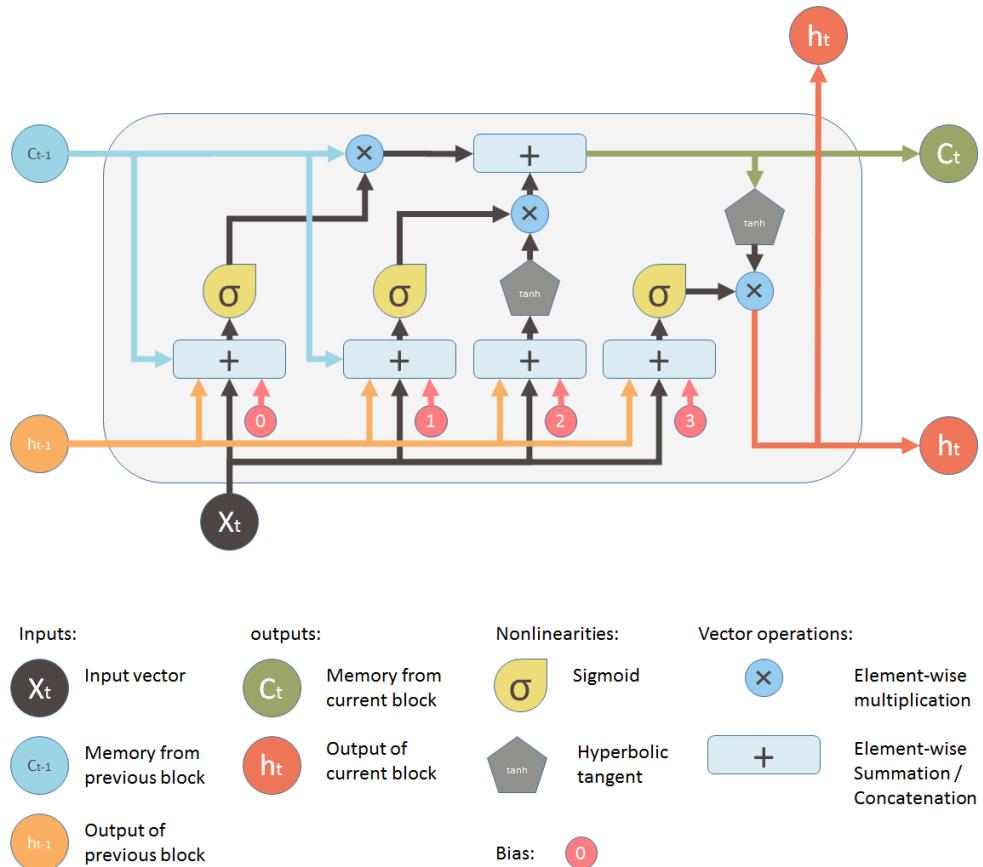


FIGURE 4.4 – Cellule LSTM (Source : medium.com)

## 4.2 Classification des données réelles

Dans cette partie, on s'intéresse à générer des avis positifs à partir d'un jeu de données en se basant sur le modèle LSTM.

Pour cela, les données ont été téléchargées à partir de Kaggle, on a choisi comme jeu de données [IMDB Dataset](#), contenant un total de 50 000 avis répartis en deux classes : **positive** et **négative**.

Après avoir mis en place le modèle initiale à partir du dataset IMDB, nous avons choisi le meilleur classifieur pour évaluer notre modèle. On utilise les mêmes mesures mentionnées auparavant. On retrouve dans la figure suivante la matrice de confusion du modèle initiale dont on se servira pour comparer les modèles générées par la suite.

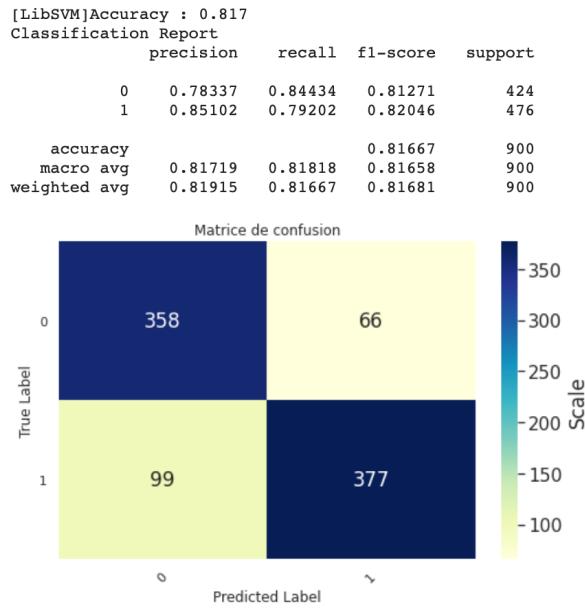


FIGURE 4.5 – Matrice de confusion du modèle de la base de données IMDB

## 4.3 Génération des données textuelles avec LSTM

### Génération des données textuelles caractère par caractère

Dans un premier temps et avant d'aborder notre vrai problème, nous avons mené quelques expérimentations. Nous nous sommes intéressées à la tâche de génération de texte au niveau des caractères, basée le livre [Alice's Adventures in Wonderland](#) comme un dataset d'entrée. Par la suite, on s'intéressera à mettre en place un modèle plus large dont le jeu de données contient plusieurs livres à la fois.

Nous traitons notre texte comme une séquence de caractères, ainsi notre modèle prédit le prochain caractère comme étant le plus probable en fonction des caractères qu'il a rencontré précédemment. Nous demanderons ensuite au modèle de générer de nouvelles séquences "originales" basées sur ce qu'il a appris.

À la fin de notre génération, on obtient la figure 4.5 comme résultat.

```
so long i got this one a few weeks ago and love
I got this one a few weeks ago and love it
good job name review length 25000 dtype object object 49992 john
a few weeks ago and love it it 49992 john garfield
this movie did a down right good job name review length
```

FIGURE 4.6 – Exemple de phrases générées lettre par lettre

## Génération des données textuelles mot par mot

Dans cette partie, on s'est attaqué aux données d'apprentissage basées sur le jeu de données d'opinions. Nous pourrons ainsi tester la capacité du système à recréer des mots, puis à les assembler dans des structures grammaticales cohérentes.

Notre modèle d'apprentissage prédira la probabilité de chaque mot étant donné une séquence de texte entrée. Le mot prédit sera donné en entrée pour générer à son tour le mot suivant. Plus précisément, nous utiliserons des embeddings pour apprendre la représentation des mots et un réseau **LSTM** pour apprendre à prédire les mots en fonction de leur contexte. Ces embeddings ont été entraînés au préalable sur une quantité importante de données.

On obtient les résultats illustrés dans la figure 4.6 :

```
besides the leads both martine beswick and rg armstrong briefly
I loved back to a simple facial and ended a sassy
This is dated a conseguenze dellamore who made the atomic year
a exwife and inhabit a land of hitchcock louise previously whilst
Good but per second gratuitous henry and the ugly sea of
So long more baffling is the movie is excellent or decent who
```

FIGURE 4.7 – Exemple de phrases générées mot par mot

### 4.3.1 Évaluation du modèle

Pour évaluer notre modèle LSTM du mot par mot, on affiche la matrice de confusion et les mesures (accuracy, f1-measure, recall) suivantes :

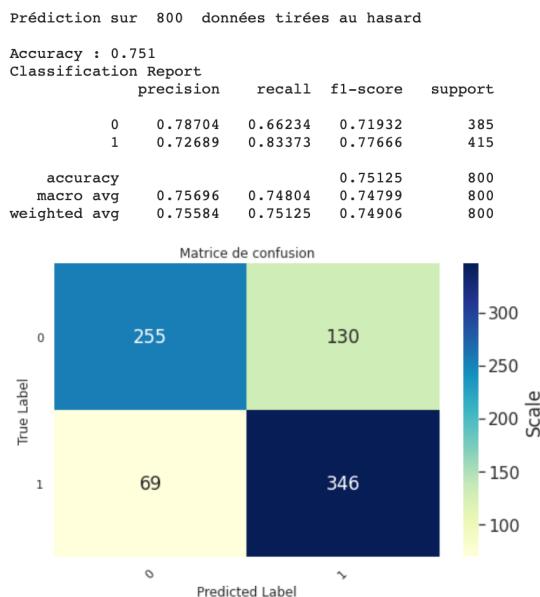


FIGURE 4.8 – Matrice de confusion du modèle généré par LSTM

On remarque que l'accuracy du modèle est légèrement différente du modèle initiale (de 0.06), on retrouve également peu d'erreurs dans la matrice de confusion.

On peut déduire qu'il s'agit d'un assez bon modèle malgré l'incohérence des phrases générées.

### 4.3.2 Discussion

Parmi les phrases générées, il y en a qui n'ont pas vraiment de sens voire incohérentes mais aussi on retrouve des phrases plus ou moins correctes, c'est à dire qu'on arrive à comprendre qu'il s'agit d'un avis sur un film sans pour autant être correctes du point de vue grammaticale.

Bien que les mots soient tous bien formés de part la présence des embeddings, le sens global des phrases est difficile à saisir. En particulier, les phrases n'ont pas vraiment de début ni de fin. On remarque également la présence de "boucles" avec une séquence de mots qui se répètent. Autrement la qualité semble augmenter avec la taille du jeu de données. En effet, les phrases générées à partir du jeu de donnée IMDB dataset ont l'air plus réalistes.

Malgré que le modèle soit bon, les phrases sont incohérentes, ce qui nous fait rappeler le même problème rencontré dans les données images. C'est à dire que là aussi il existe une explication à ceci comme on a trouvé le LIME pour les GANs.

# Partie 5

## Gestion de projet

### 5.1 Cahier des charges

Concernant l'organisation, une réunion est prévue chaque deux semaines avec l'encadrant pour assurer un suivi régulier tout au long du semestre. De chaque réunion doit résulter un compte rendu reprenant les points essentiels de la séance ainsi que les objectifs de la semaine. Enfin, vous trouverez ci-dessous le planning détaillant la répartition du travail sous la forme d'un diagramme de GANTT

	Réunion 1,2	Semaine 3,4	Semaine 4,5	Semaine 6	Semaine 7	Semaine 8	Semaine 9	Semaine 10
-Travailler sur les différentes approches pour les IRIS (manuellement avec moyenne et écart-type, SMOTE et SDV)								
-Compréhension du code GAN (générateur, discriminateur) -Compréhension du fonctionnement du jeu de données MNIST								
-Mise en place des GANs pour générer des images MNIST								
-Travailler sur les données textes "Alice in Wonderland" et utiliser les LSTM.								
-Mise en place des réseaux neuronaux LSTM pour générer les données textuelles								
-Corrections apportées sur le rapport								

FIGURE 5.1 – Diagramme de Gantt

### 5.2 Outils utilisés

Afin de mener à bien notre projet, nous avons utilisé de nombreux outils pour arriver au résultat final. Un maître mot : la collaboration. En effet, certains de ces outils permettent à plusieurs membres d'un même groupe de projet de travailler sur le même fichier et de faciliter le partage de celui-ci. De plus, avec le contexte sanitaire actuel, ces outils sont indispensables. En voici la liste :

- **Github** : service web d'hébergement et de gestion de développement de logiciels. C'est une plateforme à la fois open source et collaborative. Cela nous permet d'avoir une bonne vision du projet et également d'avoir une sauvegarde dans le cloud, très important en cas de perte de données. Chaque membre du groupe a ainsi accès à tous les fichiers du projet et est libre de faire des modifications. De plus, cet outil est très pratique pour la gestion des versions du code, nous permettant de voir l'évolution de celui-ci.
- **Discord** : outil de communication et de gestion en ligne. Très pratique pour communiquer avec notre encadrant à propos de l'avancée du projet et les tâches à effectuer.
- **Google Drive** : service de stockage et de partage de fichiers dans le cloud permettant également de modifier des documents, des feuilles de calcul, des présentations, des dessins, des formulaires, etc... tout cela directement en ligne. C'est sur ce Drive que nous stockons les comptes rendus et bilans de semaine de chaque réunion, nos recherches sur les composantes de l'application, des graphiques de scénarios, etc.
- **Zoom** : on ne le présente plus. Il a été notre meilleur ami durant ces deux dernières années scolaires du fait du contexte sanitaire. Zoom est un logiciel permettant de faire des visioconférences. Nous l'avons utilisé pour nos réunions hebdomadaires, ne pouvant pas nous rencontrer en personne.
- **Overleaf** : éditeur Latex en ligne collaboratif. Latex est un langage et système de composition de document. Ainsi, Overleaf nous permet de créer des documents PDF efficacement, avec les bonnes normes de typographie, ainsi que des diaporamas. De plus, il y possibilité pour chacun des membres du groupe d'éditer et de créer en temps réel des documents. C'est avec cet outil que nous avons réalisé ce rapport.
- **Google Colab** : Jupyter Notebook est l'outil incontournable du data scientist. C'est une application Web Open Source permettant de créer et de partager des documents contenant du code (exécutable directement dans le document), des équations, des images et du texte.  
Google Colab ou Colaboratory est un service cloud, offert par Google (gratuit), basé sur Jupyter Notebook. Cette plateforme permet de tester des bouts de code sans avoir besoin d'installer quoi que ce soit sur notre ordinateur à l'exception d'un navigateur.

# Partie 6

## Conclusion

### 6.1 Pour aller plus loin

Notre rapport de projet touche à sa fin et nous pouvons ainsi nous questionner sur l'accomplissement de notre cahier des charges. Dans l'ensemble, celui-ci a été respecté. Nous avons vu en introduction qu'il était important d'avoir des classes équilibrées, l'une des solutions était de générer des données.

En effet, dans ce projet nous avons abordé trois types de données :

- **DONNÉES NUMÉRIQUES**
- **DONNÉES IMAGES**
- **DONNÉES TEXTE**

Ainsi nous avons montré qu'il est était totalement possible de générer des données de "bonne" qualité comme le montrent les différentes expérimentations menées.

Outre, il y a des autres grandes améliorations qui nous semblent importantes.

Pour les données numériques, on peut essayer de mener de plus grandes expérimentations. On peut également s'intéresser à d'autres types d'images, à des images de plus grande dimension que celles utilisées dans nos expérimentations ( 28x28 pixels). Par contre, évidemment, le temps associé à la génération sera considérablement plus long.

Une piste d'amélioration pour les données textuelles serait de tester des autres architectures. On peut tester le modèle **sentiGAN**, constitué de plusieurs générateurs et un unique discriminateur multi-classes, dont le but est de générer des textes d'étiquettes de sentiments différents sans supervision.

### 6.2 Notre expérience avec ce TER

Ces 6 mois de travail se sont dans l'ensemble très bien déroulés pour notre groupe. En effet l'entente entre les membres était très bonne et la communication était très bien établie, que cela soit dans le groupe même ou avec l'encadrant. Les objectifs de ce TER étaient toujours très bien définis et à l'issue de chaque réunion. L'entraide et la cohésion sont très importantes dans un travail collaboratif et celles-ci ont été présentes tout au long du projet.

Concernant le projet en lui-même, il nous a permis de toucher à plusieurs domaines du Machine Learning et d'approfondir nos connaissances. On a pu se familiariser avec le Deep Learning en enchaînant plusieurs formations en ligne et tutoriels. A l'issue de cette étape, on a pu réaliser plusieurs preuves-de-concept (Classification d'image, génération de séquences de texte ...) ainsi qu'acquérir de nouvelles connaissances autour de ces algorithmes qui investissent de plus en plus notre quotidien.