

47Notre Datasetfigure.caption.5

59figure.caption.6

610formules de calcul des distancesfigure.caption.7

912création layersfigure.caption.10

1013version classification binairefigure.caption.11



NOTRE GROUPE PRÉSENTE :

HAI606I
PROJET DE PROGRAMMATION 2
RAPPORT

Classification de données textuelles...

Membres du groupe :

Rym ZEGGAR
Myriam WARD
Kenza LAHBABI
Francisco CARDENAS

Numéros d'étudiant :

21909615
21914737
21912897
21903458

Encadrant :

M. Pascal PONCELET

28 avril 2022

Table des matières

1	Introduction	1
2	Les fondations de l'apprentissage Automatique <i>Machine Learning</i>	2
2.1	Qu'est ce que le Machine Learning?	2
2.2	Les méthodes d'apprentissage	2
2.3	L'Apprentissage Supervisé	2
2.4	Les notions clés du Machine Learning	3
2.4.1	Le Dataset	3
2.4.2	Le modèle	3
2.4.3	La fonction coût <i>Cost Function</i>	4
2.4.4	L'algorithme d'apprentissage	4
2.5	Les applications les plus courantes du Machine Learning	4
2.5.1	Problème de Régression	4
2.5.2	Problème de Classification	4
3	Les fondations de l'apprentissage Profond (<i>Deep Learning</i>)	5
3.1	Qu'est ce que le Deep Learning	5
3.2	Le fonctionnement des réseaux de neurones artificielles	5
3.3	Exemples d'usage du Deep Learning	6
3.4	Entraînement d'un Réseau de Neurone	6
4	Approches pour une tâche	6
4.1	Présentation de notre DataSet	6
4.2	Traitement automatique du langage naturel <i>NLP</i>	7
4.2.1	Etape 1 : Nettoyer le jeu de données	7
4.2.2	Etape 2 :	10
4.3	Classification binaire selon l'opinion	10
4.4	Classification MultiClasse selon la Source	13
5	Classification MultiTaches	14
5.1	Etapes Générales de la classification	14
5.2	Classification Multitâche sur les Images	19
5.3	Classification Multitâche sur notre DataSet	20
6	Organisation du travail	20
6.1	Organisation prévisionnelle	20
6.2	Outils Utilisés	20
6.2.1	Pytorch	20
6.2.2	Python	20
6.2.3	Git	20
6.2.4	Scikit learn	21
6.2.5	Google Drive	21
6.2.6	Collaboratory	21
6.2.7	Messenger	21
6.2.8	Zoom	21
6.2.9	Overleaf	21

6.2.10 Miro	21
6.3 Répartition des tâches	22
7 Conclusion	22



Remerciements

En premier lieu, nous tenons à remercier notre encadrant, Mr Pascal PONCELET, pour son accompagnement, ses conseils et pour le temps qu'il nous a consacré, et qui nous a permis de réaliser ce projet.

1 Introduction

L'apprentissage automatique plus communément appelé *Machine Learning* constitue une sous-catégorie de l'intelligence artificielle. Il s'agit d'une technologie qui permet aux ordinateurs d'apprendre sans avoir été programmés spécifiquement à cet effet.

En d'autres termes, le machine learning est un concept qui tend à rendre la machine capable d'apprendre de ses propres expériences en lui fournissant des exemples.

Celle-ci repose sur la création d'algorithmes en mesure d'identifier des motifs récurrents dans des ensembles de données, quelle que soit la nature de ces dernières. Une fois ces patterns décelés au sein des données, ces algorithmes apprennent et sont aptes à améliorer leurs performances dans le cadre de l'exécution d'une tâche spécifique.

De la découverte de patterns découle une possibilité d'effectuer des prédictions. Contrairement à un programme informatique qui suivrait des instructions, un système de Machine Learning apprend par le biais de l'expérience et il devient de plus en plus performant au fil de son entraînement, à mesure de son exposition à davantage de données.

Dans la vie quotidienne, le Machine Learning est utile dans des domaines assez variés. Il permet à titre d'exemple à des aspirateurs robots de faire le ménage seuls, à sa boîte mail de détecter les spams, aux outils d'analyse médicale de repérer les tumeurs plus facilement. Il est sous-jacent aux moteurs de recommandations utilisés par divers services comme YouTube ou Amazon, réseaux sociaux comme Facebook ou Twitter, assistants vocaux comme Siri ou Alexia.

On distingue deux types d'apprentissage ; Le premier est l'apprentissage supervisé. Il correspond à un apprentissage où les données utilisées au cours de l'entraînement sont déjà étiquetées, permettant ainsi à l'algorithme de savoir quel élément chercher dans les données qui se présentent à lui. À la suite de l'entraînement, le modèle entraîné sera en mesure de retrouver les mêmes éléments sur des données déjà étiquetées.

Le deuxième, que l'on appelle apprentissage non supervisé, consiste quant à lui, à entraîner le modèle sur des données sans étiquette. L'algorithme parcourt les données sans aucun indice dans le but d'y découvrir des motifs récurrents.

2 Les fondations de l'apprentissage Automatique *Machine Learning*

2.1 Qu'est ce que le Machine Learning ?

Le machine learning tel que défini par son inventeur Arthur Samuel, un mathématicien américain est la science qui consiste à laisser l'ordinateur apprendre quel calcul effectuer, plutôt que de lui donner le calcul (c'est à dire de le programmer de façon explicite).

2.2 Les méthodes d'apprentissage

Pour donner à un ordinateur la capacité d'apprendre, on utilise des méthodes d'apprentissage qui sont fortement inspirées de la façon dont nous, les êtres humains, apprenons à faire quoi que ce soit. Parmi ces méthodes, on compte :

- L'apprentissage **supervisé** (*Supervised Learning*)
- L'apprentissage non **supervisé** (*Unsupervised Learning*)
- L'apprentissage par **renforcement** (*Reinforcement Learning*)

Dans la suite du rapport, nous nous concentrerons uniquement sur l'apprentissage supervisé, qui est la méthode utilisée dans notre projet.

2.3 L'Apprentissage Supervisé

Pour mieux visualiser ce que c'est que l'apprentissage Supervisé, Imaginons que vous souhaitiez apprendre une nouvelle langue l'arabe par exemple.

Votre premier réflexe sera de vous procurer un livre sur le sujet ou d'engager un professeur particulier. Le rôle du professeur ou du livre sera de superviser votre apprentissage en vous fournissant des exemples de traductions arabe-français que vous devriez mémoriser.

C'est exactement le même principe avec l'apprentissage supervisé, on fournit à la machine des exemples (\mathbf{x}, \mathbf{y}) dans le but de lui faire apprendre la **relation** qui relie \mathbf{x} à \mathbf{y} , comme sur la "Figure 1".

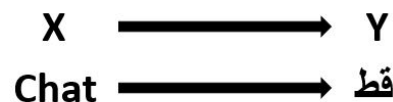


FIGURE 1 – Apprentissage par l'exemple

2.4 Les notions clés du Machine Learning

2.4.1 Le Dataset

En Machine Learning, On compile ces **exemples** (x,y) dans un tableau que l'on appelle **Dataset** :

- La variable y porte le nom de Cible (*En anglais : Target*), c'est la valeur qu'on cherche à prédire.
- La variable x porte le nom de Facteur (*En anglais : Feature*). Un facteur influence la valeur de y , et on a en général plusieurs features $(x_1, x_2, x_3, \dots, x_n)$ dans notre Dataset que l'on regroupe dans une matrice X .

Facteurs (Features)					Cible (Target)
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

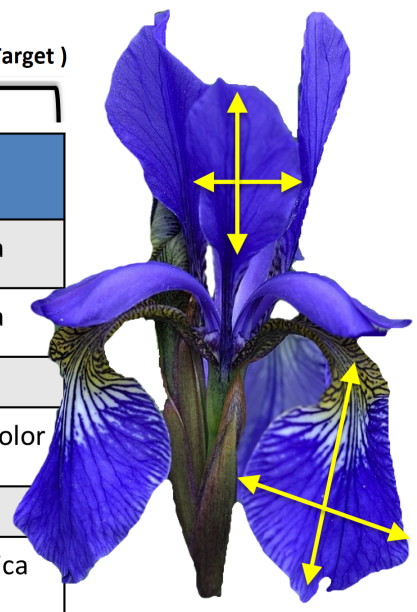


FIGURE 2 – Exemple Dataset : Iris DataSet

2.4.2 Le modèle

Un modèle de machine learning est le résultat généré lorsque vous entraînez votre algorithme d'apprentissage automatique avec des données. Après la formation, lorsque

vous fournissez des données en entrée à un modèle, vous recevez un résultat en sortie. Par exemple, un algorithme prédictif crée un modèle prédictif.

2.4.3 La fonction coût *Cost Function*

Un modèle nous retourne des erreurs par rapport à notre Dataset .On appelle fonction coût ,l'ensemble de ces erreurs.

Ainsi l'objectif de l'apprentissage supervisé ,c'est de trouver les paramètres du modèle qui minimisent la fonction coût .Pour cela , on utilise un algorithme d'apprentissage ,l'exemple le plus courant étant "La descente de gradient" (*En anglais : Gradient Descent*).

2.4.4 L'algorithme d'apprentissage

Un algorithme d'apprentissage est un algorithme utilisé dans l'apprentissage automatique pour aider la technologie à imiter le processus d'apprentissage humain.

L'un des exemples les plus significatifs est la **La Descente de Gradient** qui est un algorithme d'optimisation permettant de trouver le minimum de n'importe quelle fonction convexe en convergeant progressivement vers celui-ci.

La régression logique, la régression linéaire, les arbres de décision et les forêts aléatoires sont tous des exemples d'algorithmes d'apprentissage.

2.5 Les applications les plus courantes du Machine Learning

Avec l'apprentissage supervisé on peut développer des modèles pour résoudre deux type de problèmes :

- Les problèmes de **Régression**
- Les problèmes de **Classification**

2.5.1 Problème de Régression

Dans les problèmes de régression,on cherche à prédire la valeur d'une variable continue,c'est à dire une variable qui peut prendre une infinité de valeurs.Par exemple le prix d'un appartement (y) selon sa surface habitable (x).

2.5.2 Problème de Classification

Dans un problème de classification,on cherche à classer un objet dans différentes classes,c'est à dire que l'on cherche à prédire la valeur d'une variable **discrète**(qui ne prend qu'un nombre fini de valeurs).

Dans la suite de notre rapport , nous nous intéresserons uniquement aux problèmes de classification :

- classification d'un commentaire selon l'opinion.
- classification d'un commentaire selon sa source d'appartenance.

3 Les fondations de l'apprentissage Profond (*Deep Learning*)

3.1 Qu'est ce que le Deep Learning

L'apprentissage profond ou Deep Learning est l'une des technologies principales du Machine Learning. Avec le Deep learning, nous utilisons des algorithmes capables de mimer en quelques sortes les actions du cerveau humain grâce à des réseaux de neurones artificielles.

3.2 Le fonctionnement des réseaux de neurones artificielles

Les réseaux sont composés de dizaines voire de centaine de "couches" de neurones, chacune recevant et interprétant les informations de la couche précédente. Plus il y a de couches plus l'apprentissage est profond.

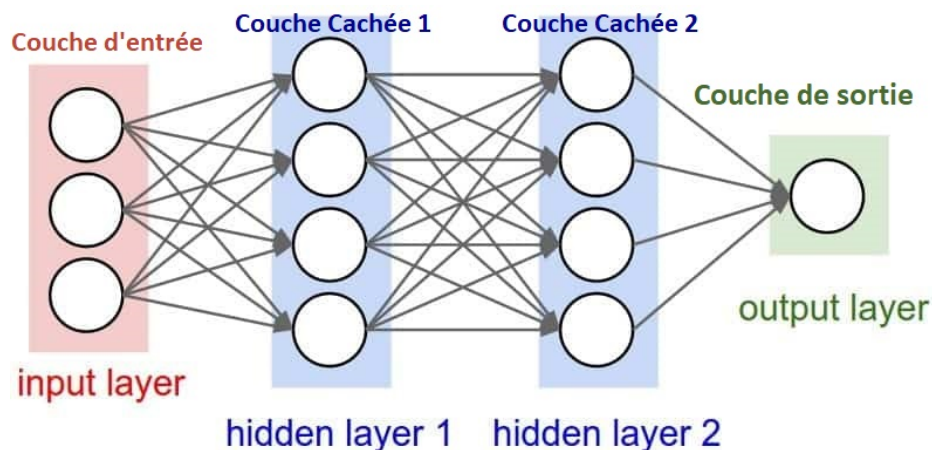


FIGURE 3 – Exemple simplifié d'un réseau de neurones artificielles

On remarque sur la figure précédente un niveau d'entrée *Input Layer* à gauche, un niveau de sortie *Output Layer* à droite, et plusieurs niveaux cachés entre les deux. Les petits ronds sont appelés les **neurones** et représentent des fonctions d'activation.

Mais qu'est ce que c'est que une fonction d'activation ?

Pour faciliter la compréhension de la fonction d'activation, on fait souvent l'analogie entre les réseaux de neurones artificielles et le cerveau humain : La fonction d'activation produit une sortie si les entrées qu'elle reçoit dépassent un certain seuil, à la manière qu'un neurone biologique produit un signal électrique en fonction des stimulus qu'il reçoit au Dendrites (Entrées des neurones).

//Figure d'un neurone biologique

Dans un neurone, ce signal circule jusqu'aux différents terminaux de l'axone pour être transmis à d'autres neurones, tout comme la fonction d'activation envoie sa sortie aux neurones de la couche suivante.

Cette analogie est en réalité loin de transcrire la réalité, il demeure indéniable que le cerveau humain dépasse de loin l'architecture "simpliste" des réseaux de neurones.

3.3 Exemples d'usage du Deep Learning

Ces fameux réseaux de neurones confèrent à nos algorithmes une énorme puissance ,leurs permettant ainsi d'entraîner les machines sur des tâches très avancées :

- La reconnaissance Faciale
- L'analyse des sentiments
- l'analyse du langage naturel
- Etc.

3.4 Entraînement d'un Réseau de Neurone

```
//Forward  
//Backward
```

4 Approches pour une tâche

4.1 Présentation de notre DataSet

Nous avons présenté plus haut la notion de dataset. Au tout début , notre travail consistera à apprendre un modèle à partir d'une colonne de notre dataset avant d'être capable de prendre en compte deux colonnes de ce dernier simultanément.

Après l'apprentissage de la classification , notre objectif a été de travailler sur des données textuelles .

Notre dataset est tiré de la base de l'UCI :

<https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences>

Il a été créée par : "From Group to Individual Labels using Deep Features", Kotzias et. al., KDD 2015".

Il est présenté sous forme de tableau où l'on trouve en première colonne , du texte , notamment des commentaires constituant l'avis de consommateurs sur des sites de e-commerce .

Sur la deuxième colonne , on indique si le commentaire en question est positif ou négatif , la valeur de la case ne peut être que 0 (pour négatif) et 1 (pour positif).

La troisième colonne , indique la source du commentaire , autrement dit , de quel site il est issu. On a trois sources possibles et par conséquent 3 valeurs possibles :

- Amazon : 0
- Imdb : 1
- Yelp : 2

On dispose de 500 commentaires positifs et 500 commentaires négatifs pour chaque source.

La difficulté de notre Dataset réside dans le fait que nous sommes amenés à traiter le langage naturel lorsqu'il s'agit de notre Feature "**Review**".

Dans la suite du rapport , nous discuterons brièvement du traitement automatique du langage naturel avant de nous attaquer aux différents modèles développés lors de ce TER.

1	sentence	sentiments	source
2	So there is no way for me to plug it in here in the US unless I go by a converter.	0	amazon
3	Good case, Excellent value.	1	amazon
4	Great for the jawbone.	1	amazon
5	Tied to charger for conversations lasting more than 45 minutes. MAJOR PROBLEMS!!	0	amazon
6	The mic is great.	1	amazon
7	I have to jiggle the plug to get it to line up right to get decent volume.	0	amazon
8	If you have several dozen or several hundred contacts, then imagine the fun of sending each of them one by one.	0	amazon
9	If you are Razr owner...you must have this!	1	amazon
10	Needless to say, I wasted my money.	0	amazon
11	What a waste of money and time!	0	amazon
12	And the sound quality is great.	1	amazon
13	He was very impressed when going from the original battery to the extended battery.	1	amazon
14	If the two were separated by a mere 5+ ft I started to notice excessive static and garbled sound from the headset.	0	amazon
15	Very good quality though!	1	amazon
16	The design is very odd, as the ear "clip" is not very comfortable at all.	0	amazon
17	Highly recommend for any one who has a blue tooth phone.	1	amazon
18	I advise EVERYONE DO NOT BE FOOLED!	0	amazon
19	So Far So Good!	1	amazon

FIGURE 4 – Notre Dataset

4.2 Traitement automatique du langage naturel *NLP*

En vue que la machine puisse comprendre les données textuelles , qui constituent des données linguistiques appartenant au langage humain , il faudrait qu'elle puisse les traduire.

Cette traduction est possible par un traitement du langage naturel (Natural Language Preprocessing) que l'on appelle plus communément NLP. Il s'agit d'une technologie , qui est l'un des principaux moteurs de l'intelligence artificielle .

Le champ d'application du NLP est vaste ,ceux qui nous intéressent ici sont d'un coté l'analyse des sentiments (*Opinion Mining*) qui permet d'évaluer le niveau de satisfaction des clients vis-à-vis de produits vendus par des entreprises (ici des sites de e-commerce) en vue d'améliorer les produits vendus , de réajuster les prix et d'adapter la publicité et d'un autre la classification de texte qui permet de catégoriser un ensemble de texte (ici les catégories sont les sources).

La phase du prétraitement est l'une des phases les plus cruciales avant de pouvoir exploiter nos données textuelles .

4.2.1 Etape 1 : Nettoyer le jeu de données

La première étape consiste à préparer et à "nettoyer" notre jeu de données pour qu'il puisse être exploitable par les machines. Elle permet également de mettre en avant les éléments principaux du texte que l'algorithme utilisera.

figure : à voir

Il existe différentes méthodes de NLP pour transformer les données de textes bruts en données exploitables. Il existe par exemple , des bibliothèques contenant des règles et des méthodes qui permettent aux ordinateurs de classer les segments de textes.

— La suppression des mots vides *Stop Words*

La première étape consiste à supprimer les Stop Words qui représentent les mots

vides autrement dit des mots utilisés souvent dans une langue et qui ne sont pas des mots-clés à la compréhension d'un texte dans la mesure où ils n'ont pas d'impact sur la compréhension du sens de la phrase .

Cette technique du NLP vise à supprimer les articles, les pronoms et les prépositions. Cela , dans le but de libérer de l'espace dans la base de données et d'accélérer le temps de traitement. Il existe des mots par défaut dans la liste des stop-words pour chaque langue dans la librairie NLTK, (Natural Language Toolkit), qui représente une suite de programmes conçue pour le traitement naturel symbolique et statistique du langage humain en langage Python. Il existe d'autres bibliothèques créées pour le traitement naturel du langage et celle-ci est l'une des plus puissantes.

— La segmentation (*tokenisation*)

Cette fois, le principe consiste à segmenter le texte en mots. Les segments de textes sont appelés «tokens ».

Figure : phrase avant apres tokenisation

— La lemmatisation

La lemmatisation permet à chaque mot des tokens d'être représenté sous forme canonique. La forme canonique d'un verbe est son infinitif , tandis que celle d'un nom est son masculin singulier. Cette substitution vise à ne conserver que le sens des mots.

Figure : tokens avant/après lemmatisation

— La racinisation (*Stemming*)

En NLP, on dispose d'une autre méthode de nettoyage des textes , le stemming. Elle consiste en la suppression des préfixes et des suffixes, elle coupe une partie du mot en vue de n'en dégager que la racine. Il existe des règles pour déterminer la bonne partie à couper. Cette approche nécessite de se référencer à des dictionnaires détaillés dans lesquels l'algorithme peut rechercher et relier les mots à leurs radicaux.

— Notre Fonction MyCleanText

On effectue en outre , dans le cadre du prétraitement , une suppression de la ponctuation , des symboles et des chiffres et on met tout le texte en minuscule.

Nous avons utilisé tous ces prétraitements sur notre jeu de données. Ces derniers sont regroupés au sein d'une fonction.

Figure : texte avantès MyCleanText

— La vectorisation

En vue de pouvoir appliquer les méthodes de Deep Learning sur le langage naturel , il est nécessaire de transformer nos données textuelles en données numériques. On appelle cette étape de transformation , la vectorization.

Figure : Texte avant/apres vectorisation

Les prétraitements que nous venons de présenter sont réalisés à partir des paramètres qui permettent la création de structures qui permettent d'effectuer l'étape de vectorization. Celles présentées par notre encadrant sont au nombre de deux.

Les sacs de mots (Bag of Words)

Cette méthode consiste à compter le nombre d'occurrences de chaque token (mot) pour chaque texte. Elle permet de créer une matrice d'occurrence où chaque texte est représenté par un vecteur d'occurrence.

La limite de cette méthode est que des mots ont tendance à se répéter , notamment dans la langue française. On ne peut donc pas savoir dans le cas d'une occurrence élevée , s'il s'agit d'un mot pertinent pour la classification que l'on veut effectuer ou s'il ne s'agit que d'un mot couramment utilisé dans la langue. Cette limite peut conduire à des résultats erronés.

Fréquence du terme * Nombre de documents contenant ce mot (Term Frequency *Inverse Document Frequency) - TF-IDF

La méthode que l'on utilise dans le cadre de notre projet est TF-IDF.

Dans la méthode précédente , on ne s'est intéressés qu'à la fréquence d'apparition des différents tokens dans le corpus pour chaque texte . Dans celle-ci , on prend compte d'un autre paramètre : le nombre d'apparitions dans les autres documents. On considère que si un mot apparaît dans d'autres documents, il est moins représentatif du document qu'un mot qui apparaît uniquement dans ce document.

On compte le nombre d'occurrences des tokens pour chaque texte que l'on divise ensuite par le nombre d'occurrences total de ces mêmes tokens dans tout le corpus.

On définit le poids du terme x présent dans le document y par la relation suivante :

FIGURE 5

Où :

$t_{fx,y}$ est la fréquence du terme x dans y ;

df_x est le nombre de documents contenant x ;

N est le total de documents.

On a par conséquent , une représentation vectorielle par poids et non pas par occurrence.

// est ce que je dois omettre les limites de cette méthodes dans un soucis de cohérence ?

Après le traitement des données et la vectorization de ces dernières , on applique l'algorithme d'apprentissage.

4.2.2 Etape 2 :

4.3 Classification binaire selon l'opinion

Au cours de notre projet , on a découvert divers modèles d'apprentissage par le biais de la documentation délivrée par notre encadrant.

Notre objectif était de choisir le modèle le plus adéquat à notre problème.

On a d'ailleurs pris connaissance des critères de sélection d'un bon modèle. Un bon modèle est en effet un modèle qui permet une généralisation , autrement dit qui soit capable d'établir des prédictions sur les données fournies mais aussi sur de nouvelles données.

On considère qu'il existe des modèles qui seront très forts sur le jeu d'entraînement mais mauvais sur certaines données.

De tels modèles sont soit trop complexes du fait qu'ils collent trop aux données d'apprentissage et à leurs moindres détails , c'est ce que l'on appelle le sur-apprentissage (overfitting) , soit trop simples , ce que l'on appelle le sous-apprentissage.

On estime que la complexité d'un modèle est proportionnelle aux nombre de paramètres de ce dernier.

Parmi les modèles découverts , on compte :

Random Forest (Forêt d'arbres décisionnels) Les forêts d'arbres décisionnels (Random Forest) constituent une technique d'apprentissage ensembliste qui reposent sur des arbres de décision.

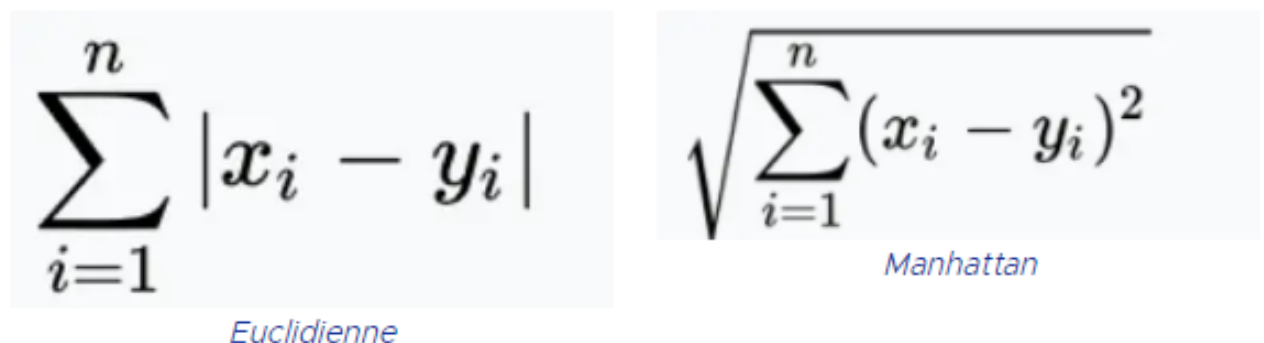
Le modèle random forest utilise des données fractionnées à partir d'un ensemble de données d'origine en vue de créer des arbres décisionnels (decision tree) multiples. Et en sélectionnant aléatoirement un sous-ensemble de variables à chaque étape de l'arbre décisionnel. Le modèle sélectionne ensuite le mode de toutes les prédictions de chaque arbre décisionnel.

KNN L'algorithme des KNN se divise en 5 étapes , qui sont les suivantes :

Étape 1 : Sélectionnez le nombre K de voisins

Étape 2 : Calculez la distance du point non classifié aux autres points.

Étape 3 : Prenez les K voisins les plus proches selon la distance calculée.



$$\sum_{i=1}^n |x_i - y_i|$$

Euclidienne

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Manhattan

FIGURE 6 – formules de calcul des distances

Étape 4 : Parmi ces K voisins, comptez le nombre de points appartenant à chaque catégorie.

Étape 5 : Attribuez le nouveau point à la catégorie la plus présente parmi ces K voisins.

On peut utiliser divers modèles d'apprentissage pour les problématiques NLP y compris ceux propres au Deep Learning.

Les modèles de réseaux de neurones nécessitent une phase de prétraitement du texte moins sophistiquée : les couches de neurones peuvent extraire de manière automatique les features. Cela permet alors de construire des modèles avec peu de prétraitement des données.

En dehors de l'avantage cité dernièrement , que l'on appelle " feature engineering", on estime que les capacités d'apprentissage des algorithmes de Deep Learning sont en général plus puissantes que celles de Machine Learning classique, ce qui permet d'obtenir de meilleurs scores sur différentes tâches complexes de NLP.

C'est pourquoi notre encadrant nous suggère d'utiliser comme modèle , les réseaux de neurones.

Un autre choix à effectuer s'est présenté à lui , celui de l'environnement de programmation (framework) à utiliser.

Un framework est une ressource mise à la disposition des programmeurs pour leur éviter de tout coder à la main. Ainsi un modèle complexe comme les réseaux de neurones peut être utilisé avec seulement quelques lignes de codes.

Parmi les options qu'il avait , on compte Keras et Pytorch.

Keras est un framework développé en 2015 qui gagne en popularité du fait de son API qui s'avère facile d'utilisation et qui s'inspire de scikit-learn, la librairie de Machine Learning standard de Python.

Keras est certes facile à utiliser , mais manque de certaines fonctionnalités « bas niveau » , Keras ne permet pas de travailler sur de grands datasets et offre une performance assez faible , cela assez lentement.

Donc , quelques mois plus tard, un nouveau framework ,qui permet d'utiliser Keras et qui est plus performant est créé et devient le framework de référence , il s'agit de TensorFlow. À l'inverse, Tensorflow donne accès à ces fonctionnalités manquantes dans Keras, mais ne ressemble pas au style habituel de Python et dispose d' une documentation très compliquée.

PyTorch a été créé comme solution , il combine les avantages de Keras (facile d'utilisation) et TensorFlow (rapidité et meilleures performances) et pallie à leurs désavantages respectifs.

PyTorch dispose d'une API accessible et facile à personnaliser et permet de créer de nouveaux types de réseaux et des architectures inédites. Par ailleurs , vu qu'il est de plus bas niveau que Keras il dispose d'une architecture plus complexe.

Le modèle utilisé aurait pu être codé en quelques lignes de codes en utilisant Keras . Seulement , dans le cadre du projet , il nous a été demandé de travailler en PyTorch pour rendre la tâche plus complexe.

Cela implique de définir soi-même ses variables et d'initialiser toutes ses données ainsi que de créer soi-même la fonction forward (où l'on positionne les layers) et la fonction backward alors que ces dernières auraient été fournies par Keras.

En Keras , l'apprentissage du modèle est initié par la méthode `.compile()`. Parmi les arguments on trouve la fonction de perte , la fonction Perte (Loss) est utilisée pour trouver une erreur ou une déviation dans le processus d'apprentissage.

Dans l'exemple suivant , la fonction de perte utilisée est "crossentropy". On retrouve aussi l'optimisation qui représente un processus optimisant les poids d'entrée en comparant la

prédiction et la fonction de perte.

Dans l'exemple , le module utilisé comme optimiseur est "Adam".

On compte aussi les métriques (Metrics) utilisées pour évaluer les performances du modèle , similaire à la fonction de perte, mais pas utilisé dans le processus de formation. Dans l'exemple le module métriques utilisé est "accuracy".

Cette étape prend comme on peut le voir une ligne de code.

```
Python >>>
>>> model.compile(loss='binary_crossentropy',
...               optimizer='adam',
...               metrics=['accuracy'])
>>> model.summary()
```

FIGURE 7 – Méthode compile

Après le compile() , l'étape d'entraînement du modèle est nécessaire (fitting).

Cette étape est possible grâce à la méthode d'ajustement fit(). Celle-ci permet d'évaluer le modèle d'entraînement.

Elle permet en outre de représenter graphiquement les performances du modèle.

Elle prend également une ligne de code.

```
Python >>>
>>> history = model.fit(X_train, y_train,
...                     epochs=100,
...                     verbose=False,
...                     validation_data=(X_test, y_test),
...                     batch_size=10)
```

FIGURE 8 – Méthode fit

Parmi les arguments , on a principalement Xtrain et ytrain qui représentent le tuple qui va permettre d'évaluer les données.

On a le nombre d'époques (epochs) qui constitue le nombre de fois où le modèle doit être évalué pendant la formation. Et le "batch size" représente la taille des instances de formation.

De surcroît, en Keras , chaque définition de layer requiert une ligne de code.

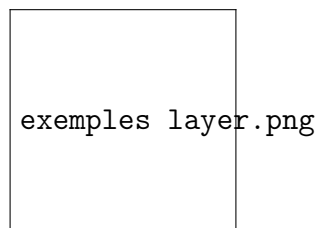


FIGURE 9 – création layers

Dans PyTorch , on est par contre , obligés d'implémenter la boucle de forward et la boucle backward que nous avons défini précédemment. Et c'est dans la boucle forward

que les layers sont définis.

Ce qui reste similaire aux deux framework , dans le cadre de la classification est d'une part , le nombre de neurones en sortie et le choix de la fonction d'activation.

La première classification que l'on a dû effectuée est une classification binaire selon l'opinion , pour chaque commentaire , la sortie du réseau de neurones est soit 0 (s'il est négatif) , soit 1 s'il est positif.

Comme nous l'avons expliqué plus tôt , le nombre de neurones sur la couche de sortie sera égal à un pour une classification binaire comme notre première classification et la fonction d'activation sera une sigmoid.

On peut observer ces deux choix sur cette portion de notre code.

```
def __init__(self, n_inputs):
    super(MLP, self).__init__()
    # input to first hidden layer
    self.hidden1 = Linear(n_inputs, 10)
    kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
    #relu est la fonction d'activation
    self.act1 = ReLU()
    # second hidden layer
    self.hidden2 = Linear(10, 8)
    kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
    self.act2 = ReLU()
    # third hidden layer and output
    self.hidden3 = Linear(8, 1)
    xavier_uniform_(self.hidden3.weight)
    self.act3 = Sigmoid()
```

FIGURE 10 – version classification binaire

4.4 Classification MultiClasse selon la Source

Pour la classification selon les sources , le principe est similaire à deux exceptions près. Dans une classification ternaire , le nombre de neurones sur la couche de sorties est 3 et la fonction d'activation de classification multi classes est softmax.

On peut observer ces deux dissimilitudes avec la classification binaire sur cette portion de notre code.

```
def __init__(self, n_inputs):
    super(MLP, self).__init__()
    # input to first hidden layer
    self.hidden1 = Linear(n_inputs, 10)
    kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
    self.act1 = ReLU()
    # second hidden layer
    self.hidden2 = Linear(10, 8)
    kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
    self.act2 = ReLU()
    # third hidden layer and output
    self.hidden3 = Linear(8, 3)
    xavier_uniform_(self.hidden3.weight)
    self.act3 = Softmax(dim=1)
```

FIGURE 11 – version classification multiclass

5 Classification MultiTaches

5.1 Etapes Générales de la classification

Avant de travailler sur notre jeu de données , des exemples de classification nous ont été présentés parmi eux :

L'ensemble de données Ionosphère constitue un document csv sur lequel on a effectué une classification binaire.

Cet ensemble de données consiste à prédire s'il existe une certaine structure dans l'atmosphère compte tenu des retours d'un radar. Pour chaque retour , on aura en sortie une valeur de 0 (pour un non) et une valeur de 1 (pour un oui).

Nous utiliserons un LabelEncoder pour coder les étiquettes de chaîne en valeurs entières 0 et 1.

Le modèle prédit la probabilité de la classe 1 et utilise la fonction d'activation sigmoïde. Le modèle est optimisé en utilisant la descente de gradient stochastique et cherche à minimiser la perte d'entropie croisée binaire.

a)LOAD THE DATA :

PyTorch fournit la classe générique CSVDataset qui permet la manipulation du fichier CSV entré en paramètres pour la construction d'un jeu de données.

Elle récupère les features d'un côté pour créer la matrice X et le label d'un autre pour créer un vecteur y. Cette classe est personnalisable.Mais elle dispose d'à peu près la même structure dans toutes les classifications étudiées.

La méthode len() permet d'obtenir la longueur de l'ensemble de données (nombre de lignes) et la méthode getitem() permet d'obtenir un échantillon spécifique par index. voici sa structure :

```
class CSVDataset(Dataset):
    # load the dataset
    def __init__(self, path):
        # store the inputs and outputs
        self.X = ...
        self.y = ...

    # number of rows in the dataset
    def __len__(self):
        return len(self.X)

    # get a row at an index
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]
```

Lors du chargement de notre ensemble de données, on peut effectuer toutes les transformations requises, telles que la mise à l'échelle ou l'encodage.

Dans la classification Ionosphere par exemple ,les données de X et y sont transformés en float.

```
# ensure input data is floats
self.X = self.X.astype('float32')
# label encode target and ensure the values are floats
self.y = LabelEncoder().fit_transform(self.y)
self.y = self.y.astype('float32')
self.y = self.y.reshape((len(self.y), 1))
```

Le dataset est donc créé en faisant appel au constructeur de la classe `CSVDataset`, et cela en mettant le chemin (path) du fichier en csv en paramètre.

```
# create the dataset
dataset = CSVDataset(path)
```

la méthode `random_split` permet de scinder le dataset en jeu d'entraînement et en jeu de test de la manière suivante :

```
train, test = random_split(dataset, [[...], [...]])
```

b) définition du modèle

La définition du modèle se fait par la définition d'une classe (MLP pour multilayer perceptron) qui représente un petit modèle de réseau de neurones.

Le constructeur de cette classe définit les couches du modèle et la fonction `forward()` qui va transmettre la propagation à travers les couches définies du modèle.

Il existe diverses couches, telles que `Linear` pour les couches entièrement connectées, `Conv2d` pour les couches convolutionnelles et `MaxPool2d` pour les couches de regroupement.

En vue de créer la première couche autrement dit la couche d'entrée on crée une couche (layer) par le biais de l'application de la fonction `linear` sur le nombre de colonne de la matrice `X`. Ce nombre représente comme on l'a mentionné plus haut le nombre de neurones dans la couche d'entrée du modèle.

```
self.layer = Linear(n_inputs, 1)
```

Suite à cela, on spécifie une fonction d'activation. Les fonctions d'activation peuvent également être définies en tant que couches, telles que `ReLU`, `Softmax` et `Sigmoid`.

```
self.activation = Sigmoid()
```

Dans la classification `Ionosphere`, comme on peut le voir la fonction d'activation utilisée sur la couche de sortie est `Sigmoid` vu qu'il s'agit d'une classification binaire.

```
class MLP(Module):
    # define model elements
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input to first hidden layer
        self.hidden1 = Linear(n_inputs, 10)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
```

```
self.act1 = ReLU()  
# second hidden layer  
self.hidden2 = Linear(10, 8)  
kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')  
self.act2 = ReLU()  
# third hidden layer and output  
self.hidden3 = Linear(8, 1)  
xavier_uniform_(self.hidden3.weight)  
self.act3 = Sigmoid()
```

Pour la classification multiclasse Iris , la fonction d'activation utilisée sur la couche de sortie est softmax.

```
class MLP(Module):  
    # define model elements  
    def __init__(self, n_inputs):  
        super(MLP, self).__init__()  
        # input to first hidden layer  
        self.hidden1 = Linear(n_inputs, 10)  
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')  
        self.act1 = ReLU()  
        # second hidden layer  
        self.hidden2 = Linear(10, 8)  
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')  
        self.act2 = ReLU()  
        # third hidden layer and output  
        self.hidden3 = Linear(8, 3)  
        xavier_uniform_(self.hidden3.weight)  
        self.act3 = Softmax(dim=1)
```

Pour le problème de régression , la fonction d'activation utilisée sur la couche de sortie est Sigmoid comme on peut le voir :

```
class MLP(Module):  
    # define model elements  
    def __init__(self, n_inputs):  
        super(MLP, self).__init__()  
        # input to first hidden layer  
        self.hidden1 = Linear(n_inputs, 10)  
        xavier_uniform_(self.hidden1.weight)  
        self.act1 = Sigmoid()  
        # second hidden layer  
        self.hidden2 = Linear(10, 8)  
        xavier_uniform_(self.hidden2.weight)  
        self.act2 = Sigmoid()  
        # third hidden layer and output  
        self.hidden3 = Linear(8, 1)  
        xavier_uniform_(self.hidden3.weight)
```

Après la création des couches du modèle , on passe au forward , qui est une méthode de notre classe qui met en évidence la structure du réseau. Elle nous indique par exemple , au tout début , le passage par la première couche :

```
X = self.layer(X)
```

puis le passage dans la fonction d'activation :

```
X = self.activation(X)
```

La matrice X est ainsi modifié au fur-et-à-mesure du passage dans les différentes couches. Sa valeur est à la fin de chaque passage stockée et récupérée à la fin de chaque forward. Voici en exemple , la structure du réseau de neurones de notre classification Ionosphere :

```
def forward(self, X):  
    # input to first hidden layer  
    X = self.hidden1(X)  
    X = self.act1(X)  
    # second hidden layer  
    X = self.hidden2(X)  
    X = self.act2(X)  
    # third hidden layer and output  
    X = self.hidden3(X)  
    X = self.act3(X)  
    return X
```

c)entraîne le modèle :

Une fois que le modèle est défini , on crée la fonction de coût que l'on utilisera à la fin de chaque propagation pour mesurer l'erreur entre la prédiction du modèle et la véritable valeur. Cette dernière est projetée à l'intérieur du réseau de neurones pour mettre à jour tous les poids..

Dans la classification Ionosphere , la fonction de perte utilisée est BCELoss.

```
criterion = BCELoss()
```

Dans la classification multiclass Iris , la fonction de perte utilisée est CrossEntropyLoss.

```
criterion = CrossEntropyLoss()
```

Pour la régression on a utilisé MSELoss.

```
criterion = MSELoss()
```

Les fonctions de perte courantes incluent les éléments suivants :

BCELoss : perte d'entropie croisée binaire pour la classification binaire.

CrossEntropyLoss : perte d'entropie croisée catégorielle pour la classification multiclass.

MSEloss : perte quadratique moyenne pour la régression.

Pour l'entraînement , on doit créer la boucle qui va permettre la propagation. Pour se faire , il nous a été demandé de prendre des échantillons du jeu d'apprentissage défini et

non pas le jeu en entier en vue d'éviter un sur-apprentissage.

La taille de cette échantillon a été déterminé dans la variable `batch_size` dans la portion de code suivante :

```
train_dl = DataLoader(train , batch_size=32, shuffle=True)
```

Dans cet exemple , à chaque epoch , seulement 32 lignes de la matrice d'apprentissage sont envoyés au réseau.

On fait passer chaque ligne dans le réseau pour chaque epoch grâce à une boucle :

```
# enumerate epochs
for epoch in range(100):
    # enumerate mini batches
    for i, (inputs , targets) in enumerate(train_dl):
        ...
```

Après le forward , il est nécessaire de mettre les gradients à 0.

```
# clear the gradients
optimizer.zero_grad()
```

On récupère la sortie du modèle dans une variable `yhat` à chaque fin de propagation.

```
yhat = model(inputs)
```

Suite à cela , on calcule la valeur de la fonction coût à laquelle on entre en paramètres le sortie du modèle (notre `yhat`) et les données réelles (`targets`) en vue de comparaison.

```
loss = criterion(yhat , targets)
```

S'en suit , une mise à jour des poids après l'étape backward.

```
# tape backward
loss.backward()
#mise à jour des poids
optimizer.step()
```

Chaque mise à jour du modèle implique le même schéma général composé de :

- Effacement du dernier gradient d'erreur.
- Passage tout au long du modèle
- Calcul de la perte pour la sortie du modèle.
- Rétropropagation de l'erreur à travers le modèle.
- Mise à jour le modèle dans le but de réduire les pertes.

Voici l'exemple de la classification Ionosphere qui ne diffère pas des autres classifications étudiées.

```
# enumerate mini batches
```

```

for i, (inputs, targets) in enumerate(train_dl):
    # clear the gradients
    optimizer.zero_grad()
    # compute the model output
    yhat = model(inputs)
    # calculate loss
    loss = criterion(yhat, targets)
    # credit assignment
    loss.backward()
    # update model weights
    optimizer.step()

```

d)évaluation du modèle :

En vue d'évaluer le modèle , on a besoin de créer une boucle qui permet de faire passer le jeu de test dans le modèle et qui récupère ce que le modèle renvoie à la sortie. Dans ce qui suit , la sortie est stockée dans yhat.

```

for i, (inputs, targets) in enumerate(test_dl):
    # evaluate the model on the test set
    yhat = model(inputs)

```

Après cela , il sera question de calculer l'accuracy en confrontant les données réelles et les prédictions entrés en paramètres dans la méthode accuracy_score.

```

# calculate accuracy
acc = accuracy_score(actuals, predictions)

```

e)prédiction

Dans la méthode prediction , en passant une ligne en paramètre de notre modèle , on obtient notre prédiction.

Voyons l'exemple de Ionosphère en vue d'illustrer cela :

```

#cr ation du mod le MLP
model = MLP(34)

#pr diction pour une ligne du mod le
def predict(row, model):
    # convert row to data
    row = Tensor([row])
    # make prediction
    yhat = model(row)
    # retrieve numpy array
    yhat = yhat.detach().numpy()
    return yhat

```

5.2 Classification Multitâche sur les Images

//Myriam

5.3 Classification Multitâche sur notre DataSet

6 Organisation du travail

6.1 Organisation prévisionnelle

//Diagramme de GANTT

6.2 Outils Utilisés

//Outils de collaborations : partie ennuyeuse //en haut rajouter , les outils reellement utilisés Pytorch , Git , Scikit learn ,Python

L'objectif de notre travail, est de présenter la meilleure version de notre projet dans le temps imparti, et d'arriver à collaborer tous ensemble de manière efficace. Pour ce faire, nous avons mis en place plusieurs outils :

6.2.1 Pytorch

PyTorch est une bibliothèque logicielle Python open source d'apprentissage machine qui s'appuie sur Torch développée par Meta. Dans le cadre de l'intelligence artificielle, le système permet de manipuler des tenseurs et de réaliser le calcul des algorithmes associés. Et ce qui nous permettra de mettre en place les classifications de données indispensables pour l'avancement de notre projet.

Certains pourraient se demander : Pourquoi le choix s'est porté sur Pytorch, et pas un autre framework Python tels que Keras ou Tensorflow par exemple ? Et bien, PyTorch est réputé pour sa simplicité, sa facilité d'utilisation, sa flexibilité, son utilisation efficace de la mémoire et ses graphes de calcul dynamiques. Il se sent également natif, ce qui rend le codage plus gérable et augmente la vitesse de traitement.

6.2.2 Python

Python est un langage de programmation interprété, multi-paradigme et multiplateformes. C'est aussi le langage le plus populaire dans le monde de l'intelligence artificielle. Python est orienté objet et se veut relativement facile d'accès. Il est très utilisé au sein de la communauté scientifique et particulièrement dans le domaine de l'IA.

6.2.3 Git

Git est un logiciel de gestion de versions créé en 2005 par Linus Torvalds, le créateur de Linux. Il permet de conserver un historique des modifications effectuées sur un projet afin de pouvoir rapidement identifier les changements effectués et de revenir à une ancienne version en cas de problème.

Les logiciels de gestion de versions sont quasiment incontournables aujourd'hui car ils facilitent grandement la gestion de projets, et permettent de travailler en équipe de manière beaucoup plus efficace.

6.2.4 Scikit learn

Scikit learn est une librairie Python qui donne accès à des versions efficaces d'un grand nombre d'algorithmes courants. Elle offre également une API propre et uniformisée. Par conséquent, un des gros avantages de Scikit-Learn est qu'une fois que vous avez compris l'utilisation et la syntaxe de base de Scikit-Learn pour un type de modèle, le passage à un nouveau modèle ou algorithme est très simple.

6.2.5 Google Drive

Google Drive est un service de stockage et de partage de fichiers dans le cloud lancé par la société Google. Ce service a permis à chaque membre du groupe de travailler de son côté pour ensuite pouvoir partager nos idées, ainsi que nos points de vue.

6.2.6 Collaboratory

Collaboratory est un outil qui permet d'écrire et d'exécuter du code Python dans le navigateur avec aucune configuration requise et un partage facile. Nous nous sommes servis de Collaboratory pour nos implémentations.

6.2.7 Messenger

Messenger est un système de messagerie instantanée créé par la société Facebook, et notre moyen de communication principal pendant toute la durée du projet. Après avoir créé un salon de discussion réunissant tous les membres du groupe, Messenger nous a permis de nous coordonner et de tenir les autres membres du groupe informés de nos avancées et/ou des difficultés rencontrées.

6.2.8 Zoom

Zoom est une application de vidéoconférence que nous utilisons chaque jour pour suivre nos cours en distanciel, et ce, pendant les deux dernières années, cette dernière nous a permis de faire des réunions de groupe à fin de rassembler nos connaissances, et de se réunir chaque vendredi à 8h30 avec notre Encadrant, qui nous aidait à corriger nos erreurs, et à avancer sur le projet.

6.2.9 Overleaf

Overleaf est une plateforme en ligne gratuite permettant d'éditer du texte en LATEX sans aucun téléchargement d'application. En outre, elle offre la possibilité de rédiger gratuitement des documents de manière collaborative. pour travailler de manière simultanée sur le rapport mais aussi fournir le lien du rapport à notre encadrant pour qu'elle puisse suivre son avancement, nous guider et nous corriger au besoin.

6.2.10 Miro

Miro est un outil collaboratif gratuit, permettant de travailler en ligne et à distance sur des projets communs. Nous l'avons utilisé pour tracer le diagramme de Gantt et guider notre travail. Le langage utilisé lors de la réalisation de ce projet est :

6.3 Répartition des tâches

//avant la conclusion , il y'a surement des choses à rajouter

7 Conclusion

Voici quelques commandes utiles :



FIGURE 12 – Légende de la figure

Ici, je cite l'image 12

$$\rho + \Delta = 42 \quad (1)$$

L'équation 1 est cité ici.

// TEMPORAIRE Pour écrire des variables dans le texte, il suffit de mettre le symbole \$ entre le texte souhaité comme : constante ρ .

Version 1 : Myriam Pour faire plus simple, nous définirons Le machine learning comme

la capacité pour une ligne de codage d'apprendre par elle-même, sans intervention humaine, et ce, à l'aide d'algorithmes conçus pour découvrir et analyser des "patterns", et réaliser des prédictions basées sur des statistiques. Pour ce faire, nous aurons besoin de connaître plusieurs notions :

.....

Version 2 :kenza sous forme de données labellisées, c'est à dire des données qui ont déjà été étiquetées avec le bon label (*"la bonne réponse"*) et on s'en sert par la suite pour prédire le label des nouvelles données qui ne sont pas encore étiquetées.

Un dataset en machine learning regroupe un ensemble de données. Celles-ci dépendent d'une variable associée aux valeurs. Les datasets peuvent être4 Approches pour une tache 2.1.4 la fonction coût (Gradient Descent) La Descente de Gradient est un algorithme d'optimisation qui permet de trouver le minimum de n'importe quelle fonction convexe en convergeant progressivement vers celui- ci. Cet algorithme est utilisé dans les problèmes d'apprentissage supervisé dans le but de minimiser la fonction coût, et ainsi,trouver le meilleur modèle. 2.1.5 l'algorithme d'apprentissage Un algorithme d'apprentissage est un algorithme utilisé dans l'apprentissage automa- tique pour aider la technologie à imiter

le processus d'apprentissage humain. Combinés à des technologies comme les réseaux de neurones, les algorithmes d'apprentissage créent des programmes d'apprentissage sophistiqués et impliqués. La régression logique, la régression linéaire, les arbres de décision et les forêts aléatoires sont tous des exemples d'algorithmes d'apprentissage.

2.2 C'est quoi le Deep Learning (Réseau de Neurones ..)

L'apprentissage profond (deep Learning en anglais) est une technique de machine learning reposant sur le modèle des réseaux neurones ; Il est basée sur des algorithmes capables de mimer les actions du cerveau humain grâce à des réseaux de neurones artificielles. Les réseaux sont composés de dizaines voire de centaines de «couches» de neurones, chacune recevant et interprétant les informations de la couche précédente.

3 Traitement des données textuelles NLP

4 Approches pour une tâche

4.1 Présentation De notre DataSet

Début de rédaction de l'axe //parler de dataset , rappeler qu'on la déjà définit //rap- peler l'enjeu de l'usage du dataset en préambule // dire que l'enjeu (classification implique le traitement de données textuelles) /* utiliser ça comme perche pour : présenter le jeu de données (+image) -expliquer dans quel but on a besoin d'utiliser le NLP évoquer les grandes étapes du traitement de données textuelles :

- a)parler des pré-traitement que l'on met en paramètres lors de la création d'une instance de type tf-idf et countvectorizer (dire qu'ils sont dans my clean text , éventuellement afficher la fonction)
- b)-parler de la vectorization : expliquer l'intérêt puis présenter les deux méthodes sac de mots et tf-idf , dire que l'on a utilisé tf-idf */

présentation de notre dataset : Nous avons présenté plus haut la notion de dataset. Au tout début , notre travail consistera à apprendre un modèle à partir d'une colonne de notre dataset avant d'être capable de prendre en compte deux colonnes de ce dernier simultanément. Après l'apprentissage de la classification , notre objectif a été de travailler sur des données textuelles . Notre dataset est tiré de la base de l'UCI : <https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences> et a été créée par : "From HAI606I Rapport - Classification de données textuelles... 3 4 Approches pour une tâche Group to Individual Labels using Deep Features', Kotzias et. al., KDD 2015". Il est présenté sous forme de tableau où l'on trouve en première colonne , du texte , notamment des commentaires constituant l'avis de consommateurs sur des sites de e-commerce . Sur la deuxième colonne , on indique si le commentaire en question est positif ou négatif , la valeur de la case ne peut être que 0 (pour négatif) et 1 (pour positif). La troisième colonne , indique la source du commentaire , autrement dit , de quel site il est issu. On a trois sources possibles et par conséquent 3 valeurs possibles : 0 pour Yeld , 1 pour Amazon et 2 pour Imbd. On dispose de 500 commentaires positifs et 500 commentaires négatifs pour chaque source.

4.2 NLP

En vue que la machine puisse comprendre les données textuelles , qui constituent des données linguistiques appartenant au langage humain , il faudrait qu'elle puisse les traduire. Cette traduction est possible par un traitement du langage naturel (Natural Language Preprocessing) que l'on appelle NLP. Il s'agit d'une technologie , qui est l'un des principaux moteurs de l'intelligence artificielle . Le champ d'application du NLP est vaste , ceux qui nous intéressent ici sont l'analyse des sentiments (Opinion Mining) qui permet d'évaluer le niveau de satisfaction des clients vis-à-vis de produits vendus par des entreprises (ici des sites de e-commerce) en vue d'améliorer les produits vendus , de réajuster les prix et d'adapter la publicité et la classification de texte qui permet de catégoriser un ensemble de texte (ici les catégories sont les sources). Elle comporte deux étapes : le prétraitement des données ainsi que le développement de l'algorithme permettant d'effectuer la tâche que l'on souhaite effectuer. La première étape consiste à préparer et à "nettoyer" notre jeu de données pour qu'il puisse être exploitable par les

machines. Elle permet également de mettre en avant les éléments principaux du texte que l'algorithme utilisera. Il existe différentes méthodes de NLP pour transformer les données de textes bruts en données exploitables. Il existe par exemple , des bibliothèques contenant des règles et des méthodes qui permettent aux ordinateurs de classer les segments de textes. La suppression des mots vides (Stop Words) La première étape consiste à supprimer les Stop Words qui représentent les mots vides autrement dit des mots utilisés souvent dans une langue et qui ne sont pas des mots-clés à la compréhension d'un texte dans la mesure où ils n'ont pas d'impact sur la compréhension du sens de la phrase . Cette technique du NLP vise à supprimer les articles, les pronoms et les prépositions. Cela , dans le but de libérer de l'espace dans la base de données et d'accélérer le temps de traitement. Il existe des mots par défaut dans la liste des stop-words pour chaque langue dans la librairie NLTK, (Natural Language Toolkit), qui représente une suite de programmes conçue pour le traitement naturel symbolique et statistique du langage humain en langage Python. Il existe d'autres bibliothèques créées pour le traitement naturel du langage et celle-ci est l'une des plus puissantes. La segmentation (tokenisation) Cette fois, le principe consiste à segmenter le texte en mots. Les segments de textes sont appelés «tokens ». La lemmatisation La lemmatisation permet à chaque mot des tokens d'être représenté sous forme canonique. La forme canonique d'un verbe est son infinitif , tandis que celle d'un nom est son masculin singulier. Cette substitution vise à ne conserver que le sens des mots. La racinisation (stemming) En NLP, on dispose d'une autre méthode de nettoyage des HAI606I Rapport - Classification de données textuelles... 4 représentés sous différents types, que ce soient des tableaux, des graphes, des arbres ou autres. Il existe également différents modèles, comme le dataset d'entraînement, le dataset de test et le dataset de validation.

..... Un algorithme d'apprentissage est un algorithme utilisé dans l'apprentissage automatique pour aider la technologie à imiter le processus d'apprentissage humain. Combinés à des technologies comme les réseaux de neurones, les algorithmes d'apprentissage créent des programmes d'apprentissage sophistiqués et impliqués. presentation de notre Dataset

Début de rédaction de l'axe //parler de dataset , rappeler qu'on la déjà définit //rappeler l'enjeu de l'usage du dataset en préambule // dire que l'enjeu (classification implique le traitement de données textuelles) /* utiliser ça comme perche pour : présenter le jeu de données (+image) -expliquer dans quel but on a besoin d'utiliser le NLP évoquer les grandes étapes du traitement de données textuelles : a)parler des pré-traitement que l'on met en paramètres lors de la création d'une instance de type tf-idf et countvectorizer (dire qu'ils sont dans my clean text , éventuellement afficher la fonction) b)-parler de la vectorization : expliquer l'intérêt puis présenter les deux méthodes sac de mots et tf-idf , dire que l'on a utilisé tf-idf

*/

présentation de notre dataset :