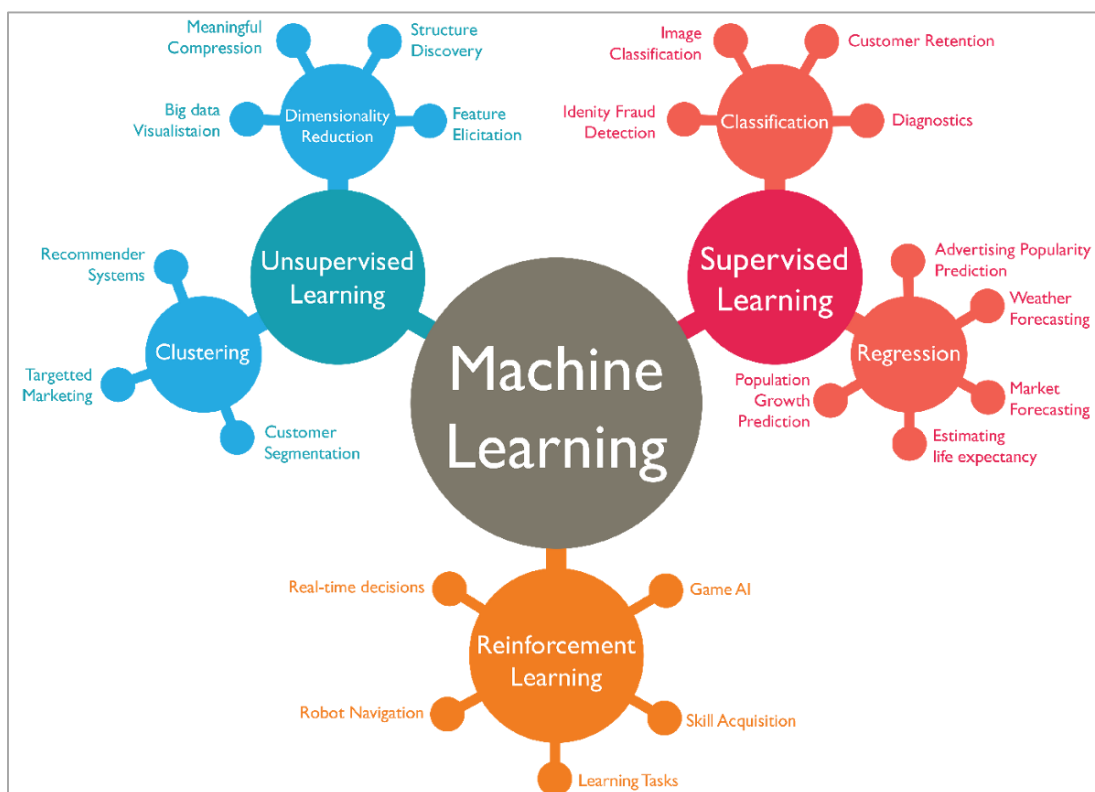


1^{ère} Année Master Informatique, Semestre 2

Module : TER

Étude d'un algorithme de Machine Learning



Rapport réalisé par :

SARAH OUHOCINE (AMIS)

Sujet proposé et encadré par :

M. LUC BOUGANIM

M. LUDOVIC JAVET

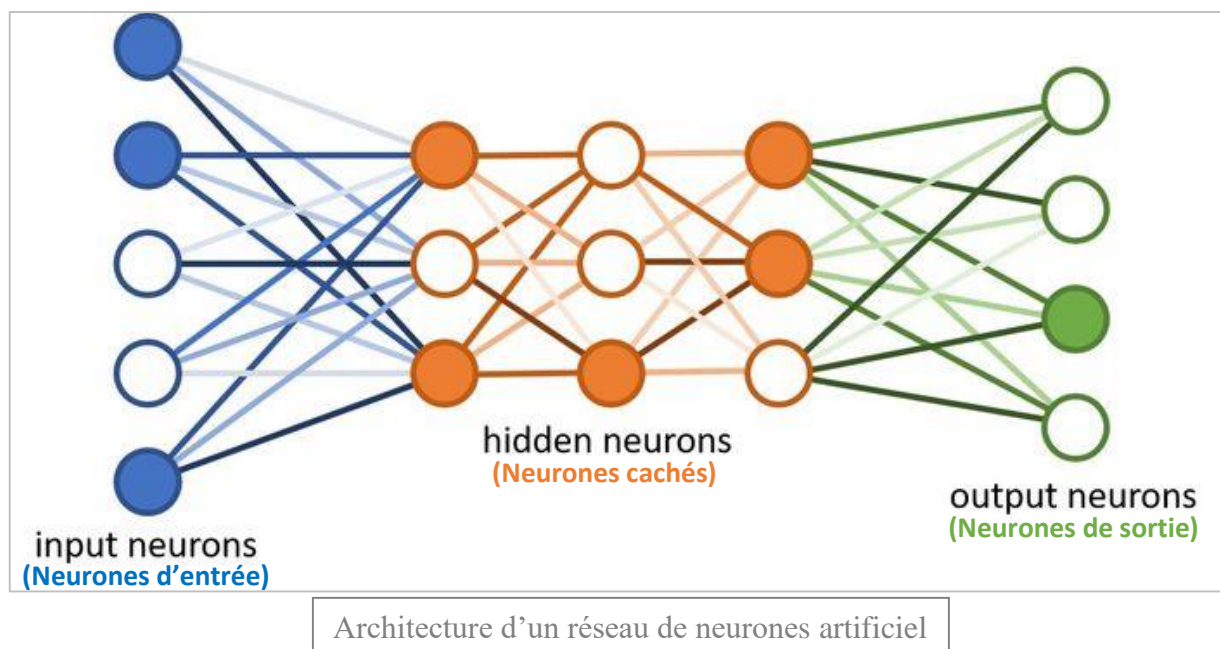
INTRODUCTION GENERALE :

Le [Machine Learning](#) (ML) est une branche de l'[Intelligence Artificielle](#) permettant aux ordinateurs d'apprendre à partir de données et d'exécuter, ainsi, plusieurs tâches sans être explicitement programmés pour les faire.

En se nourrissant de nouvelles données, les algorithmes de Machine Learning sont capables d'évoluer en permanence.

En effet, Il existe plusieurs algorithmes de Machine Learning, selon le mode d'apprentissage qu'ils emploient ([supervisé](#), [non supervisé](#), [semi-supervisé](#), [par renforcement](#), [par transfert](#) etc.)

Dans le cadre du projet TER, après avoir lu la feuille de route, nous avons opté pour l'algorithme « **les Réseaux de Neurones Artificiels** », en anglais « **Artificial neural networks (ANN)** » ; qui sont en fait, des systèmes informatiques inspirés des [réseaux de neurones biologiques](#) (Circuits neuronaux) qui constituent le cerveau des animaux.



L'objectif de ce TER est de réaliser une étude détaillée sur cet algorithme « **les Réseaux de Neurones** », (ci-après nommé **ALGO**), afin d'améliorer nos connaissances dans le domaine de Machine Learning.

Étape 1

1- Les liens vers les pages qui décrivent **ALGO** :



Scikit-learn :

- [Modèles de réseaux de neurones \(supervisés\).](#)
- [Modèles de réseaux de neurones \(non supervisés\).](#)



Wikipédia :

- [Réseaux de neurones artificiels.](#)

2- Motivation personnelle pour **ALGO** :

Après de longues recherches et différentes comparaisons entre les algorithmes de Machine Learning ; notre choix s'est porté sur **ALGO** pour les raisons suivantes :

- ✓ **ALGO** respecte rigoureusement les contraintes posées sur l'algorithme à choisir (feuille de route). En effet, **ALGO** est [un algorithme d'apprentissage supervisé](#) applicable aux problèmes de [Régression](#) et de [Classification](#). De plus, il est distribuable et adapté à [l'apprentissage semi-supervisé](#), plus exactement au « self-training » étant donné qu'il dispose de la méthode « [predict-proba](#) ».
- ✓ **ALGO** est un algorithme indépendant des données ([data agnostic algorithm](#)) qui nous donne la liberté de choisir n'importe quel ensemble de données. En effet, les réseaux de neurones avec l'apparition de plusieurs types tels que les [CNN](#) « Convolutional Neural Networks », les [RNN](#) « Recurrent Neural Networks » ...etc. brillent vraiment lorsqu'il s'agit de problèmes complexes tels que la classification d'images, le traitement du langage naturel, la reconnaissance vocale et bien plus encore.
- ✓ Les réseaux de neurones ont connu une explosion de popularité, notamment dans les dernières années grâce à leurs supériorités en termes de précision une fois entraînés avec une énorme quantité de données.

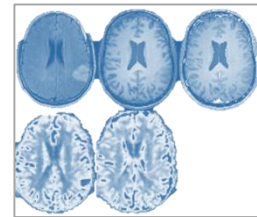
3- Scénarios pertinents pour ALGO :

Les cas d'usages des réseaux de neurones sont nombreux et se multiplient avec les avancées de l'intelligence artificielle. Parmi les exemples d'usages, on peut citer :

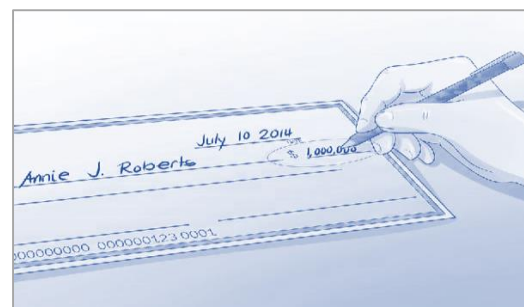
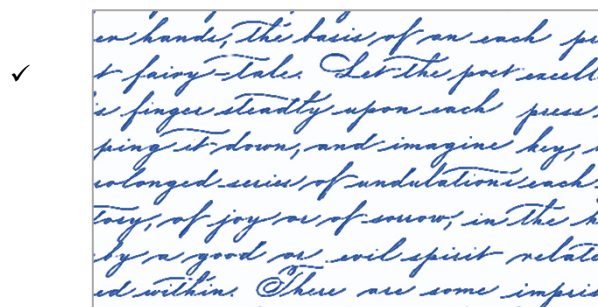
- ✓ La classification de textes et d'images.



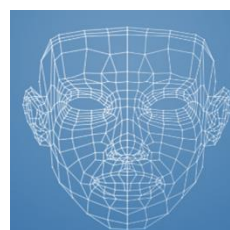
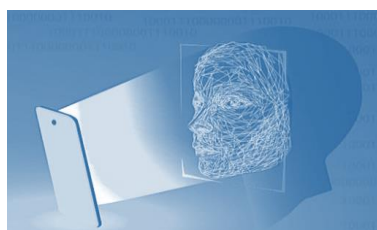
- ✓ La reconnaissance d'images, exemple : identifier et délimiter des tumeurs dans des images médicales.



- ✓ La reconnaissance d'écriture manuscrite, exemple : il est désormais possible de construire un système capable d'analyser le montant souscrit sur un chèque et reconnaître les nombres manuscrits.



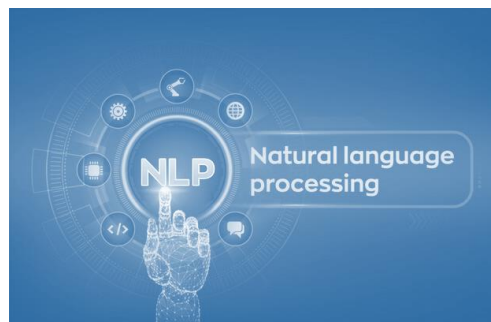
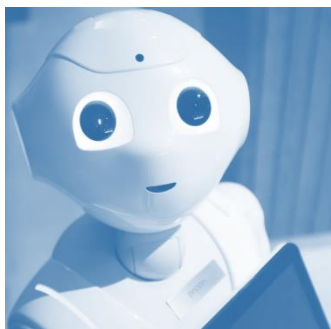
- ✓ [La reconnaissance faciale.](#)



- ✓ La prévision des marchés financiers, aussi appelée « [trading algorithmique](#) ».



- ✓ Entraînement des [chatbots](#) et des algorithmes de traitement naturel du langage ([NLP](#)).



- ✓ La prédiction météo.



- ✓ [L'analyse prédictive](#) en entreprises.



En résumé, les réseaux de neurones artificiels excellent pour la reconnaissance de patterns, le traitement de signaux complexes et la prédiction.

Étape 2

1- Jeux De Données (Datasets) :

1.1- Un Dataset, C'est Quoi ?

Une fois l'algorithme de Machine Learning choisi (**ALGO**), vient l'étape de choix de jeux de données (**DATASETS**). En effet, un dataset est un ensemble de données (**valeurs**) où chaque donnée est associée à un attribut (**variable**) ou à une observation.

- **Un attribut** : décrit l'ensemble des données ou valeurs décrivant la même variable.
- **Une observation** : contient l'ensemble des données ou valeurs décrivant les attributs d'une unité (ou individu statistique).
-

1.2- Les Différentes Structures D'un Dataset :

Les datasets regroupent un ensemble de données cohérents qui peuvent se présenter sous différents formats (textes, chiffres, images, vidéos etc...). Ainsi, ils peuvent avoir être représentés avec différentes structures, par exemple structure tabulaire ([fichier CSV](#)), structure d'arbre (fichier [XML](#) ou [JSON](#)), structure de graphe ([RDF](#)) etc...

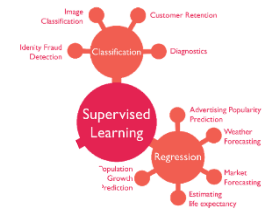
On travaille souvent avec **des structures tabulaires** dans les algorithmes de machine Learning.

1.3- Les Avantages Des Datasets :

Le premier prérequis pour créer un modèle de Machine Learning est d'avoir un dataset **annoté** (**description du contenu de chacune des données du dataset**). Sans données, il est impossible de créer ou de mesurer les performances d'un modèle d'apprentissage. Un modèle se construit selon les données que constitue un dataset, celui-ci doit se rapprocher au maximum des données que le modèle va traiter en production. Ainsi, les datasets nous permettent de pouvoir manipuler et effectuer différentes modifications sur les données.



‘‘APPRENTISSAGE SUPERVISÉ’’



*Dans cette deuxième étape, nous allons utiliser **scikit-learn** pour tester ALGO en apprentissage **supervisé** sur les jeux de données fournis avec **scikit-learn**.*

1.4- Les Datasets et l’Apprentissage Supervisé :

C’est une tâche d’apprentissage automatique qui consiste à apprendre une fonction de prédiction à partir d’exemples annotés. En effet, La machine reçoit des données caractérisées par des variables X (Features / Inputs) et annotées d’une variable Y (Target / Output / Label), dans le but que la machine apprenne à prédire la valeur de Y en fonction de X.

Nous distinguons des problèmes de régression, ainsi que des problèmes de classification.

1.5- Régression VS Classification :

La principale différence entre les algorithmes de régression et de classification est que les algorithmes de régression sont utilisés pour prédire les valeurs continues telles que le prix, le salaire, l’âge, etc... Alors que, les algorithmes de classification sont utilisés pour prédire ou classer les valeurs discrètes telles qu’Homme ou Femme, Vrai ou Faux, Spam ou pas de Spam, etc...

1.6- Datasets Choisis :

Les datasets de scikit-learn ont une structure tabulaire (fichiers CSV), i.e. les données sont représentées d’une façon telle que, chaque ligne correspond à une observation et chaque colonne à un attribut (variable).

scikit-learn présente plusieurs datasets ([datasets de jouets](#), [datasets du monde réel](#), [datasets générés](#), [autres datasets](#)) adaptés à la [régression](#) et à la [classification](#).

Nous avons choisi 2 datasets de jouets (**Toy datasets**) :

- ✓ **Pour le problème de Régression** : Nous avons choisi de tester notre ALGO sur le dataset « [Diabetes](#) ».

Il s’agit des données relatives à 442 patients diabétiques (observations / lignes), tels qu’un patient est caractérisé par 11 attributs (colonnes) :

- 10 attributs (features / inputs) : qui sont des valeurs prédictives numériques / réelles (âge, sexe, indice de masse corporelle, pression artérielle moyenne, et six mesures différentes de sérum sanguin).
- 1 attribut (target / output) : qui est une valeur entière entre 25 et 346, représentant une mesure quantitative de la progression de la maladie un an après la ligne de base.

✓ **Pour le problème de Classification** : Nous avons choisi de tester notre ALGO sur le dataset « [*Breast cancer wisconsin*](#) (diagnostic) ».

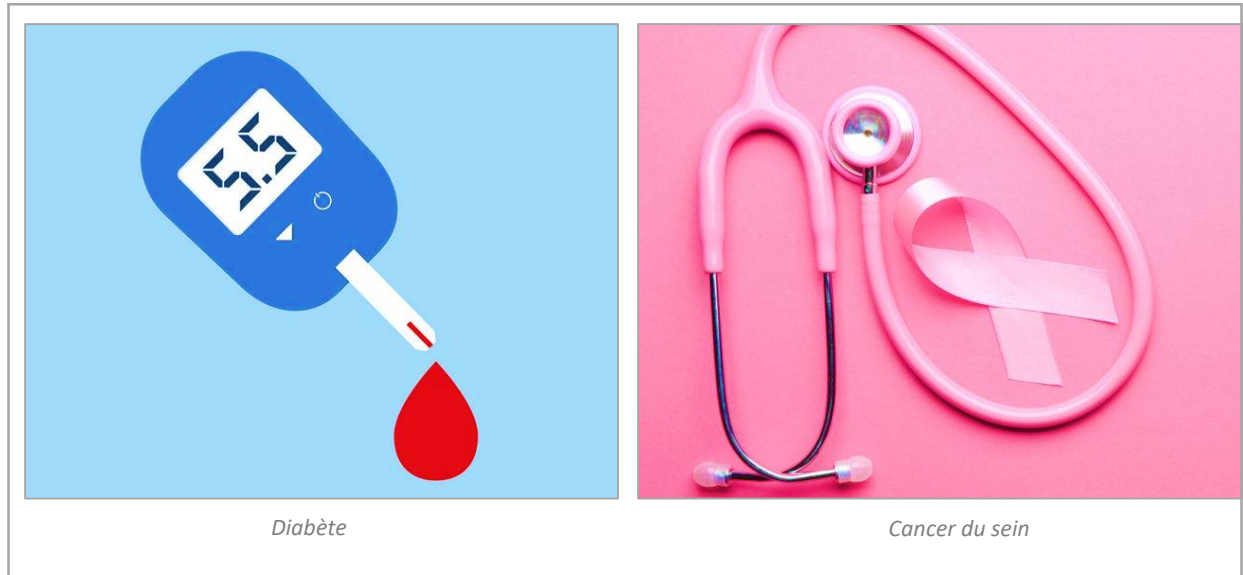
Il s'agit des données relatives à 569 images médicales de tumeurs / cancer (observations / lignes), tels que chaque tumeur est caractérisée par 31 attributs (colonnes) :

- 30 attributs (features / inputs) qui sont des valeurs numériques prédictives / réelles / positives (rayon, texture, périmètre, zone, douceur, compacité, concavité, points concaves, symétrie, dimension fractale, etc...).
- 1 attribut (target / output) qui est une valeur discrète représentant la classe dont la tumeur appartient (**Bénigne** ou **Maligne**).

1.7- Justification Des Choix Des Datasets :

- ✓ Les données des deux datasets ne nécessitent pas une étape de nettoyage (aucune valeur d'attribut manquante), ce qui va nous permettre d'accélérer (ou de sauter) l'étape de prétraitement des données.
- ✓ Les deux datasets sont d'une bonne taille (pas trop grands). De ce fait, ils sont utiles pour illustrer rapidement le comportement des différents algorithmes implémentés dans scikit-learn et ainsi, obtenir rapidement les résultats.
- ✓ Les deux datasets sont parfaits et optimisés pour scikit-learn. De ce fait, même les modèles de base donnent de bons résultats.
- ✓ Le thème des données des deux datasets sont approximativement proches « domaine médical », qui est un domaine passionnant, très intéressant et donne de plus en plus des résultats brillants avec l'évolution de l'Intelligence Artificielle et notamment, de la Machine Learning.
- ✓ Nous voulons travailler sur un même thème pour la régression ainsi que pour la classification. Cela va donner un sens à notre travail, mais aussi, va nous faciliter le choix des métriques et l'interprétation des résultats.

- ✓ Nous sommes curieux de savoir s'il y a des (faibles / moyennes / fortes) corrélations entre le diabète et le cancer du sein.



2- Implémentation d'un programme de test de base (paramètres par défaut) :

2.1- Classification :

Notre programme de test est basé sur les réseaux de neurones artificiels de type [Perceptron Multicouche](#) (Multi-Layer Perceptron – MLP). En effet, nous avons utilisé la fonction [MLPClassifier](#) définie dans le module `sklearn.neural_network`, qui fournit une implémentation du Perceptron Multicouche pour la classification.

MLPClassifier (la dernière version) comprend 23 paramètres à ajuster tels que le nombre de neurones pour chaque couche cachée (`hidden_layer_sizes`), la fonction d'activation (`activation`) et l'algorithme d'optimisation (`solver`) ...etc.

Notre Implémentation est une implémentation **de base**, i.e. les 23 paramètres ont été pris avec leur valeur par défaut dans `scikitlearn` (**paramètres par défaut**).
PS : l'explication des différents paramètres va venir dans l'étape 3.



2.1.1- Lien vers le code de base : [GitHub](#)

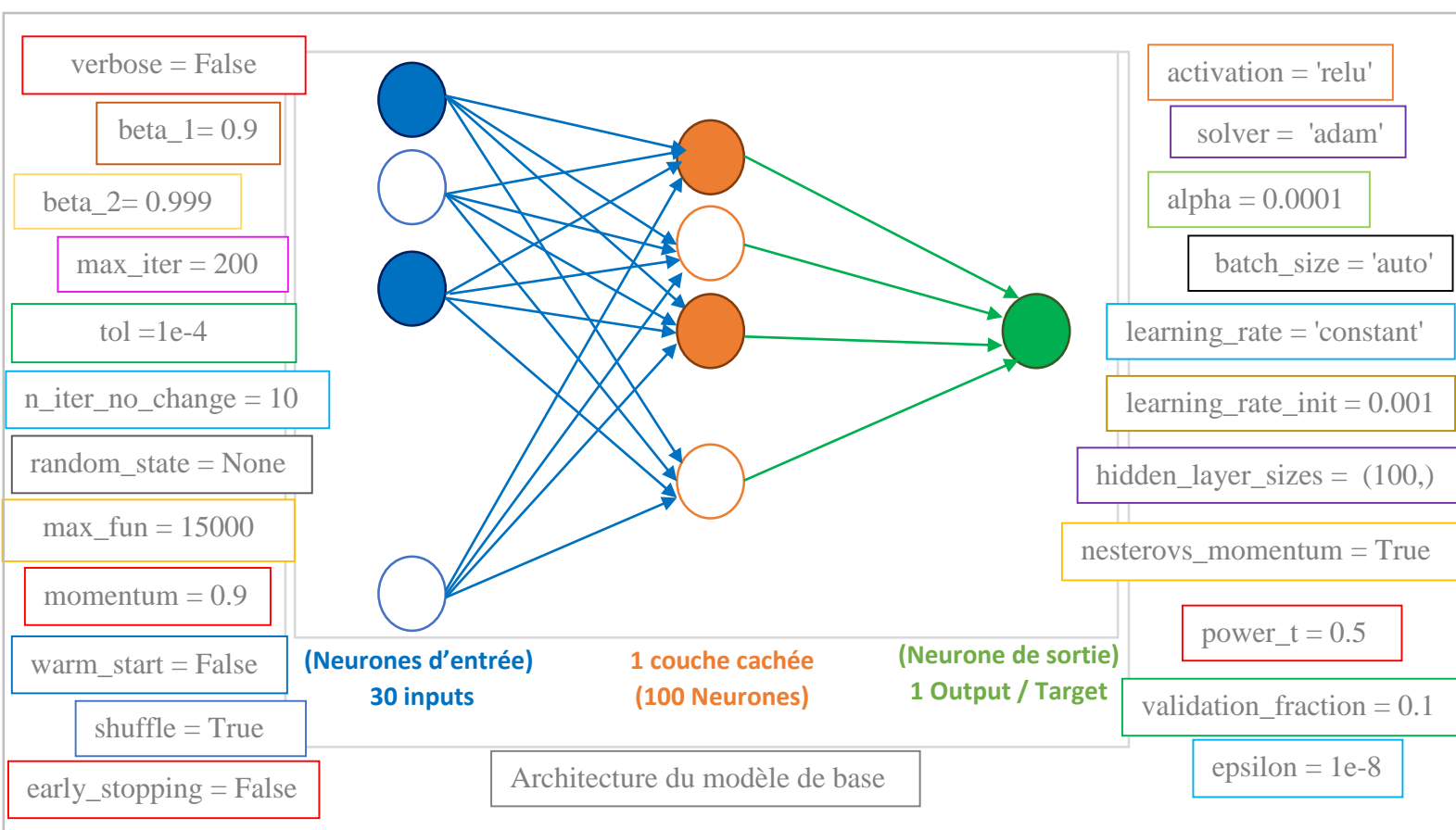
2.1.2- Préparation de données pour l'apprentissage :

Nous avons divisé le dataset « [Breast cancer wisconsin](#) » ayant 569 instances, adapté à la classification en 2 ensembles :

- ✓ Ensemble d'apprentissage (TRAINING) : contenant les données dédiées à l'apprentissage. Nous avons consacré 80% des données (i.e. 455 instances) du dataset pour cette partie-là.
- ✓ Ensemble de test (TEST) : contenant que 20% des données du dataset, vu que l'ensemble de données est petit (114 instances), mais c'est largement suffisant pour faire le test dans notre cas.

2.1.3- Architecture du modèle / Apprentissage :

Il s'agit d'un réseau de neurones « Perceptron Multicouche » de base ayant une seule couche cachée contenant 100 neurones et 22 autres paramètres (par défaut) que nous expierons plus tard.



La fonction d'apprentissage prend en entrée les paramètres du modèle décrit ci-dessus, ainsi que les données d'apprentissage et celles de test pour faire l'apprentissage. En effet, cette fonction peut calculer plusieurs métriques afin de pouvoir évaluer le modèle, et cela, en se basant sur [la matrice de confusion](#), aussi appelée tableau de contingence.

		État prévu	
		Positif (PP)	Négatif (PN)
État réel	Positif (P)	Vrai positif VP	Faux négatif FN
	Négatif (N)	Faux positif FP	Vrai négatif VN

Population totale = $P + N$

Matrice (2x2)

- VP (Vrai Positif) : requête affectée à la classe i et elle appartient à la classe i .
- VN (Vrai Négatif) : requête n'est pas affectée à la classe i et elle n'appartient pas à la classe i .
- FP (Faux Positif) : requête n'est pas affectée à la classe i et elle appartient à la classe i .
- FN (Faux Négatif) : requête affectée à la classe i et elle n'appartient pas à la classe i .

2.1.4- Interprétation des Résultats Obtenus :

		État prévu	
		Cancer	Non cancéreux
État réel	Cancer	40	1
	Non cancéreux	2	71

Total
41 + 73 = 114

Matrice de confusion :

```

[ [ 40  1 ]
  [  2 71 ] ]

```

Matrice (2x2)

- Sur les 42 échantillons avec cancer (40 + 2), le système a prédit que 2 étaient sans cancer.
- Sur les 72 échantillons sans cancer (2 + 71), il a prédit que 1 avait un cancer.
- Toutes les **prédictions correctes** sont situées dans la diagonale du tableau (surlignées en vert) → 40 + 71 = 111.
- Toutes les **prédictions incorrectes** sont situées dans la diagonale du tableau (surlignées en rouges) → 1 + 2 = 3. Il est donc facile d'inspecter visuellement le tableau pour les erreurs de prédiction, car les valeurs en dehors de la diagonale les représenteront. En additionnant les 2 lignes de la matrice de confusion,

- Le nombre total d'échantillons positifs (P) est donné par : $P = VP + FN = 40 + 1 = 41$.
- Le nombre total d'échantillons négatifs (N) est donné par : $N = FP + VN = 2 + 71 = 73$.
- Le nombre total d'échantillons est donné par : $P + N = 41 + 73 = 114$.

2.1.5- Métriques Choisies pour l'évaluation du modèle :

Pour mieux évaluer et apprécier la qualité du modèle, nous avons choisi 5 [métriques adaptées aux problèmes de classification](#)

✓ **Le taux d'erreur** ([zero one loss](#))

Le taux d'erreur ou perte ou ERR, c'est une métrique qui est calculée en faisant la somme de toutes les prédictions incorrectes sur le nombre total de données (positives et négatives).

$$\frac{FN + FP}{VP + FN + FP + VN}$$

Plus il est bas, mieux c'est. Le meilleur taux d'erreur possible est de 0, mais il est rarement atteint par un modèle dans la pratique.

Notre modèle a un taux d'erreur assez bas (0.026), on constate dans un premier temps que notre modèle de base est bon et performant.

✓ **Accuracy / Exactitude** ([accuracy score](#))

Ce paramètre fait la somme de tous les vrais positifs et vrais négatifs qu'il divise par le nombre total d'instances. Il permet d'apporter une réponse à la question suivante : de toutes les classes positives et négatives, combien parmi elles ont été prédites correctement ?

$$\frac{VP + VN}{VP + FN + FP + VN}$$

Il peut également être calculé avec la formule suivante : $1 - \text{ERR}$.

Plus il est élevé, mieux c'est. La meilleure accuracy possible est de 1, mais il est rarement atteint par un modèle dans la pratique.

Notre modèle a une accuracy assez élevée (0.974), on constate dans un second temps que notre modèle de base est bon et performant.

Cependant, cette métrique ne permet pas de déterminer le nombre de vrais positifs que le modèle peut identifier. Il cache donc certains détails nécessaires à l'appréciation de la performance du modèle. C'est là qu'entrent en jeu trois autres paramètres, à savoir **la précision**, **le rappel** et **la F-mesure**.

✓ **La précision** ([precision_score](#))

La précision indique le **rapport entre les prévisions positives correctes et le nombre total de prévisions positives**. Ce paramètre répond donc à la question suivante : sur tous les enregistrements positifs prédits, combien sont réellement positifs ?

$$\frac{\text{VP}}{\text{VP} + \text{FP}}$$

En résumé, il s'agit d'une métrique classique qui évalue à quel point le modèle est bon et fiable pour prédire les classes.

Plus il est élevé, mieux c'est. La meilleure précision possible est de 1, mais il est rarement atteint par un modèle dans la pratique.

Notre modèle a atteint une précision assez élevée (0.973), on confirme que notre modèle de base est bon et performant.

✓ **Le rappel** ([recall_score](#))

Le rappel est un paramètre qui permet de mesurer le nombre de prévisions positives correctes sur le nombre total de données positives. Il permet de répondre à la question suivante : sur tous les enregistrements positifs, combien ont été correctement prédits ?

$$\frac{\text{VP}}{\text{VP} + \text{FN}}$$

En résumé, il s'agit d'une métrique qui évalue la capacité du modèle à classer les échantillons correctement par rapport à tout l'ensemble de test.

Plus il est élevé, mieux c'est. Le meilleur rappel possible est de 1, mais il est rarement atteint par un modèle dans la pratique.

Notre modèle a atteint un rappel assez élevée (0.986), on confirme encore une fois que notre modèle de base est bon et performant.

✓ La F-mesure ([f1 score](#))

La F-mesure est la **moyenne harmonique de la précision et du rappel**. Il équivaut au double du produit de ces deux paramètres sur leur somme.

$$\frac{2 * \text{précision} * \text{rappel}}{\text{précision} + \text{rappel}}$$

Sa valeur est maximale lorsque le rappel et la précision sont équivalents.

En résumé, il s'agit d'une mesure qui combine la précision et le rappel. Elle essaye de donner un score plus global au modèle de classification, en prenant compte des résultats du rappel et de la précision.

Dans certains modèles, on cherche à connaître la distribution des faux positifs et des faux négatifs. La métrique F-mesure est alors utilisée pour évaluer la performance de l'algorithme. De même, il est particulièrement difficile de comparer deux modèles avec une faible précision et un rappel élevé. Le contraire est également vérifié. Dans ces conditions, la F-mesure permet de mesurer ces deux paramètres simultanément.

Plus il est élevé, mieux c'est. La meilleure F-mesure possible est de 1, mais il est rarement atteint par un modèle dans la pratique.

Notre modèle a atteint une F-mesure assez élevée (0.979). Maintenant nous sommes certains que notre modèle de base est bon et performant.

2.1.6- Conclusion :

En résumé, les paramètres par défaut nous ont permis de construire un bon modèle et d'obtenir de bons résultats malgré le peu de données que nous avons (petit dataset). Notre interprétation est que, les paramètres par défaut construisent de bons modèles et donnent de bons résultats lorsque nous utilisons les jeux de données de scikitlearn pour les problèmes de classification. Pour cette raison-là, que nous allons nous focaliser dans les étapes qui suivent sur d'autres jeux de données, mais aussi, sur l'ajustement des paramètres du modèle.

2.2- Régression :

Dans cette partie, nous allons nous concentrer sur les modèles de régression :

En effet, nous avons utilisé le `SGDRegressor` définie dans le module `sklearn`, qui effectue une régression linéaire. `SGDRegressor` une implémentation du Perceptron Multicouche pour la régression.

`MLPRegressor` (la dernière version), exactement comme le `MLPClassifier`, comprend 23 paramètres à ajuster tels que le nombre de neurones pour chaque couche cachée (`hidden_layer_sizes`), la fonction d'activation (`activation`) et l'algorithme d'optimisation (`solver`) ...etc.

Notre Implémentation est une implémentation **de base**, i.e. les 23 paramètres ont été pris avec leur valeur par défaut dans `scikitlearn`.

PS : l'explication des différents paramètres va venir dans l'étape 3.



2.2.1- Lien vers le code de base : [GitHub](#)

2.2.2- Préparation de données pour l'apprentissage :

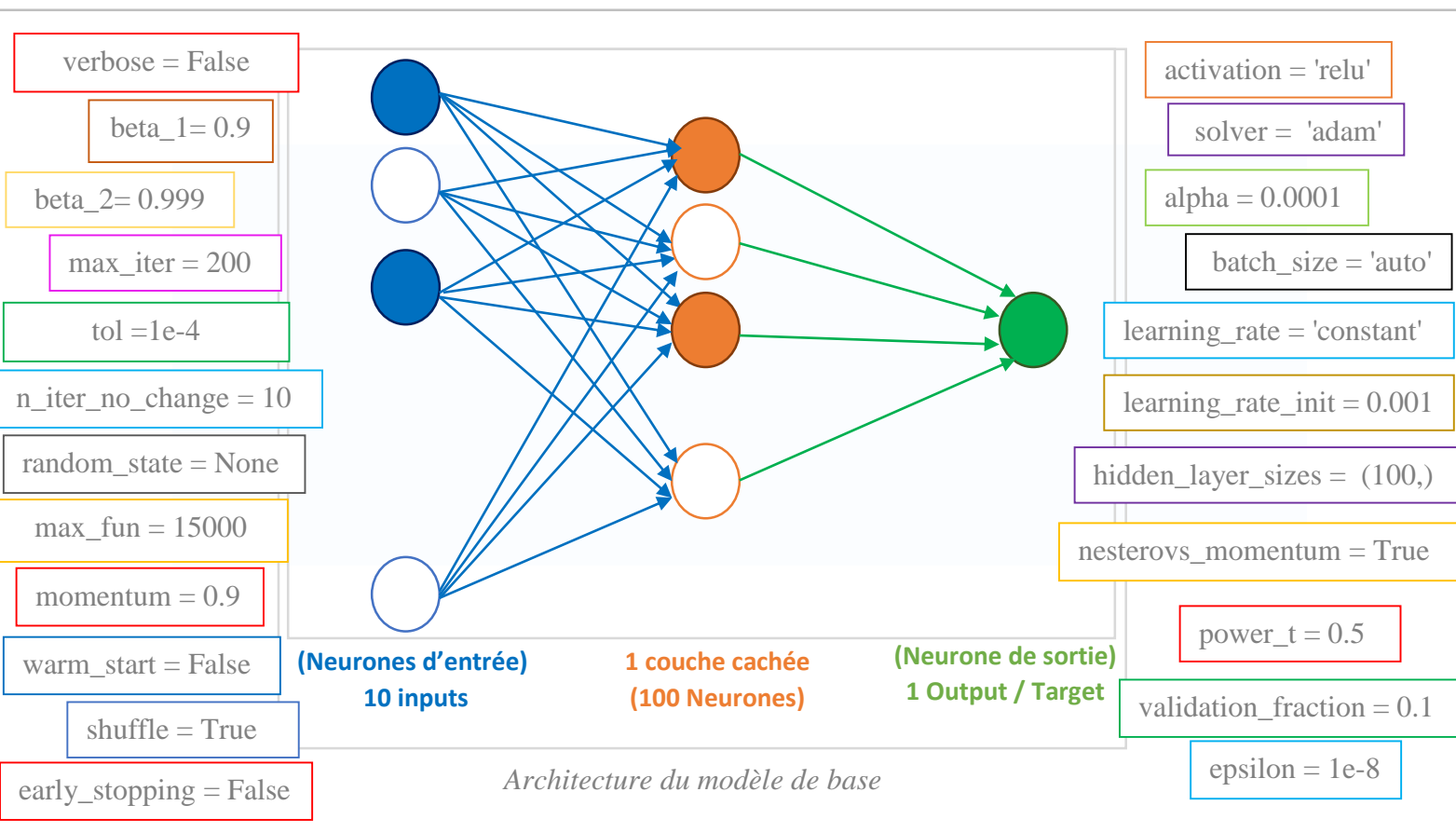
Nous avons divisé le dataset « [Diabetes](#) » ayant 442 instances, adapté à la régression en 2 ensembles :

- ✓ Ensemble d'apprentissage (TRAINING) : contenant les données dédiées à l'apprentissage. Nous avons consacré 80% des données (i.e. 353 instances) du dataset pour cette partie-là.
- ✓ Ensemble de test (TEST) : contenant que 20% des données du dataset, vu que l'ensemble de données est petit (89 instances), mais c'est largement suffisant pour faire le test dans notre cas.

2.2.3- Architecture du modèle / Apprentissage :

Il s'agit d'un réseau de neurones « Perceptron Multicouche » de base ayant une seule couche cachée contenant 100 neurones et 22 autres paramètres (par défaut) que nous expliquerons plus tard.

La fonction d'apprentissage prend en entrée les paramètres du modèle décrit ci-dessous, ainsi que les données d'apprentissage et celles de test pour faire l'apprentissage. En effet, cette fonction peut calculer plusieurs métriques afin de pouvoir évaluer le modèle.



2.2.4- Interprétation des Résultats Obtenus :

Dans un modèle de régression, la matrice de confusion ne sert à rien, En effet, le nombre de classes à prédire n'est pas déterminé, car les valeurs sont continues.

Les modèle issue du MLPRegressor pour chaque attribut du dataset, n'est ni une droite, ni une courbe issue d'une fonction polynomiale, mais bien un modèle complexe (Ce qui est cohérent pour les réseaux de neurones). De ce fait, nous allons nous contenter par interpréter les résultats directement à partir des valeurs obtenues des métriques dédiées à l'évaluation des modèles de régression.

En grosso modo, avant de passer aux métriques, on précise qu'un bon modèle donne pour chaque attribut du dataset, de petites erreurs entre les prédictions $f(X_i)$ et les vraies valeurs (Y_i) du dataset.

$$|f(x_i) - y_i|$$

I.e. le but est de tracer un modèle (de régression) qui s'insère bien dans le nuage de points associé au jeu de données pour chaque attribut du dataset.

2.2.5- Métriques Choisies pour l'évaluation du modèle :

Pour évaluer un modèle de régression, on calcule la distance entre valeurs prédites et vraies valeurs.

✓ **la somme des carrés des résidus / RSS** ([Residual Sum of Squares](#)) :

Il s'agit de calculer pour chaque point X_i du dataset de test, la distance entre sa vraie valeur et la valeur prédite et en faire la somme.

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

Puis, on calcule la **corrélation** entre les valeurs prédites et les vraies valeurs, pour faciliter l'interprétation des résultats.

Notre modèle a atteint une $RSS = 964060.62$.

✓ **l'Erreur Carrée Relative / RSE** ([Relative Squared Error](#))

Il s'agit de normaliser la RSS par la somme des distances entre chacune des valeurs à prédire et leur moyenne.

$$RSE = \frac{\sum_{i=1}^n (y_i - f(x_i))^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad \leftarrow \text{RSS}$$
$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

Plus il est bas, mieux c'est. La meilleure RSE est de 0, mais elle est rarement atteinte par un modèle dans la pratique.

Notre modèle a une RSE assez élevée (1.80), on constate dans un premier temps que notre modèle de base n'est pas bon et n'est pas performant.

✓ Coefficient de Détermination ([r2_score](#))

Le coefficient de détermination est le carré de la corrélation de Pearson (R) entre les valeurs prédites et vraies valeurs. Il se calcule en complétant la RSE à 1

$$R^2 = 1 - RSE$$

Le coefficient de détermination est un indicateur qui mesure la qualité d'une prédiction. Plus cet indicateur est proche de 1, plus le modèle est proche de la réalité et les valeurs prédites sont corrélées aux vraies valeurs. Le meilleur coefficient de détermination possible est de 1, mais il est rarement atteint par un modèle dans la pratique.

Il peut être **négatif** (car le modèle peut être arbitrairement pire, lorsqu'il est très loin de la réalité).

Notre modèle a un coefficient de détermination (R^2) **négatif** = -0,80. En effet, notre modèle de base est très loin du modèle réel, et les valeurs prédites ne sont pas corrélées aux vraies valeurs. Nous constatons dans un second temps que notre modèle de base n'est pas bon et n'est pas performant.

D'après ces métriques, nous pouvons dire que notre modèle de régression de base (avec les paramètres par défauts) ne s'insère pas bien dans le nuage de point pour les différents attributs du dataset. Ce qui veut dire qu'il y a de grandes marges d'erreurs entre les prédictions $f(X_i)$ et les vraies valeurs (Y_i) du dataset. De ce fait, nous constatons que notre modèle est très loin de la réalité (loin des vraies valeurs et représente mal les données).

2.2.6- Conclusion

En résumé, contrairement à la classification. Dans la régression, les paramètres par défaut ne nous ont pas permis de construire un bon modèle et d'obtenir de bons résultats malgré que le

jeu de données utilisé est un jeu de données de Sickitlearn (Diabetes). Notre interprétation est que, la régression nécessite beaucoup plus de données pour avoir un bon modèle et obtenir de bons résultats. Pour cette raison-là, que nous allons nous focaliser dans les étapes qui suivent sur d'autres jeux de données (plus grands), mais aussi, sur l'ajustement des paramètres des deux modèles pour essayer de les améliorer, notamment celui de la régression.

Étape 3


Les modèles de MLP (MLPClassifier / MLPRegressor) ont de nombreux paramètres (23) et trouver la meilleure combinaison de paramètres (meilleur modèle / **paramètres optimaux**) peut être traité comme un problème de recherche.


Bien qu'il existe maintenant de nombreux [algorithmes d'optimisation et de réglage des paramètres](#), nous avons choisi une stratégie simple qui est la recherche exhaustive sur la grille. En effet, dans scikit learn, il existe la méthode [GridSearchCV](#) qui trouve facilement les paramètres optimaux parmi les valeurs données.

Nous avons donc exploité cette méthode pour faire un programme automatique variant différentes valeurs pour les 23 paramètres de ALGO (notre espace de recherche), pour régler les paramètres et ajuster les modèles pour différentes combinaisons de paramètres et de donner la meilleure combinaison en **fonction des précisions, rappels, f-mesures, taux d'erreurs et exactitudes**. Et ainsi, observer les effets (ou les non-effets) de la modification de ces paramètres.

Nous avons choisi cette méthode, car elle est très répandue, efficace, et donne des résultats rapidement (gain du temps au lieu de varier les paramètres manuellement).

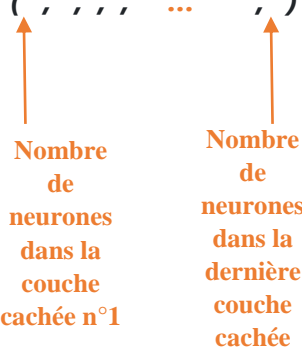
1- Les liens vers le code

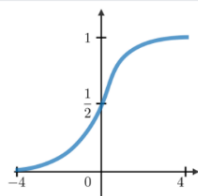
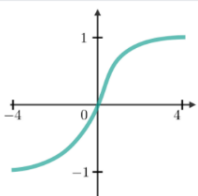
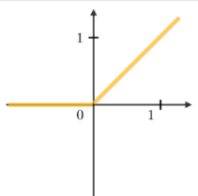
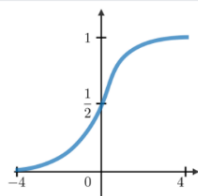
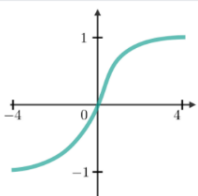
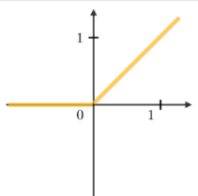
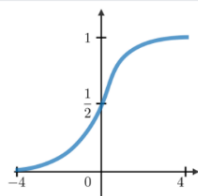
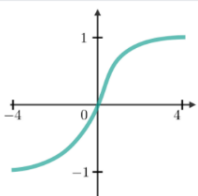
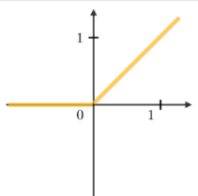
1.1- Classification :  [GitHub](#)

1.2- Régression :  [GitHub](#)

PS : nous avons varié plusieurs valeurs pour les 23 paramètres de ALGO, mais nous avons supprimées quelques valeurs testées et nous n'avons laissé dans notre code que les valeurs qui ont un effet sur ALGO (résultats acceptables), car plus on rajoute des valeurs, plus on aura de combinaisons, plus le temps d'exécution de ALGO augmente.

2- Description et explication des paramètres

Paramètre	Description / Explication	Efficace pour le solveur		
		adam	sgd	lbfgs
hidden_layer_sizes = (, , , , ... ,) 	<p>Ce paramètre, permet de spécifier le nombre de couches cachées ainsi que le nombre de neurones que nous voulons avoir dans chaque couche cachée.</p> <p>Dans un réseau de neurones, une couche cachée est une couche entre les couches d'entrée et les couches de sortie, où les neurones prennent un ensemble d'entrées pondérées et produisent une sortie via une fonction d'activation.</p>	X	X	X
activation	<p>La fonction d'activation est spécifique à chaque couche cachée. Elle sert à introduire une complexité non-linéaire au modèle et à transformer les données, autrement dit, elle modifie de manière non-linéaire les données. Cette non-linéarité permet de modifier spatialement leur représentation (changer notre manière de voir les données).</p> <p>i.e. Un modèle étant composé de multiples couches cachées, et donc de multiples fonctions d'activation ; rend la représentation des données complexe et</p>	X	X	X

	changeable successivement. Ce qui nous permet d'avoir un nouveau point de vue sur nos données.												
<div><div>identity →</div><div>logistic →</div><div>tanh →</div><div>relu →</div></div>	<p>Les 4 fonctions d'activation de MLP sont :</p> <ul style="list-style-type: none">✓ activation sans opération : $f(x) = x$✓ sigmoïde logistique : $f(x) = 1 / 1+\exp^{-x}$✓ tan hyperbolique : $f(x) = \tanh(x)$✓ unité linéaire rectifiée : $f(x) = \max (0, x)$ <table><tr><th>Sigmoïde</th><th>Tanh</th><th>ReLU</th></tr><tr><td>$g(z) = \frac{1}{1 + e^{-z}}$</td><td>$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$</td><td>$g(z) = \max(0, z)$</td></tr><tr><td></td><td></td><td></td></tr></table>	Sigmoïde	Tanh	ReLU	$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$						
Sigmoïde	Tanh	ReLU											
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$											
													
solver	<p>Le solveur ou l'optimiseur.</p> <p>Il joue le rôle de corriger / optimiser l'algorithme (corriger les poids d'un perceptron), pour que la fois suivante, il prédise une valeur plus proche de la valeur réelle. Autrement dit, le solveur diminue / optimise la fonction d'erreur (loss function) ; dans le but d'améliorer les prédictions et la stabilité du modèle construit auparavant.</p> <p>Les 3 solveurs de MLP sont :</p> <ul style="list-style-type: none">✓ lbfgs : est un optimiseur de la famille des méthodes quasi-newton. Il converge plus rapidement et fonctionne mieux sur des petits datasets.✓ sgd : fait référence à la descente du gradient stochastique.✓ adam : fait référence à un optimiseur basé sur un gradient stochastique proposé Kingma, Diederik et Jimmy Ba. En terme de temps d'apprentissage												

	et de score de validation, il fonctionne assez bien sur des grands datasets (1000 instances et plus).			
alpha	<p>Un terme de régularisation</p> <p>La Régularisation est un ensemble de méthode qui permet à la fois d'optimiser l'apprentissage d'un modèle de réseaux de neurones et d'empêcher l'overfitting (manque de généralisation).</p> <p>En effet, lors de l'apprentissage, certains poids du modèle augmentent tandis que d'autres diminuent. MLP utilise le paramètre L2, qui est un outil pour faire la régularisation en diminuant les valeurs des poids trop important, permettant la généralisation lors de l'apprentissage du modèle. Ainsi, le modèle va être performant à la fois sur les données d'entraînements mais aussi sur toutes les autres données.</p>	X	X	X
batch_size	<p>La taille du lot définit le nombre d'échantillons (instances) qui seront propagés sur le réseau.</p> <p>La taille du lot est inférieure au nombre de tous les échantillons, ainsi, l'Avantages de l'utiliser est qu'il nécessite moins de mémoire. Étant donné que nous formons le réseau en utilisant moins d'échantillons, la procédure de formation globale nécessite moins de mémoire. C'est important si nous ne sommes pas en mesure de faire tenir l'ensemble de données dans la mémoire de notre machine.</p> <p>PS : les réseaux s'entraînent plus rapidement avec des mini-lots. C'est parce que nous mettons à jour les poids après chaque propagation.</p>	X	X	

learning_rate	<p>Le taux d'apprentissage. Il indique la vitesse à laquelle les poids évoluent (mis à jour). Ce taux peut être fixe ou variable.</p> <p>PS : Adam est l'une des méthodes les plus populaires à l'heure actuelle qui a un taux d'apprentissage qui s'adapte au fil du temps.</p> <p>Les 3 types de learning_rate de MLP sont :</p> <ul style="list-style-type: none"> ✓ constant : taux d'apprentissage constant donné par learning_rate_init $\text{learning_rate} = \text{learning_rate_init}$ ✓ invscaling : diminue progressivement le taux d'apprentissage à chaque pas de temps 't' en utilisant 'power_t' $\text{learning_rate} = \text{learning_rate_init} / t^{\text{power_t}}$ ✓ adaptative : <div> <p><u>SI</u> la perte continue à se diminuer :</p> $\text{learning_rate} = \text{learning_rate_init}$ <p><u>SINON</u> si (la perte ne se diminue pas d'au moins 'tol') <u>ou</u> (le score de validation n'augmente pas d'au moins 'tol') au bout de 2 époques consécutives avec 'early_stopping' activée</p> $\text{learning_rate} = \text{learning_rate_init} / 5$ </div> 		X	
learning_rate_init	Le taux d'apprentissage initial. Il contrôle la taille du pas dans la mise à jour des poids.	X	X	
power_t	L'exposant du taux d'apprentissage de mise à l'échelle inverse. Il est utilisé pour mettre à jour le taux d'apprentissage lorsque learning_rate est défini sur 'invscaling'.		X	
max_iter	C'est le nombre maximal d'itérations. En effet, le solveur itère jusqu'à :	X	X	X

	<ul style="list-style-type: none"> - la convergence (déterminée par 'tol'), ou - ce nombre d'itérations. <p>Ce paramètre détermine pour les solveurs :</p> <ul style="list-style-type: none"> ✓ lbfgs : le nombre d'étapes de gradient. ✓ Adam / sgd : le nombre d'époques (combien de fois chaque point de données sera utilisé) 			
shuffle	Détermine s'il faut mélanger les échantillons à chaque itération.	X	X	
random_state	<p>Génère des random (nombres aléatoires) pour :</p> <ul style="list-style-type: none"> - l'initialisation du biais et des poids. - La division en train/test si 'early_stopping' est utilisée. - 'batch_size' lorsque solver = 'sgd' ou 'adam'. 	X	X	X
tol	<p>Tolérance pour l'optimisation. 'tol' est aussi un paramètre qui aide à déterminer si la convergence est atteinte.</p> <p>Si learning_rate est définie sur :</p> <ul style="list-style-type: none"> ✓ constant / invscaling : lorsque la perte ou le score ne s'améliore pas d'au moins 'tol' pour 'n_iter_no_change' itérations consécutives la convergence est considérée comme atteinte et l'apprentissage s'arrête. 	X	X	X
verbose	Détermine s'il faut afficher les messages de progression relatifs à l'apprentissage (par exemple si l'apprentissage s'est arrêté à cause de 'tol' ou 'max_iter').	X	X	X
warm_start	Détermine s'il faut réutiliser la solution précédente comme initialisation (solutions dépendantes), ou bien, effacer simplement la solution précédente.	X	X	X

momentum	<p>Momentum ou l'élan. C'est un coefficient entre 0 et 1 appliqué dans la mise à jour des poids. En effet, Au lieu de dépendre uniquement du gradient actuel pour mettre à jour le poids, la descente de gradient avec momentum remplace le gradient actuel par momentum, qui est un agrégat de gradients. Cet agrégat est la moyenne mobile exponentielle des gradients actuels et passés (c'est-à-dire jusqu'au temps t)</p> <p>L'élan est connu pour accélérer l'apprentissage et aider à ne pas rester coincé dans les minimums locaux.</p> $\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$ <p style="text-align: center;"> weight increment learning rate weight gradient </p> $\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$ <p style="text-align: center;"> momentum factor weight increment, previous iteration </p>		X	
nesterovs_momentum	l'élan de Nesterov, utilisé uniquement lorsque 'momentum' > 0 et solveur = sgd.		X	
early_stopping	<p>Détermine s'il faut utiliser l'arrêt anticipé pour mettre fin à la. S'il est définie sur :</p> <ul style="list-style-type: none"> ✓ Vrai : mettra fin à l'apprentissage lorsque le score de validation ne s'améliore pas d'au moins 'tol' pour 'n_iter_no_change' époques consécutives. ✓ Faux : l'apprentissage s'arrête lorsque la perte d'apprentissage ne s'améliore pas d'au moins 'tol' pour 'n_iter_no_change' époques consécutives sur l'ensemble d'apprentissage. 	X	X	
validation_fraction	La proportion de données d'apprentissage à mettre de côté en tant qu'ensemble de validation pour			

	'early_stopping'. Elle doit être comprise entre 0 et 1, et utilisée uniquement si early_stopping vaut Vrai.	X	X	
beta_1	Les taux de décroissances exponentielles pour les estimations du vecteur du premier et du deuxième moment dans adam. En effet, Au lieu d'adapter les taux d'apprentissage des paramètres en fonction du premier moment moyen (la moyenne), Adam utilise également la moyenne des seconds moments des gradients (la variance non centrée). Plus précisément, l'algorithme calcule une moyenne mobile exponentielle du gradient et du gradient au carré, et les paramètres beta1 et beta2 contrôlent les taux de décroissance de ces moyennes mobiles.	X		
beta_2				
epsilon	La Valeur de stabilité numérique dans adam	X		
n_iter_no_change	Le nombre maximql d'époques pour ne pas avoir 'tol' amélioration.	X	X	
max_fun	C'est le nombre maximal d'appels de fonction de perte. En effet, Le solveur itère jusqu'à atteindre : ✓ la convergence (déterminée par 'tol'), ou ✓ le nombre max d'itérations 'max_iter', ou ✓ ce nombre d'appels de fonction de perte.			X

3- Discussion des variations et des nouveaux résultats obtenus

En variant différentes valeurs pour les paramètres pour MLPClassifier et MLPRegressor, nous avons observé de remarquables effets lors de la modification de 4 paramètres : **activation**, **solver**, **learning_rate** et **max_iter** (augmentation). En effet, la modification de ces 4 paramètres nous a permis d'améliorer nos deux modèles (classification et régression) obtenus dans l'étape 2.

PS : Les modèles de régression restent plus difficiles à améliorer par rapport aux modèles de classification. Et cela car le nombre des classes à prédire n'est pas déterminé, étant donné que les valeurs sont continues

En revanche, la variation des valeurs des 19 autres paramètres restants ne nous a permis d'observer aucun effet particulier. Ce que nous avons constaté, c'est que les autres paramètres maintenus à leur valeur par défaut, donnent bien de bons résultats.

Un autre point important que nous ayons constaté, c'est que l'implémentation d'un algorithme de **Deep Learning** (augmentation du nombre de couches cachées et des neurones dans chaque couche cachée) dans notre cas n'améliore pas les 2 modèles, au contraire il les empire. Selon nous, cela est dû à la taille des datasets (petits datasets). De ce fait, nous allons utiliser dans l'étape prochaine des datasets plus grands pour ré-observer les effets et les non-effets de la modification de ces 23 paramètres.

3.1- Nouveau modèle et résultat pour la Classification

Le modèle :

```
{'activation': 'logistic', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-08, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_iter': 3000, 'momentum': 0.9, 'n_iter_no_change': 10, 'power_t': 0.5, 'solver': 'lbfgs', 'tol': 0.0001, 'validation_fraction': 0.1, 'warm_start': 'False'}
```

Nous a permis d'améliorer le modèle de classification par défaut de : $1 - 0,974 = 0.026$

3.2- Nouveau modèle et résultat pour la Regression

Le modèle :

```
{'activation': 'tanh', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-08, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.001, 'max_iter': 3000, 'momentum': 0.9, 'n_iter_no_change': 10, 'power_t': 0.5, 'solver': 'sgd', 'tol': 0.0001, 'validation_fraction': 0.1, 'warm_start': 'False'}
```

Nous a permis d'améliorer le modèle de régression par défaut de : $0.61 - (-0,80) = 1.41$

Conclusion

Les deux modèles Ci-dessus sont les deux meilleurs modèles obtenus jusqu'à maintenant. Mais, il se peut que d'autres architectures (modèles) différentes, donnent des résultats identiques aux nôtres, ou encore des résultats meilleurs. I.e. Nos 2 modèles (classification et régression) re

stent améliorables et perfectibles, en essayant par exemple d'autres valeurs (que nous n'avons pas testées). Notamment celui de la régression.

PS : avant de conclure cette étape, visualisons d'abord les effets de variation de chaque paramètre (que nous avons jugé comme pertinent : activation, solver, learning_rate, max_iter) indépendamment des autres paramètres sur la métrique [LOSS](#) (fonction de perte) tels que plus le résultat de cette fonction est minimisé, plus le réseau de neurones est performant. Et cela pour chacun des deux datasets : de classification (breast_cancer) et de la régression (diabetes).

[VISUALISATION DES GRAPHS](#)

Nous rappelons que ces modèles, sont inclus dans l'ensemble des modèles retournés par GridSearchCV (car ce dernier retourne tous les modèles possibles).

On remarque que les valeurs des paramètres qui minimisent plus la LOSS pour la classification, ainsi, que pour la régression correspondent bien aux valeurs des paramètres des deux meilleurs modèles sélectionnés par la méthode GridSearchCV.

Classification :

```
{'activation': 'logistic', 'learning_rate': 'constant', 'max_iter': 3000, 'solver': 'lbfgs'}
```

Regression :

```
{'activation': 'tanh', 'learning_rate': 'adaptative', 'max_iter': 3000, 'solver': 'sgd'}
```

Étape 4

Dans cette étape nous allons chercher des datasets hors ceux proposés par scikit-learn, afin de les analyser, les nettoyer en cas de besoin.

1. Dataset de Classification :

1.1- Lien vers le dataset : [Dataset de la classification](#)

1.2- Lien vers l'analyse du dataset : [Analyse du dataset de la classification](#)

Après plusieurs recherches nous avons décidé de travailler sur l'un des datasets proposés par le site de [« SuperAnnotate »](#), ce dernier propose plus de 75 datasets dans divers sujets.

Le dataset choisi est : [« The Fetal Heath Classification »](#). Il est crée le 07 septembre 2010, et il vise à classer la santé d'un fœtus comme normale, suspecte ou pathologique à partir des données d'examens de cardiotocogrammes (CTG).



Figure : Fœtus

Le CTG est un appareil qui a pour but d'enregistrer les bruits du cœur d'un bébé et de mesurer les contractions sur un temps donné.

Il s'agit des données relatives à 2126 examens de CTG (observations / lignes), tels que chaque examen est caractérisée par 22 attributs (colonnes) :

- 21 attributs (features / inputs) :
 - **'Baseline value'** Valeurs de fréquence cardiaque foetale de base (FHR).
 - **'accelerations'** Nombre d'accélération par seconde.
 - **'fetal_movement'** Nombre de mouvements du fœtus par seconde.
 - **'uterine_contractions'** Nombre de contractions utérines par seconde.
 - **'light_decelerations'** Nombre de décélérations légères par seconde.
 - **'severe_decelerations'** Nombre de décélérations sévères par seconde.
 - **'prolongued_decelerations'** Nombre de décélérations prolongées par seconde.
 - **'abnormal_short_term_variability'** Pourcentage de temps avec une variabilité anormale à court terme.
 - **'mean_value_of_short_term_variability'** Valeur moyenne de la variabilité à court terme.
 - **'percentage_of_time_with_abnormal_long_term_variability'** Pourcentage de temps avec une variabilité anormale à long terme.
 - **'mean_value_of_long_term_variability'** Valeur moyenne de la variabilité à long terme.
 - **'histogram_width'** Largeur de l'histogramme.
 - **'histogram_min'** Minimum (basse fréquence) de L'histogramme.
 - **'histogram_max'** Maximum (haute fréquence) de l'histogramme FHR.
 - **'histogram_number_of_peaks'** Nombre de pics d'histogramme.
 - **'histogram_number_of_zeroes'** Nombre de zéros d'histogramme.
 - **'histogram_mode'** Mode histogramme.
 - **'histogram_mean'** Moyenne de l'histogramme.

- **'histogram_median'** Médiane de l'histogramme.
- **'histogram_variance'** Écart d'histogramme.
- **'histogram_tendency'** Tendance de l'histogramme.
- 1 attribut (target / output) :
 - **'fetal_health'** : 1 (Normal), 2 (Suspect) et 3 (Pathologique)

1.3- Justification du choix du dataset :

Notre choix se justifie par le fait que :

- ✓ Nous voulions rester dans le même thème (domaine médical) que celui des datasets précédents (étape 2 et 3). donc nos recherches étaient orientées vers ce dernier.
- ✓ Le dataset traite un sujet pas très courant mais intéressant.
- ✓ La taille du dataset est importante : le nombre d'instances est suffisant pour faire un bon apprentissage (2126 instances), ainsi que le nombre des features (22).
- ✓ Les données du dataset ne nécessitent pas une étape de nettoyage (aucune valeur d'attribut manquante d'après l'analyse), ce qui va nous permettre d'accélérer le travail en passant directement à l'étape suivante (apprentissage).

On peut estimer que le dataset donnera de bons résultats après l'apprentissage étant donné que ses données sont de bonne qualité, il n'a pas de valeurs manquantes. Mais aussi, d'après l'analyse du dataset, tous les attributs sont pertinents (i.e. les valeurs de tous les attributs sont significative), ce qui fait dans l'étape 5 on va prendre en compte tous les attributs du dataset pour faire l'apprentissage

2. Dataset de Régression :

2.1- Lien vers le dataset : [Dataset de la regression](#)

2.2- Lien vers l'analyse du dataset : [Analyse du dataset de la régression](#)

Après plusieurs recherches nous avons décidé de travailler sur l'un des datasets proposés par le site « [kaggle](#) », qui est une plateforme web organisant des compétitions en science des données proposées par des entreprises appartenant à Google. ce dernier propose plus de plus de 50.000 ensembles de données publics dans divers sujets.

Pour la régression nous avons choisi le dataset « [Medical Cost Personal](#) », qui consiste à prédire le coût de l'assurance maladie via des données collectées par des experts.

Il s'agit des données relatives à 1338 patients (observations / lignes), tels que chaque patient est caractérisée par 7 attributs (colonnes) :



Figure : Assurance maladie

- 6 attributs (features / inputs) :
 - ✓ **Âge** : âge du principal bénéficiaire
 - ✓ **Sexe** : sexe de l'assureur, féminin, masculin
 - ✓ **BMI** : indice de masse corporelle, permettant de comprendre le corps, les poids relativement élevés ou faibles par rapport à la taille, indice objectif de poids corporel (kg/m^2) utilisant le rapport taille sur poids, idéalement 18,5 à 24,9
 - ✓ **Enfants** : Nombre d'enfants couverts par l'assurance maladie / Nombre de personnes à charge
 - ✓ **Fumeur** : Fumer
 - ✓ **Région** : zone de résidence du bénéficiaire aux États-Unis, nord-est, sud-est, sud-ouest, nord-ouest.
- 1 attribut (target / output) :
 - ✓ **Charges** : Frais médicaux individuels facturés par l'assurance maladie.

2.3- Justification du choix du dataset :

- ✓ Nous voulions rester dans le même thème (domaine médical) ..
- ✓ La taille du dataset est importante : le nombre d'instances est suffisant pour faire un bon apprentissage (1338 instances), ainsi que le nombre des features (7).
- ✓ Les données du dataset ne nécessitent pas une étape de nettoyage (aucune valeur d'attribut manquante d'après l'analyse), ce qui va nous permettre d'accélérer le travail en passant directement à l'étape suivante (apprentissage).

On peut estimer que le dataset donnera de bons résultats après l'apprentissage étant donné que ses données sont de bonne qualité, il n'a pas de valeurs manquantes. Mais aussi, d'après l'analyse du dataset, ce n'est pas tous les attributs qui sont pertinents (i.e. les valeurs de certains attributs tels que sex, smoker, region ne sont pas significatives), ce qui fait dans l'étape 5 on ne

va pas prendre en compte ces 3 attributs cités du dataset pour ne pas fausser le résultat de l'apprentissage.

Étape 5

PS : d'après les consignes et la conclusion que nous avons tirée de l'étape 3 ; dans cette étape nous avons décidé de ne travailler qu'avec les 4 paramètres que nous avons jugés 'pertinents' (activation, solver, learning_rate et max_iter).

Dans cette étape nous avons refait les :



- **Étapes 2 :** en faisant l'apprentissage avec les paramètres par défaut de Sickitlearn sur les deux datasets décrits dans l'étape 4.
- **Étape 3 :** en faisant l'apprentissage avec les paramètres optimaux (meilleur modèle trouvé grâce à GridSearchCV de Sickitlearn) sur les deux datasets décrits dans l'étape 4.

Dans cette étape :

- Nous avons gardé une seule métrique pour faciliter la comparaison : `accuracy_score` pour la classification et `r2_score` pour la régression.
- Nous avons exploité le TPU de [google colab](#) pour faire l'apprentissage (notamment pour trouver les paramètres optimaux), vu la taille des deux datasets de l'étape 4.

1. Les liens vers le code

Pour chaque type de problème (Classification/ Régression), nous avons rassemblé l'apprentissage avec les paramètres par défaut et celui avec les paramètres optimaux dans un seul code.

- **1.1. Classification :**  [GitHub](#)
- **1.2. Régression :**  [GitHub](#)

2. Discussion des Résultats Obtenus :

CLASSIFCATION				RÉGRESSION				
Étape 2/4		Étape 3/5		Étape 2/4		Étape 3/5		
« paramètres par défaut »		« paramètres optimaux »		« paramètres par défaut »		« paramètres optimaux »		
Dataset de Sickit Learn	Dataset de Super Annonate	Dataset de Sickit Learn	Dataset de Super Annonate	Dataset de Sickit Learn	Dataset de Kaggle	Dataset de Sickit Learn	Dataset de Kaggle	
«breast_cancer»	«featl_health»	«breast_cancer»	«featl_health»	«diabetes »	«insurance»	«diabetes »	«insurance»	
Taille du dataset	Score de prédiction							
	0.97	0.87	1	0.9	-0.80	-1.10	0.61	0.12
	569	2126	569	2126	442	1338	442	1338

- La classification donne toujours des résultats beaucoup mieux que la régression quel que soit le dataset utilisé Sickit-Learn ou autre (scores : 0.97, 0.87, 1, 0.9 pour la classification) contre (scores : -0.80, -1.10, 0.61, 0.12 pour la régression).

Nous concluons que la régression nécessite beaucoup plus de données (données massives) pour avoir un bon modèle étant donné que le nombre de classes à prédire dans la classification est déterminé alors que dans la régression est indéterminé.

La régression donne souvent un score négatif lorsque le modèle est très loin du modèle réel.

- La classification et la régression sur les datasets de Scikit-Learn donne de bons résultats par rapport aux autres datasets hors Sickit-Learn, malgré le peu d'instances qu'ils contiennent comparé aux nombre d'instances de ces derniers (569 instances contre 2126 pour la classification) et (442 instances contre 1338 pour la régression). Et cela que ce soit l'apprentissage fait avec paramètres par défaut ou avec paramètres optimaux.

Nous concluons que quel que soit le dataset de Sickit-Learn utilisé (taille, thème, type de problème : Classification ou Régression). Ce dernier sera parfait et optimisé pour Sickit-Learn. Du genre que, même les modèles de base construits avec les paramètres par défaut donnent de très bons résultats.

- L'apprentissage avec les paramètres optimaux (utilisation de GridSearchCV) donnent de meilleurs résultats par rapport à l'apprentissage avec les paramètres par défaut. Et cela quel

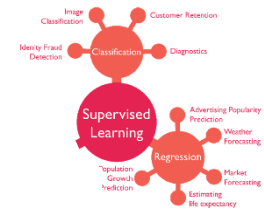
que soit le type de problème (classification ou régression) et la source du dataset (Sickit-Learn ou autre) : (scores : 1 contre 0.97, 0.9 contre 0.87, 0.61 contre -0.80, 0.12 contre -1.10).

Nous concluons, qu'il est important d'utiliser une méthode qui calibre les paramètres d'une façon automatique au lieu de les calibrer manuellement pour obtenir de meilleurs résultats. En procédant ainsi, nous serons sûrs que nous ressortirons tous les modèles qui puissent exister (d'après les valeurs introduites pour chaque paramètre) et ainsi, nous sélectionnerons le meilleur modèle.

Le seul inconvénient de cette méthode est que plus la taille du dataset augmente, plus elle consomme de mémoire, mais de nombreuses solutions sont présentes telles que faire l'apprentissage avec *google colab* qui fournit à l'utilisateur des GPU et TPU gratuitement (cette solution reste meilleure par rapport aux méthodes de calibrage manuelles).



'APPRENTISSAGE SEMI-SUPERVISÉ'

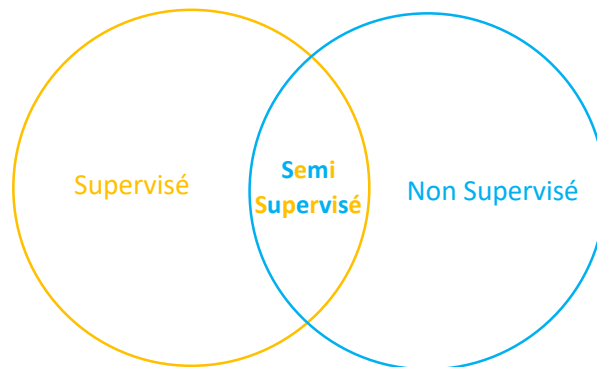


*Dans cette sixième étape, nous allons utiliser **scikit-learn** pour tester **ALGO** en apprentissage **semi-supervisé** sur les 2 jeux de données de classification précédents : **breast_cancer** de **scikit-learn** et **fetal_health** de **kaggle**.*

Étape 6

1.1 L'Apprentissage Semi-Supervisé :

C'est une technique d'apprentissage automatique qui consiste à apprendre une fonction de prédiction à partir d'un ensemble de données étiquetées (annotées) et non étiquetées. Ainsi, il se situe entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non supervisé qui n'utilise que des données non étiquetées.



1.2 Lien vers le code

Nous avons découpé les 2 datasets de classification en LAB, non-LAB (de 10% à 80%) et TEST (10 %). Puis nous avons entraîné le modèle en supervisé sur LAB. ensuite, nous avons amélioré le modèle en semi-supervisé sur non-LAB et enfin, nous avons testé le modèle sur TEST.

[- SEMI-SUPERVISE SUR Breast_cancer de Sk-learn](#)



[- SEMI-SUPERVISE SUR fetal_health de Kaggle](#)



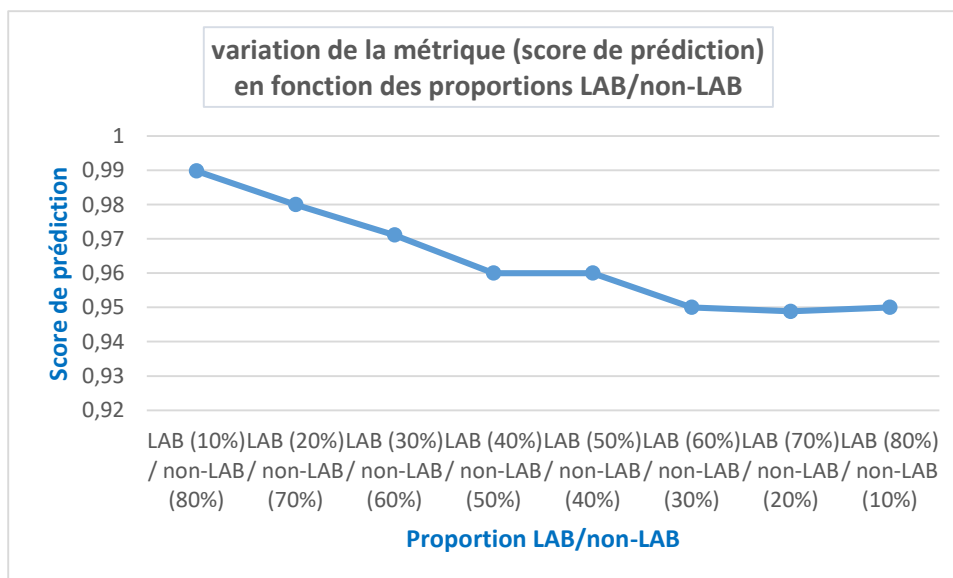
1.3 Résultats obtenus

Pour bien observer l'effet de cette technique ([semi supervisé](#)) sur la qualité de l'apprentissage, nous avons varié la proportion LAB/non-LAB et recalculé la métrique de qualité (score de prédiction) sur TEST.

LAB(10%)	non-LAB(80%)		TEST(10%)
LAB(20%)	non-LAB(70%)		TEST(10%)
LAB(30%)	non-LAB(60%)		TEST(10%)
LAB(40%)	non-LAB(50%)		TEST(10%)
LAB(50%)	non-LAB(40%)		TEST(10%)
LAB(60%)	non-LAB(30%)		TEST(10%)
LAB(70%)	non-LAB(20%)		TEST(10%)
LAB(80%)	nonLAB(10%)		TEST(10%)

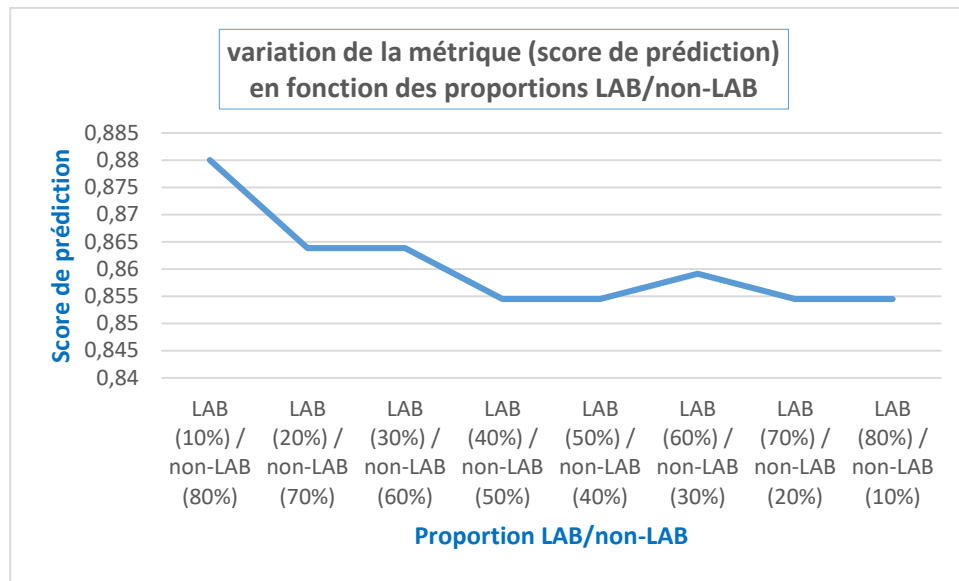
[NB : les graphes ont été générés avec Excel.](#)

Dataset Breast_cancer :



Nous remarquons que plus la quantité des données non étiquetées diminue (en // la quantité des données étiquetées augmente), plus le score de prédiction diminue (i.e la qualité de l'apprentissage diminue).

Dataset Fetal_health :



De même comme le graphe précédent, nous remarquons que plus la quantité des données non étiquetées diminue (en // la quantité des données étiquetées augmente), plus le score de prédiction diminue (i.e la qualité de l'apprentissage diminue).

1.4 Comparaison des résultats du semi-supervisé et supervisé et explication des résultats

Nous avons défini pour le semi-supervisé un modèle de base (avec paramètres par défaut), ce qui fait, nous avons comparé les résultats avec les résultats de l'étape 2 du supervisé (modèle de base) car il a donné déjà de très bons résultats.

Classification			
Supervisé « paramètres par défaut »		Semi-supervisé « paramètres par défaut »	
Dataset de Sickit Learn «breast_cancer»	Dataset de Super Annonate «featl_health»	Dataset de Sickit Learn «breast_cancer»	Dataset de Super Annonate «featl_health»
Score de prédiction 0.97	0.87	0.989	0.88
Taille du dataset 569	2126	569	2126

- La classification dans le semi supervisé donne des résultats mieux que le supervisé quel que soit le dataset utilisé Sickit-Learn ou autre et quel que soit sa taille (score : 0.989 contre 0.97 pour le dataset breast_cancer de sk-learn) et (score : 0.88 contre 0.87 pour le dataset fetal-health de kaggle).

Nous concluons que la classification en semi-supervisé est plus efficace qu'en supervisé.

1.5 Conclusion

Les données non étiquetées, en combinaison avec **peu** de données étiquetées, permettent de produire une amélioration considérable de la précision de l'apprentissage. Ainsi, ceci améliore significativement la qualité de l'apprentissage.

Pour un problème d'apprentissage donné, l'étiquetage de données nécessite souvent l'intervention d'un utilisateur humain. En revanche, les estimateurs semi-supervisés **sklearn.semi_supervised** (grâce à la fonction **predict_proba** pour les RN) sont capables d'utiliser ces données supplémentaires non étiquetées pour mieux saisir la forme de la distribution des données sous-jacentes et mieux les généraliser à de nouveaux échantillons.

Ces algorithmes peuvent bien fonctionner lorsque nous avons une très petite quantité de données étiquetées et une grande quantité de points non étiquetés, exemple de (LAB 10% non-LAB 80%).

.

CONCLUSION GENERALE

Pour conclure, nous témoignons que ce projet de TER nous a été d'une très grande aide pour améliorer collectivement nos connaissances en apprentissage automatique supervisé et semi-supervisé pour les problèmes de classification et de régression, notamment en réseaux de neurones (un domaine qui nous intéresse beaucoup).

Nous tenons ainsi, à exprimer notre gratitude à nos deux encadreurs, M. Ludovic JAVET et M. Luc BOUGANIM pour leur encadrement, leur patience et leurs constantes orientations tout au long de ce projet. En y accordant une méticuleuse attention, pour leurs judicieux conseils et leurs précieuses explications qui nous ont été indispensables à la conduite de ce projet. Nous les remercions ainsi pour leur extrême amabilité et leur totale disponibilité.

Qu'il trouve dans ce modeste travail un hommage vivant à leur haute personnalité.

