L2 MI - MINI PROJET
Challenge HADACA
Membres:
Yanis ROCHDI, Polina GRIGORIEVA
Sarah HAMMACHE, Samy FERROUDJI
Moussa NIAKATE, Yasmine GHEZALI
URL du challenge: https://codalab.lri.fr/competitions/333
Github: https://github.com/HEALTH123Soleil/health
Groupe administratif : 3
Nom de l'équipe : HEALTH
Numéro de la dernière soumission de CODE : 8506
URL youtube : https://youtu.be/Mv5e6wCTgXs

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir®

Introduction:

Le challenge HADACA consiste en un problème de classification multi-classes dans lequel nous devons prédire la phase d'avancement du cancer d'un patient à partir de ses prélèvements grâce auxquels nous auront 5000 caractéristiques d'ADN. Il y a en tout 10 stages différents, et nous devrons donc prédire ces stages avec une précision au moins supérieure à 10%. Nos données sont sous forme d'une matrice de (Nombre de patients) lignes * (nombres d'attributs par patient) colonnes. Les attributs (features) correspondent aux informations relatives à la méthylation de chaque patient.
La particularité des données de ce challenge est qu'il n'y a pas assez de patients par rapport au nombre d'attributs (features).

Le projet a été divisé en trois parties : la partie preprocessing, ou nous avons combiné entre des méthodes de Dimension Reduction afin de trouver la meilleure combinaison pour un meilleur traitement de données; la partie classification, où comme nous avons un problème de classification multi classes, nous avons décidé de tester les modèles SVM, Random Forest. Pour finir, il y'a la partie interface graphique, où nous avons principalement utilisé la librairie Pandas et sklearn, car nous trouvons qu'elles sont les plus adaptées à notre projet et les plus complètes.

1) Preprocessing : Sara HAMMACHE et Samy FERROUDJI

L'étape primordiale du Preprocessing consiste à réaliser le prétraitement des données qui est nécessaire pour éliminer les données inutiles et transformer les données brutes en données exploitables pour l'étape de la classification. Les données brutes (réelles) sont souvent incomplètes et ne peuvent donc pas être utilisées directement par le modèle de prédiction.

Dans ce challenge HADACA, le dataset est une matrice de (Nombre de patients) lignes * (nombres d'attributs par patient) colonnes. Les attributs (features) correspondent aux informations relatives à la méthylation de chaque patient.
La particularité des données de ce challenge est qu'il n'y a pas assez de patients par rapport au nombre d'attributs.

Afin de permettre le séquencement des différentes transformations des données, qui doivent être exécutées dans le bon ordre, nous avons utilisé la classe **Pipeline** fournie par **Scikit-Learn.** Le constructeur Pipeline prend en entrée une liste de paires nom/estimateur définissant une suite d'étapes. Tous les estimateurs à l'exception du dernier doivent posséder une méthode fit_transform ().
A l'appel de la méthode fit () du pipeline, celle ci appelle tour à tour fit_transform () de chacun des transformateurs, récupérant la sortie de chacun des appels pour la transmettre à l'appel suivant

```
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# ##############################################################################
# Define a pipeline  with a classifier (on devrait pouvoir le remplacer par le model du binome prediction)
pipeline = Pipeline([
    ('svd',TruncatedSVD()),
    ('pca', PCA()),
    ('svc', SVC()),
```

Nous avons également utilisé la méthode **GridSearchCV** qui aide à régler les hyper paramètres de l'estimateur, en sélectionnant les paramètres qui produisent le meilleur score dans les données de validation. Et donc rajouter un classifieur dans la Pipeline pour pouvoir tester les meilleurs paramètres.

Nous avons donc comparé entre plusieurs combinaisons de méthodes dans la pipeline. [voir Annexe Preprocessing]

Il est à noter que nous avons eu **un meilleur score** en n'utilisant que la méthode **TruncatedSVD** seule pour le Preprocessing des données, et ceci dans le **model.py**. Tandis que la classe **Preprocessing** contient tout ce qui a été mentionné auparavant.

Etant donné un score de prédiction de 1 à la soumission, il est fort possible qu'il y' ait une fuite de données "**Data leakage**" entre les données de Train et ceux du Test. Pour cela, nous comptons faire une **étude de corrélation** entre ces deux ensemble de données. Et ceci à l'aide de la méthode **`matthews_corrcoef`** proposée par Sklearn.metrics.

Représentation des données :
Par ailleurs, nous avons également représenté nos données d'apprentissage en 2D selon les composantes PCA et LDA afin de comparer entre ces deux méthodes.
La LDA, contrairement à la PCA, est une méthode **supervisée**, et qui donc utilise les labels des données d'apprentissage [voir Annexe Preprocessing] .

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
data_slice = LDA(n_components=2).fit_transform(data.loc[:,~data.columns.str.contains('^target$')].values, data['target'].values)
```

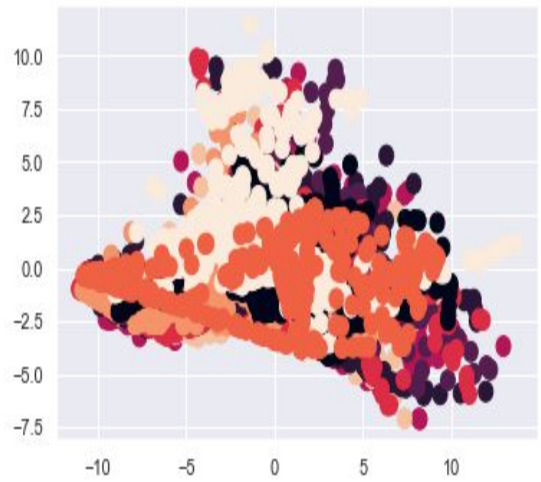


Figure 01

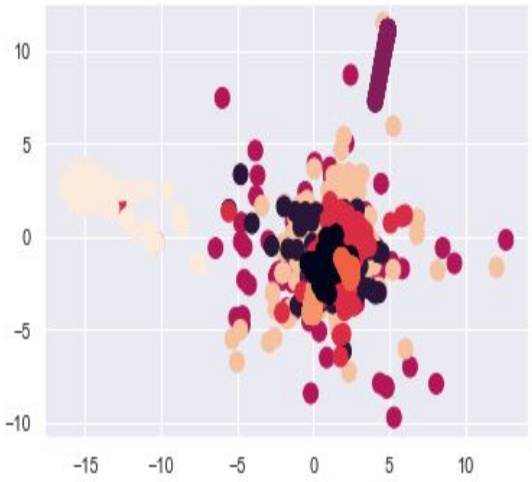


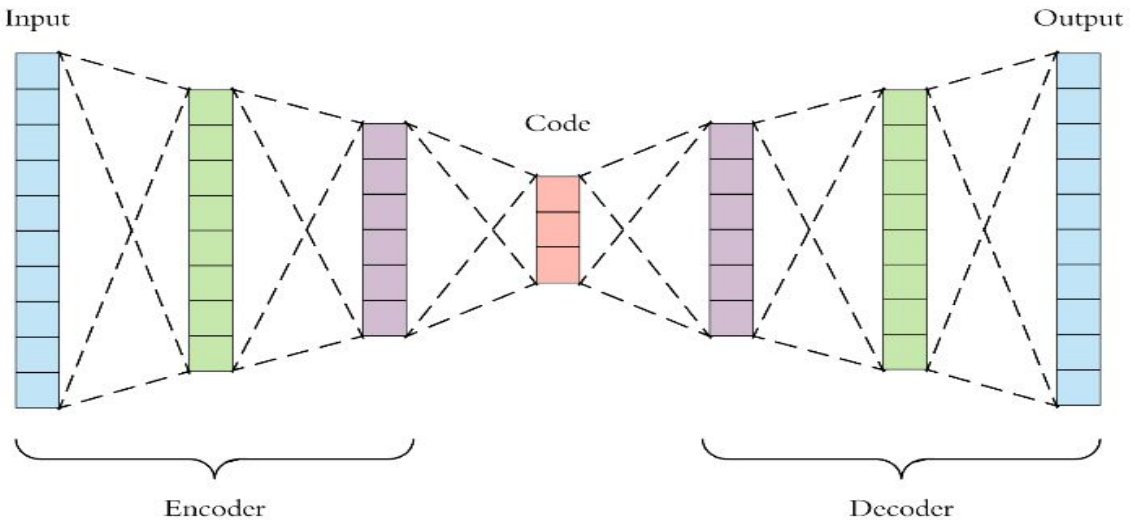
Figure 02

Figure 01 : Projection 2D des données d'apprentissage en PCA.
Figure 02 : Projection 2D des données d'apprentissage en LDA.

Avenue de recherche future :

En faisant de la bibliographie sur le mode de fonctionnement général des **auto-encodeurs**, nous avons constaté que c'est une méthode d'apprentissage assez puissante et prometteuse pour faire de la réduction de dimension. Nous avons pu comprendre le fonctionnement général et pensé à utiliser Keras comme bibliothèque, mais nous n'avons finalement pas pu l'implémenter et l'appliquer à nos données, le temps ne nous a pas permis de le faire en espérant que ce sera l'une des prochaines améliorations.

Les auto-encodeurs sont des algorithmes d'apprentissage **non supervisé** à base de **réseaux de neurones** artificiels, qui permettent de construire une nouvelle représentation d'un jeu de données. Généralement, celle-ci est plus compacte, et présente moins de descripteurs, ce qui permet de réduire la dimensionnalité.L'architecture d'un auto-encodeur est constitué de deux parties : l'**encodeur** et le **décodeur.**



Dans un tel réseau, 4 hyper paramètres sont nécessaires :

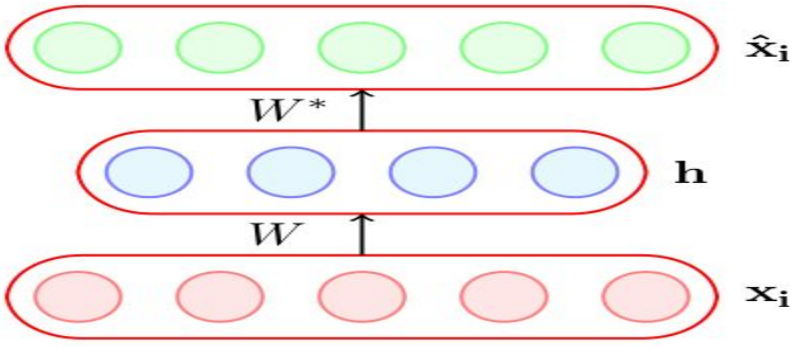
- **Nombre de couches** : l'auto-encodeur peut être aussi profond que l'on veut. Dans la figure précédente, il est composé de 2 couches y compris l'encodeur et le décodeur, sans compter les entrées et les sorties.
- **Code size** : nombre de neurones de la couche d'au milieu (Code).
- **Nombre de neurones par couche** : le décodeur est souvent symétrique à l'encodeur.
- **Fonction de coût (Loss function)** : principalement la méthode de l'erreur quadratique moyenne (MSE pour Mean Squared Error).

Les auto-encodeurs sont entraînés via la **backpropagation**.

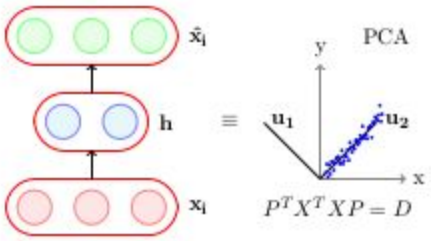
La relation entre les auto-encodeurs et la PCA :

Si des activations **linéaires** sont utilisées, ou uniquement une **seule** couche cachée sigmoïde, la solution optimale d'un auto-encodeur est apparentée à une analyse en composantes principales (PCA).

Un autoencoder encode l'entrée x dans la couche cachée h, puis la décode à partir de cette couche cachée. Le modèle est entraîné pour minimiser la fonction de coût.



A l'aide de la couche cachée h, il est possible de reconstruire ceci. Cette couche **h** capture toutes les caractéristiques importantes de **xi**. L'analogie avec PCA est la suivante: **h** se comporte comme la **matrice à dimensions réduites** de **PCA** à partir de laquelle la sortie est reconstruite avec une certaine perte de valeur. Par conséquent, la partie **encodeur** montre une ressemblance avec la **PCA**.



Les conditions qui font d'un auto-encodeur un PCA :

Ces deux méthodes seront équivalentes si l'encodeur et le décodeur sont **linéaires** et la fonction de coût est quadratique avec des entrées normalisées.
En raison de ces contraintes de linéarité, opter pour des encodeurs avec des fonctions **non linéaires** de type **sigmoïde**, offrent une plus grande **précision** dans la reconstruction des données.

2) Classification: Yasmine GHEZALI et Moussa NIAKATE

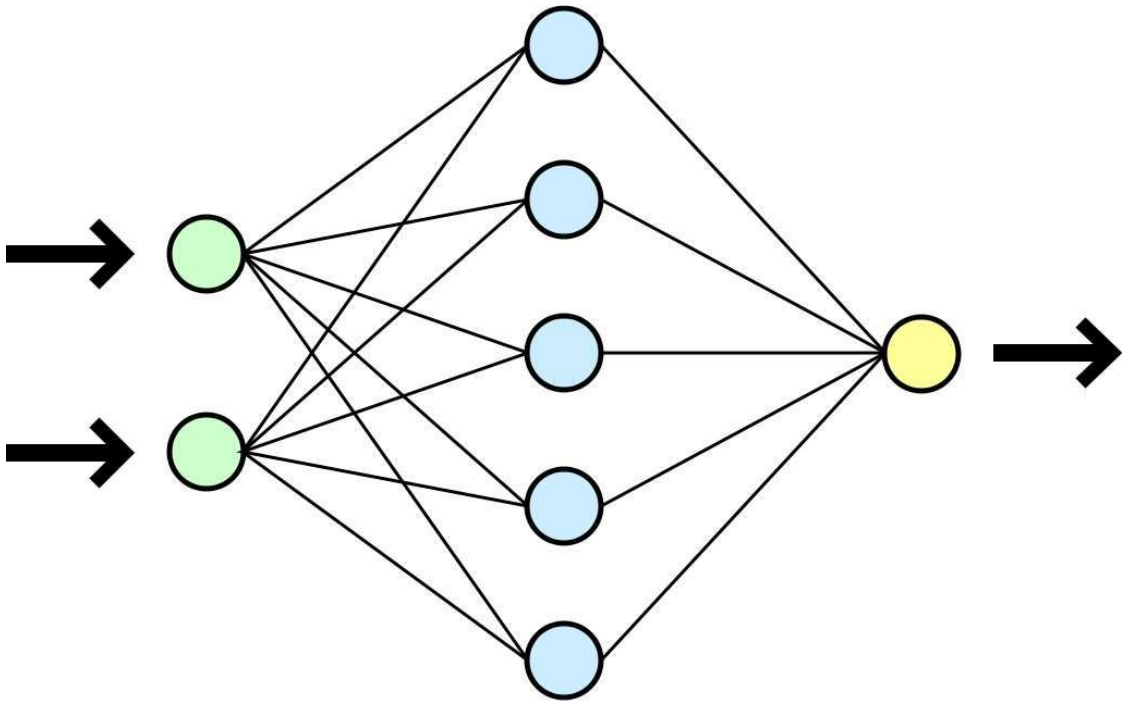
Pour la partie classification, nous avons récupéré les données après les avoir "fit" et avons utilisé en premier lieu le classifieur SVM. Nous avons eu énormément de bugs dans notre code et avons passé beaucoup de temps à le corriger pour réussir enfin à le faire fonctionner. En revanche, les résultats obtenus ne sont pas satisfaisants.

Suite à quelques modifications tel que le choix de classifieur ou encore les méthodes appliqués, nous nous sommes concentrés sur la méthode MLP(**multilayer perceptron**).

Cette méthode s'applique dans le cadre d'un apprentissage supervisé. Par cette méthode, nous avons pu changé radicalement notre score et simplifié notre code pour obtenir un score de 100%. Il s'avère que cette méthode est la bonne pour nos données car elle nous a totalement remis notre score à jour et concorde parfaitement avec nos donnés. On peut donc en déduire que concernant la partie prédiction, le challenge est réussi.

<u>Quelques notions importantes</u>:

- **Réseau de neurones**: système créé à partir du modèle du cerveau et du système nerveux des hommes. Les "neurones" sont interconnectés et résolvent les problèmes grâce à l'ajustement de coefficients de pondération pendant la phase d'apprentissage .



- **Somme pondérée**: Pour calculer la somme des valeurs d'une série, pondérés par leurs effectifs (ou par des coefficients) : on multiplie chaque valeur par son effectif (ou son coefficient) puis on additionne tous les produits obtenus.
- Dans un modèle de réseau de neurones, les poids correspondent à l'efficacité de la connexion entre deux neurones (qualité de transmission des signaux)

- **Règle de Hebb :** C'est une règle d'**apprentissage** des réseaux de neurones artificiels dans le contexte de l'étude d'assemblées de neurones. Lorsque deux neurones sont excités conjointement, il se crée ou renforce un lien les unissant.

$$W_i' = W_i + \alpha(Y.X_i)$$

Où W_i' représente le poids corrigé et alpha le taux d'apprentissage.

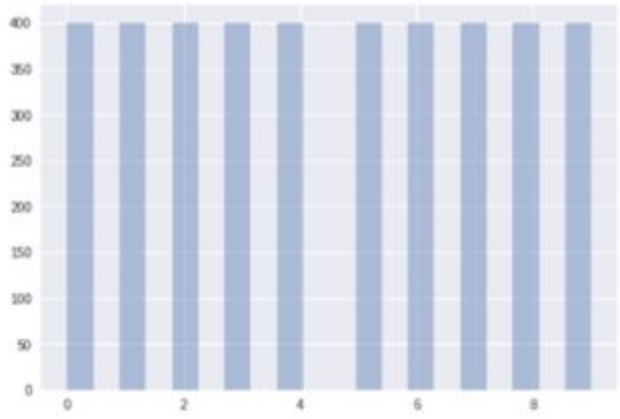
- **Règle d'apprentissage du perceptron** : Le perceptron est très proche de la règle de Hebb, la grande différence étant qu'il tient compte de l'erreur observée en sortie.

$$W_i' = W_i + \alpha(Y_t - Y)X_i$$

3) <u>Visualisation</u> : Yanis ROCHDI et Polina GRIGORIEVA

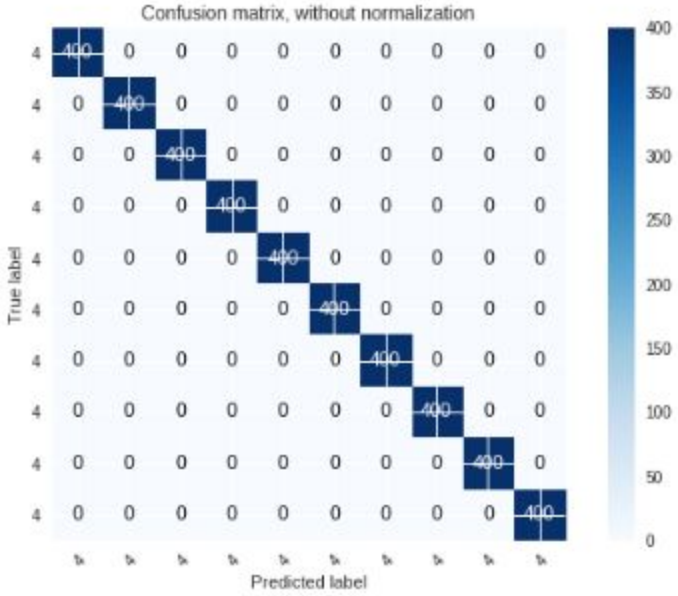
Afin d'avoir un aperçu plus global des donnée nous avons crée un histogramme Pour cela nous avons utilisé la librairie sklearn et LabelEncoder car les classes ("target") étaient en string et nous les avons encodé en numéro afin de tracer le plot à l'aide de la doc sns.displot **[9]**

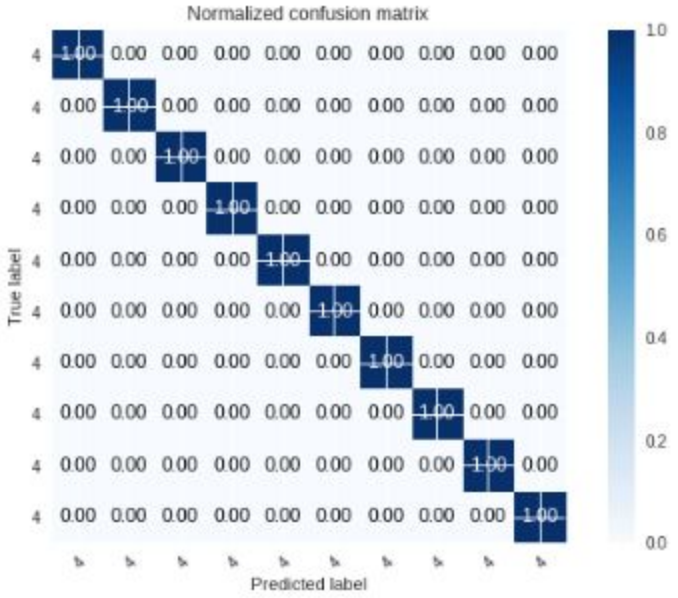
```
In [40]: import seaborn as sns
         from sklearn.preprocessing import LabelEncoder
         data=data.apply(LabelEncoder().fit_transform)
         #data['target']
         sns.distplot(data['target'].values, bins=20, kde=False, rug=False)
         #data
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc050560e10>
```



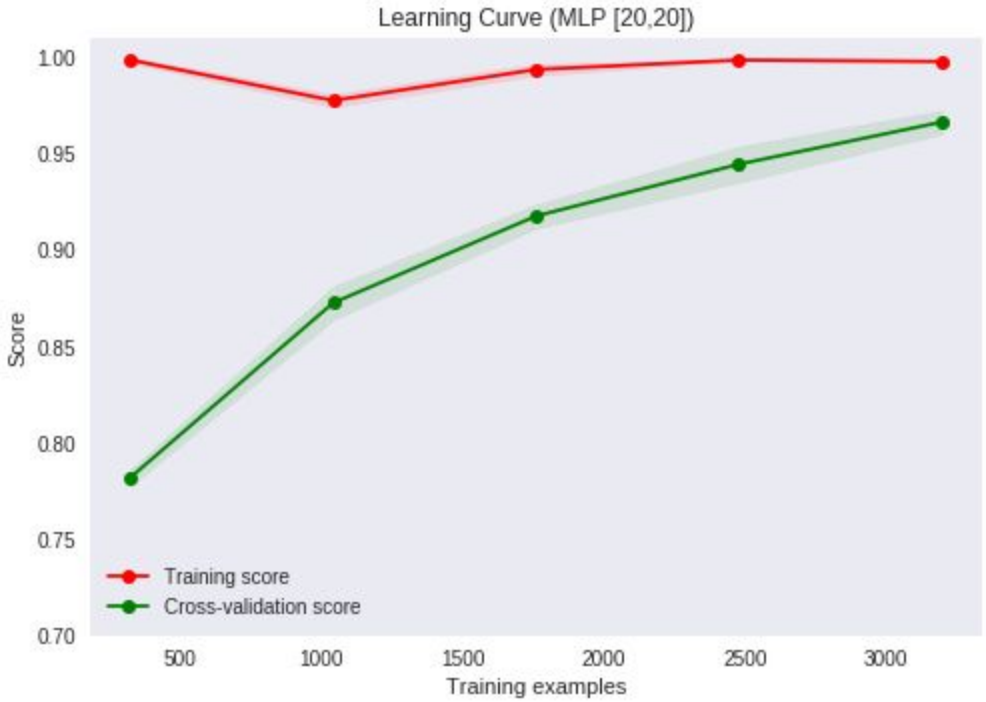
Pour mieux résumer les résultats de ce que fait notre algorithme nous avons décidé de créer une matrice de confusion grâce à la classe Matrice confusion de la bibliothèque

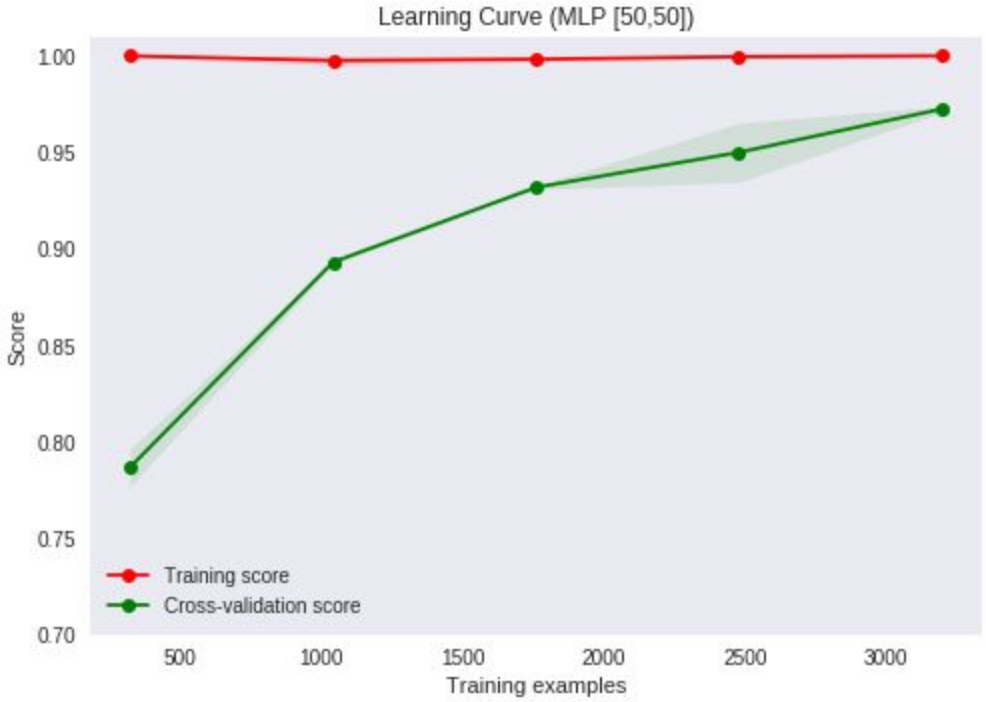
SKlearn:[10]

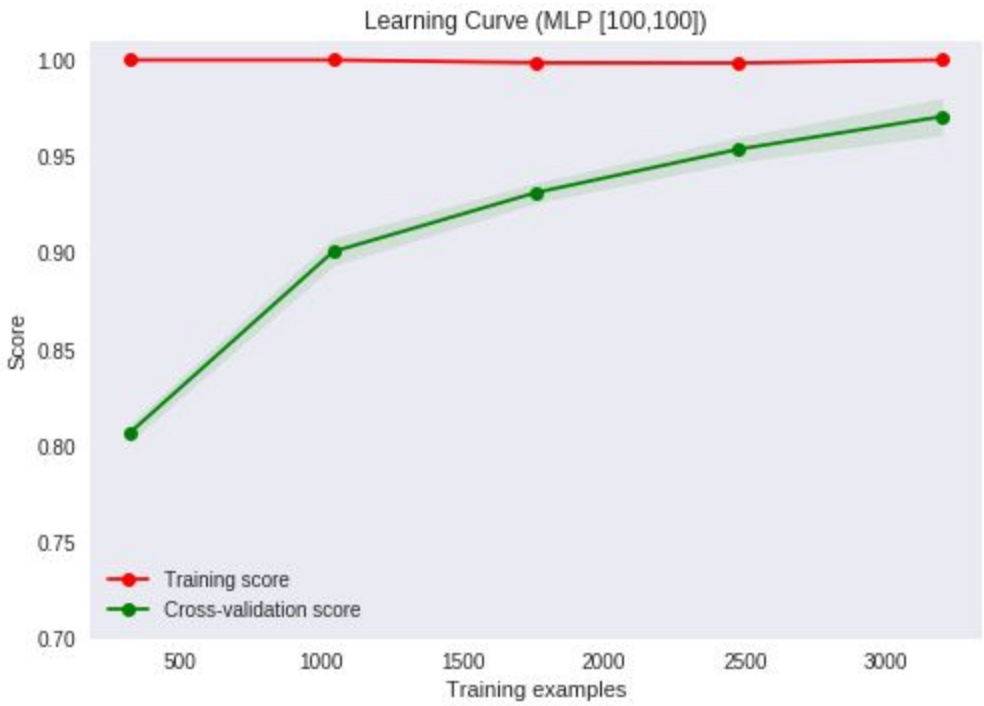




De plus l'aide de **[10]** nous avons fait des learning curve qui représentent le score en fonction des donnée d'apprentissage en faisant varier l'hyper paramètre des couches de notre modèle pour savoir si il y avait une baisse du score passé un nombre de donnée .


Learning Curve (MLP [20,20])
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Score
Training score
Cross-validation score
500
1000
1500
2000
2500
3000
Training examples


Learning Curve (MLP [50,50])
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Score
Training score
Cross-validation score
500
1000
1500
2000
2500
3000
Training examples



<u>Conclusion & discussion</u> :

Nous avons donc pu voir l'importance de **l'intelligence artificielle**, qui sert dans tous les domaines, notamment concernant notre santé, chose des plus importantes, et peut-être même dans des progrès scientifiques dans ce domaine. En ce qui concerne les étudiants qui nous lisent, il faut aussi noter l'importance d'**application des notions théoriques** apprises dans le cours du premier semestre de Vie Artificielle, ce qui nous a, au demeurant, permis de mieux comprendre et de consolider nos connaissances et notions liées au machine learning, d'où l'importance de ce projet.
De plus, ce projet nous a été utile dans la découverte de la vie professionnelle. En effet, cela nous a initié au respect des **cahiers des charges**, et des **délais** du rendu de travail, ainsi que **l'esprit d'équipe** : la coordination des tâches, la communication, la division de travail...toutes ces choses très importantes qui vont nous servir dans notre vie professionnelle. Nous espérons que ce projet sera utile dans le domaine médical, car cette thématique nous tient à coeur, et nous voulons donner au moins une once de nous dans l'espoir d'aider ces personnes.

Références :

Auto-encodeurs:https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Réseau de neurones:https://www.futura-sciences.com/tech/definitions/informatique-reseau-neuronal-601/

Hebb & Perceptron : https://fr.wikipedia.org/wiki/Perceptron

Sur-apprentissage https://mrmint.fr/overfitting-et-underfitting-quand-vos-algorithmes-de-machine-learning-derapent

LDA and PCA : https://sebastianraschka.com/faq/docs/lda-vs-pca.html

https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-lda-py

Cross validation https://openclassrooms.com/fr/courses/4297211-evaluez-et-ameliorez-les-performances-dun-modele-de-machine-learning/4308241-mettez-en-place-un-cadre-de-validation-croisee

Histogramme

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

https://seaborn.pydata.org/generated/seaborn.distplot.html

Matrice de confusion :

https://www.lebigdata.fr/confusion-matrix-definition

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

10. Learning Curve :

https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

Section Bonus

Table 1: Statistiques sur les données

Dataset	Num. Examples	Num. Variables/features	Sparsity	Has categorical variables?	Has missing data?	Num. examples in each class
Training	4000	4000	0	No	No	400
Validation	500	500	0	No	No	50
Test	500	500	0	No	No	50

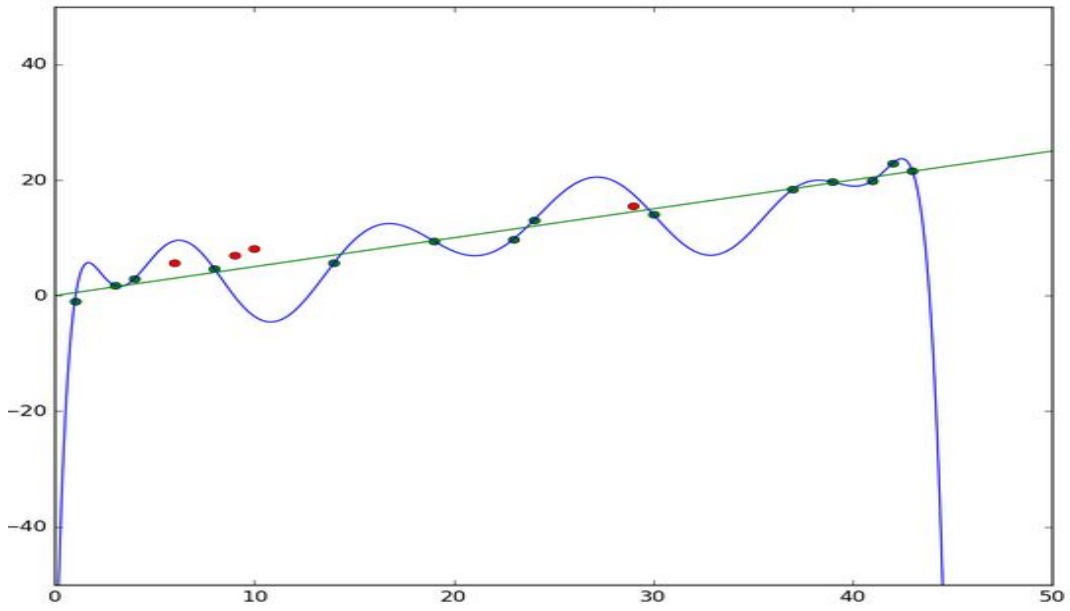
Data Sparsity : la proportion des 0 dans la matrice de données.

Nous disposons en tout de 5000 exemples, nous remarquons donc que les données d'apprentissage (train) représentent 80% des données, données de validation 10%, et les données de test 10%.

Le sur-apprentissage (Overfitting) :

Le sur-apprentissage désigne le fait que le modèle prédictif s'adapte trop bien aux données d'apprentissage.
Ainsi, le Overfitting est caractérisé par une erreur de type variance très élevée

Quand un tel événement se produit, le modèle prédictif pourra donner de très bonnes prédictions sur les données d'apprentissage (les données qu'il a déjà "vues" et auxquelles il s'y est adapté), mais il prédira mal sur des données qu'il n'a pas encore vues lors de sa phase d'apprentissage. Et donc, un tel modèle manque de capacité de **généralisation**.



Le graphe ci-dessus montre un exemple de sur-apprentissage. Le tracé en bleu représente une fonction de prédiction qui passe par toutes les données d'apprentissage *(points en vert)*. On voit bien que la fonction est instable (grande variance) et qu'elle s'écarte beaucoup des points rouges qui représentent des données non vues lors de la phase d'apprentissage *(données de test)*.

La Cross Validation :

Est une méthode d'estimation de fiabilité d'un modèle fondée sur une technique d'échantillonnage. Elle est utilisée pour détecter un surajustement, par exemple, l'échec de la généralisation d'une tendance.
Il est acquis qu'un modèle doit être évalué sur une base de test différente de celle utilisée pour l'apprentissage. Mais la performance est peut-être juste l'effet d'une aubaine et d'un découpage particulièrement avantageux. Pour être sûr que le modèle est robuste, on recommence plusieurs fois. On appelle cela la validation croisée.

Dans la **validation croisée k-fold**, on divise les données d'entrée en k sous-ensembles de données (ou échantillons). On forme un modèle d'apprentissage-machine sur tous les sous-ensembles sauf un (k-1), puis évalue le modèle sur le sous-ensemble qui n'a pas été utilisé pour la formation. Ce processus est répété **k fois**, avec un sous-ensemble différent réservé à l'évaluation (et exclu de la formation) à chaque fois.

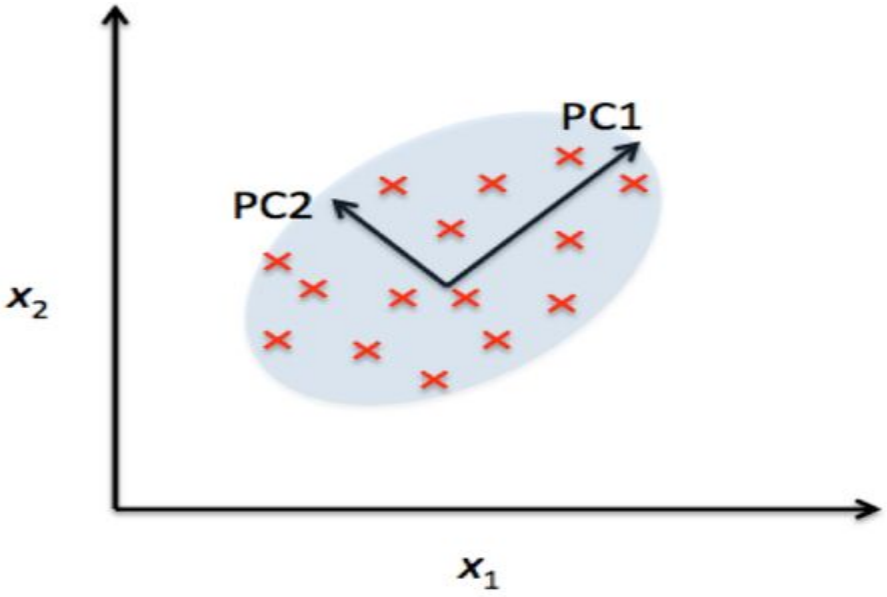
Annexe Preprocessing

I. Différence entre PCA & LDA pour la réduction de dimension :

Ce sont tous les deux des techniques de transformation linéaire.

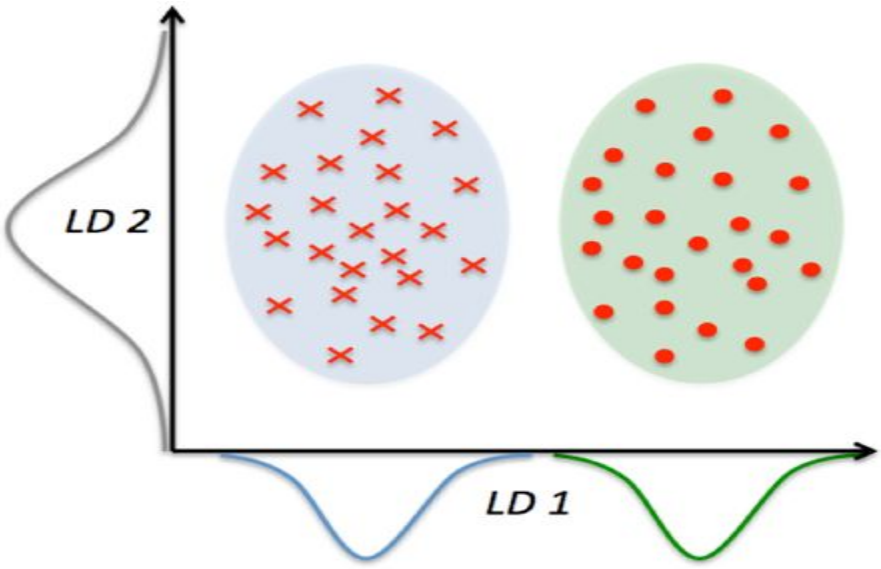
1. PCA (Principal Component Analysis)

- La PCA est une méthode **non-supervisée** (ignore les labels des classes) qui trouve la direction qui **maximise la variance**.
- Découvre la relation entre les variables **indépendantes** seulement.



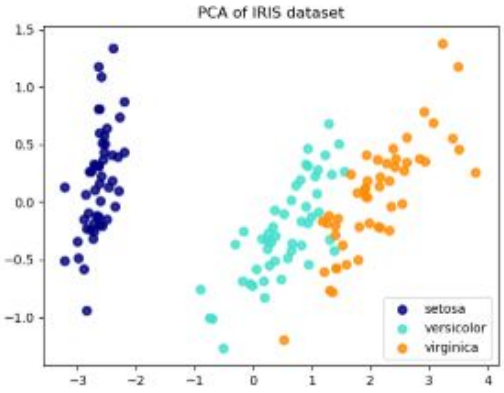
2. LDA (Linear Discriminant Analysis)

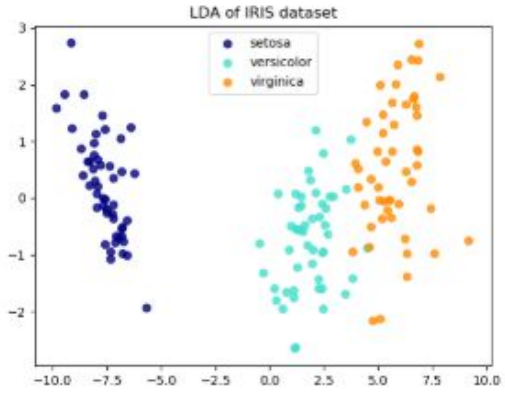
- C'est une généralisation du **FLD** (Fisher's linear discriminant) et une technique **supervisée** d'analyse discriminante prédictive.
- Contrairement à la PCA, la LDA tends à trouver la direction qui maximise la **différence** entre les classes (un sous-ensemble de features qui maximise la **séparabilité** des classes).
- Notons que la LDA fait des hypothèses concernant les classes **normalemen**t distribuées (selon la loi gaussienne/normale) et des classes ayant une **covariance** égale.
- Découvre la relation entre les variables ET **dépendantes** ET **indépendantes**.



L'utilisation de cette méthode LDA dans le preprocessing des données fait partie des projets futurs de notre Challenge.

Exemple : cas de Iris Dataset





II. La méthode TruncatedSVD :

La SVD (Décomposition en valeurs singulières), l'une des méthodes de la Dimension Reduction, est une manière de factoriser une matrice non-carrée qui généralise l'opération de décomposition en valeurs propres (pour une matrice symétrique), et qui fonctionne donc pour n'importe quelle matrice de taille m × n (pas forcément carrée). La troncation des valeurs singulières produit des approximations de bas-rang d'une matrice d'entrée.

Après l'application de la méthode SVD avec 2 composantes sur 100 exemples et 5000 features :

	component_1	component_2	target
0	29.810658	-5.346333	stage ib
1	25.964798	0.995554	stage ib
2	31.508732	-1.363731	stage ib
3	24.906895	-1.200450	stage ib
4	26.681459	0.100476	stage ib

Figure 6

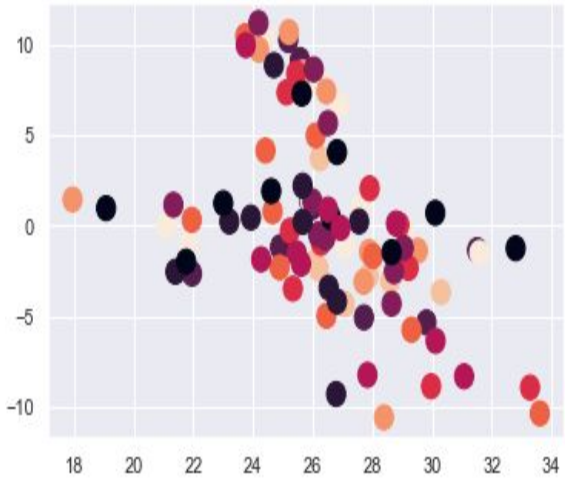


Figure 7

- **La comparaison de plusieurs combinaisons de méthodes**

```
pipeline: ['SelectKbest', 'PCA', 'MLP']

parameters:
{'PCA__n_components': (2, 3, 4, 5, 6, 7, 8, 9, 10)}

Fitting 3 folds for each of 9 candidates, totalling 27 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  27 | elapsed:    0.4s remaining:    0.5s

Best score: 0.220
Best parameters set:
        PCA__n_components: 10
```

```
pipeline: ['PCA', 'MLP']

parameters:
{'PCA__n_components': (2, 3, 4, 5, 6, 7, 8, 9)}

Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Best score: 0.250
Best parameters set:
        PCA__n_components: 5
```

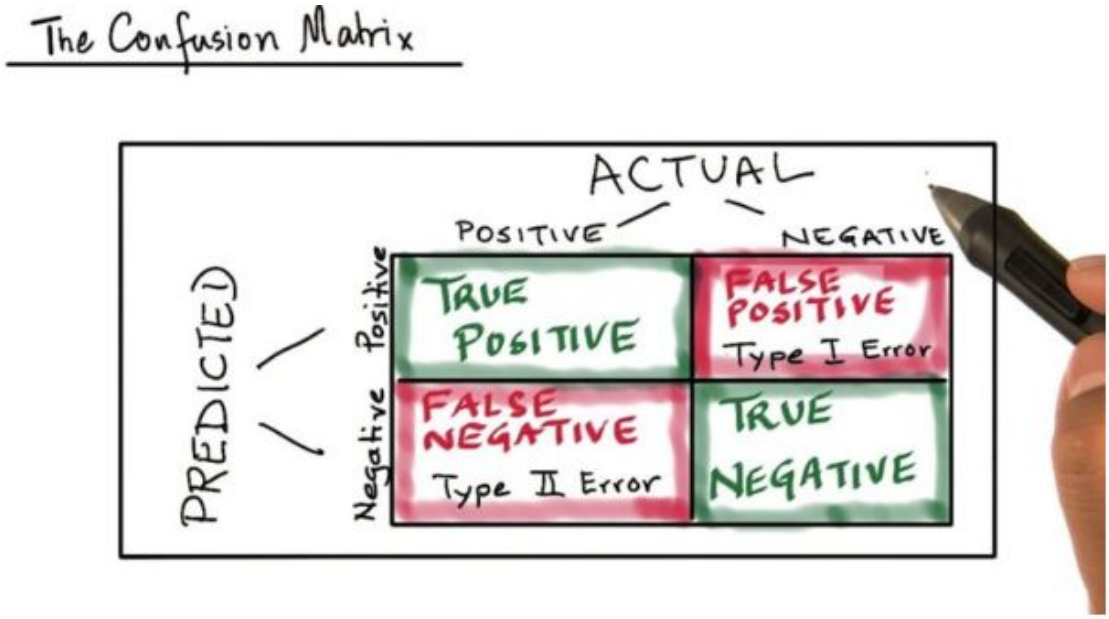
```
pipeline: ['SVD', 'MLP']

parameters:
{'SVD__n_components': (2, 3, 4, 5, 6, 7, 8, 9)}

Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 out of  24 | elapsed:    0.2s remaining:    0.5s

Best score: 0.260
Best parameters set:
        SVD__n_components: 8
```

```
pipeline: ['SelectKbest', 'SVD', 'MLP']

parameters:
{'SelectKbest__score_func': (<function f_classif at 0x000001852D517AE8>, <function f_regression at 0x000001852D517C80>), 'SVD__
n_components': (2, 3, 4, 5, 6, 7, 8, 9)}

Fitting 3 folds for each of 16 candidates, totalling 48 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Best score: 0.210
Best parameters set:
        SVD__n_components: 3
        SelectKbest__score_func: <function f_classif at 0x000001852D517AE8>
```

```
pipeline: ['SelectKbest', 'LDA', 'MLP']

parameters:
{'SelectKbest__score_func': (<function f_classif at 0x000001852D517AE8>, <function f_regression at 0x000001852D517C80>), 'LDA__
n_components': (2, 3, 4, 5, 6, 7, 8, 9, 10)}

Fitting 3 folds for each of 18 candidates, totalling 54 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.3s

Best score: 0.250
Best parameters set:
        LDA__n_components: 9
        SelectKbest__score_func: <function f_classif at 0x000001852D517AE8>
```

ANNEXE VISUALISATION

Matrice de confusion : Une Confusion Matrix est un résumé des résultats de prédictions sur un problème de classification. Les prédictions correctes et incorrectes sont mises en lumière et réparties par classe. Les résultats sont ainsi comparés avec les valeurs réelles.[11]



- **TP (True Positives)** : les cas où la prédiction est positive, et où la valeur réelle est effectivement positive. Exemple : le médecin vous annonce que vous êtes enceinte, et vous êtes bel et bien enceinte
- **TN (True Negatives) :** les cas où la prédiction est négative, et où la valeur réelle est effectivement négative. Exemple : le médecin vous annonce que vous n'êtes pas enceinte, et vous n'êtes effectivement pas enceinte.
- **FP (False Positive) :** les cas où la prédiction est positive, mais où la valeur réelle est négative. Exemple : le médecin vous annonce que vous êtes enceinte, mais vous n'êtes pas enceinte
- **FN (False Negative) :** les cas où la prédiction est négative, mais où la valeur réelle est positive. Exemple : le médecin vous annonce que vous n'êtes pas enceinte, mais vous êtes enceinte.