**RAPPORT**

***Réalisation d’un model de prédiction des fleurs « Iris »***

|  |
| --- |
|  |
|  |

Table de matières

[Importation du DataSet : 3](#_Toc67927573)

[Encodage des donnes non numériques : 3](#_Toc67927574)

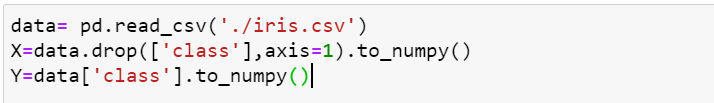
[Normalisation des donnes : 4](#_Toc67927575)

[Exploitation du Dataset 5](#_Toc67927576)

[Architecture et conception du model 5](#_Toc67927577)

[Entrainement et Evaluation : 7](#_Toc67927578)

# Importation du DataSet :

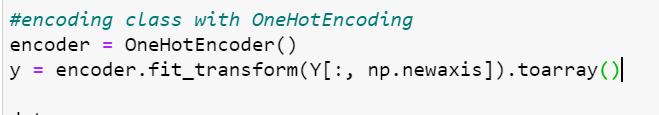


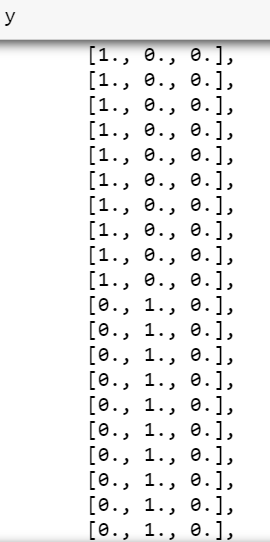
En utilisant la bibliothèque Pandas on a importer le fichier csv (Comma-Separated Values) qui contient les donnes des fleurs IRIS. Les donnes sont représentées par 4 variable quantitatifs et une classe qui détermine le type de fleurs (Iris-setosa, Iris-versicolor et Iris-virginica).

Dans la ligne 2 et 3 on sépare les variables explicatifs (SepalLength, SepalWidth, PetalLength, PetalWidth) du variable expliqué (class), et on les transforme en tableaux du type Numpy\_Array car c’est le format le plus simple et les plus utilisé dans l’analyse de donnes



# Encodage des donnes non numériques :

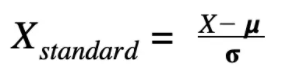


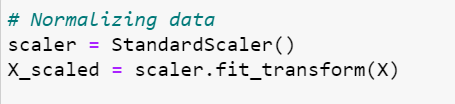
La fonction fit\_transform du classe OneHotEncoder appliquée sur un tableau de données qualitatif (la classe des fleurs), consiste à encoder ce dernier à *n* états sur *n* bits dont un seul prend la valeur 1 , le numéro du bit valant 1 étant le numéro de l'état pris par la variable.

Cette étape est nécessaire pour pouvoir utiliser les techniques du Machine Learning et Deep Learning (Classification et régression)

# Normalisation des donnes :

Travailler avec des données d’échelles variables peut constituer un problème dans l’analyse en ce sens qu’une variable numérique dont la plage de valeurs est comprise entre 0 et 100 sera plus pesante dans l’analyse qu’une variable dont les valeurs sont comprises entre 0 et 1 ce qui causerait un problème de biais par la suite.



La bibliothèque Scikit-Learn offre la classe la classe StandarScaler() qui contient plusieurs fonction qui nous aide a la normalisation des données

# Exploitation du Dataset

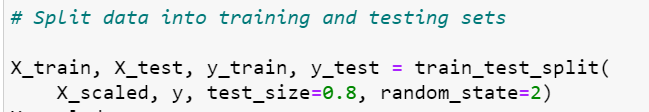
L’entraînement d’un modèle revient à mesurer l’erreur de la sortie de l’algorithme avec les données d’exemple et chercher à la minimiser.

Donc il faut éviter d'évaluer la qualité du modèle final à l'aide des mêmes données qui ont servi pour l'entraînement, sinon l'erreur sera précisément minimum sur ces données. Alors que l'erreur sera toujours plus élevée sur des données que le modèle n'aura jamais vues !

Pour minimiser ce problème, la meilleure approche est de séparer dès le départ notre jeu de données en deux parties distinctes :

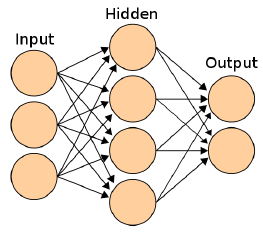
* Le training set, qui va nous permettre d’entraîner notre modèle et sera utilisé par l’algorithme d’apprentissage.
* Le testing set, qui permet de mesurer l’erreur du modèle final sur des données qu’il n’a jamais vues. On va simplement passer ces données comme s'il s'agissait de données que l’on n'a encore jamais rencontrées (comme cela va se passer ensuite en pratique pour prédire de nouvelles données) et mesurer la performance de notre modèle sur ces données

En général, les données sont séparées avec les proportions suivantes : 80 % pour le training set et 20 % pour le testing set. Mais dans notre cas le dataset est de petit taille (150 échantillons), donc on entraine le model sur 50% du data et on le test sur 50% restant

On peut utiliser la fonction de scikit-learn  train\_test\_split qui prend en paramètre la proportion désirée :

# Architecture et conception du model

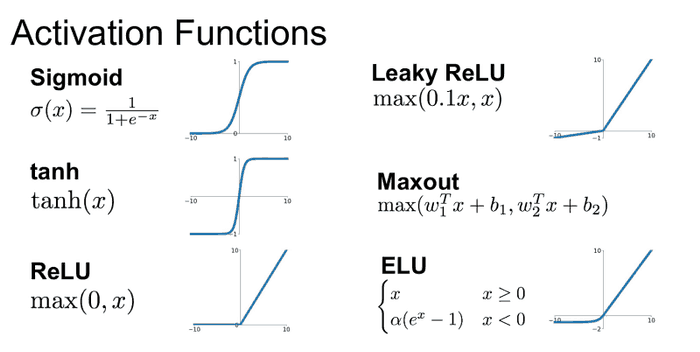
Pour le model on a opté pour utiliser un réseau de neurones (Neural Network) avec 4 couches (layers) avec le librairie Keras du Framework TensorFlow a cause de sa facilitée de l’utilisation et manipulation du Neural Networks.



Pour la premier partie du code, on a créé le réseaux ; la boucle cree les « 3 Hidden layers »,dans chaque itération du boucle, une couche est créer avec 8 neurones, et enfin la dernier couche est pour la prédication et contient 3 neurones qui indique les 3 classe de la classification.

Pour les argument du fonction : model.add()) :

* **input\_dim** : doit contenir le nombre de variable expliquatifs (4 dans notre cas)
* **Activation** : doit contenir une fonction mathematique, elle applique a chaque sortie du neurone, les meilleurs fonction a utiliser sont : relu, sigmoid, tanh,… . car ce sont les plus pratique



Pour la deuxième partie du code, c’est la partie de la configuration des paramètres du model, avec la fonction « model.compile() ».

Dans le processus de recherche des meilleurs poids (Matrice des weights), on trouve les deux fonctions essentielles des réseaux de neurones : Forwad Propagation et Backpropagation, les deux sont implémenter avec la version optimisée de « l’algorithme de ADAM » qui sert à accélérer le processus de la recherche du minimum de la « fonction du cout (Cost Function) »

# **Entrainement et Evaluation** :

* La fonction « model.fit() » entraine le model en utilisant l’échantillon du training set (X\_train,y\_train) , et aussi utilise la technique du « Mini Batch Gradient descent » pour accélérer la recherche du matrice des poids (W matrix), avec un taille de batch : 5 echantillons/batch. Aussi on fixe 50 itérations (epoche) pour trouver la meilleur matrice W. et on teste notre model sur (X\_test, y\_test)
* La fonction « model.evalute() » donne le pourcentage du bonne prédiction parmi toute les prédiction du testing Set

