Explication de chaque modèle

7.1 La régression logistique

La régression logistique est un algorithme d'apprentissage supervisé utilisé pour la classification. L'objectif principal de la régression logistique est de modéliser la probabilité qu'une observation appartienne à une catégorie particulière. Elle est particulièrement adaptée lorsque la variable dépendante est binaire.

Fonctionnement :

- La fonction de coût mesure la différence entre les valeurs prédites par le modèle et les valeurs réelles observées. Les coefficients du modèle sont ajustés pendant la phase d'entraînement à l'aide d'une technique d'optimisation (généralement la descente de gradient) pour minimiser la fonction de coût.
- La fonction sigmoïde est utilisée pour modéliser la probabilité que la variable dépendante soit égale à 1 en fonction des variables indépendantes.
- Une fois le modèle entraîné, il peut être utilisé pour prédire la probabilité qu'une nouvelle observation appartienne à la classe 1.

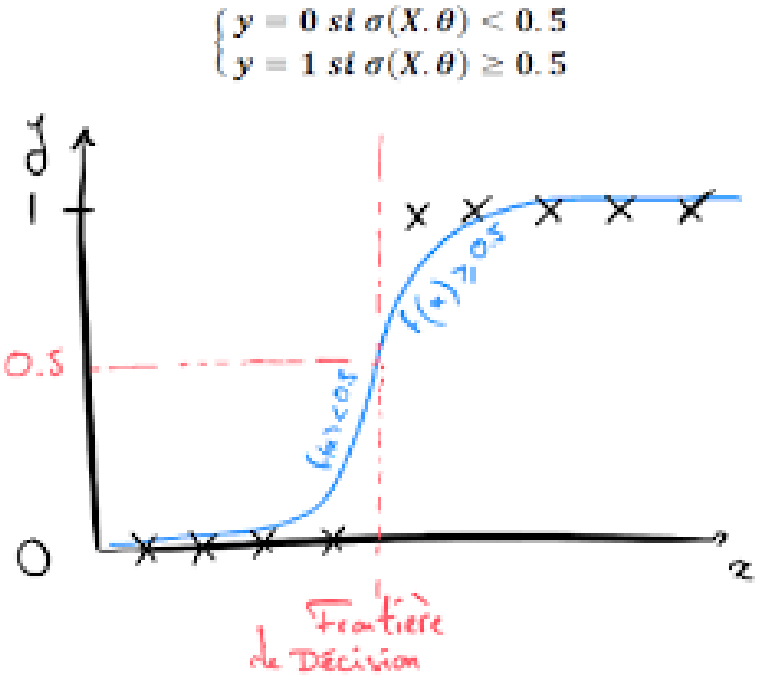

Figure 1 : Régression Logistique

Dans notre projet, nous définissons une classe LogisticRegressionModel() qui importe le classifieur LogisticRegression() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont notamment le terme inverse de régularisation C et l'algorithme à utiliser dans le problème d'optimisation.

7.2 Machine à vecteur de support

Les machines à vecteurs de support sont une classe d'algorithmes d'apprentissage supervisé utilisés pour la classification et la régression.

Fonctionnement :

Le principe fondamental des SVM est de trouver un hyperplan qui sépare au mieux les données en différentes classes. Si les données ne sont pas linéairement séparables, les SVM peuvent les transformer dans un espace de plus grande dimension grâce à une fonction appelée le noyau (kernel), permettant ainsi de trouver un hyperplan de séparation.

Les SVM cherchent à maximiser la marge, qui est la distance entre l'hyperplan de séparation et les points les plus proches de chaque classe, appelés vecteurs de support. Ces vecteurs de support sont cruciaux car ils déterminent l'emplacement et l'orientation de l'hyperplan.

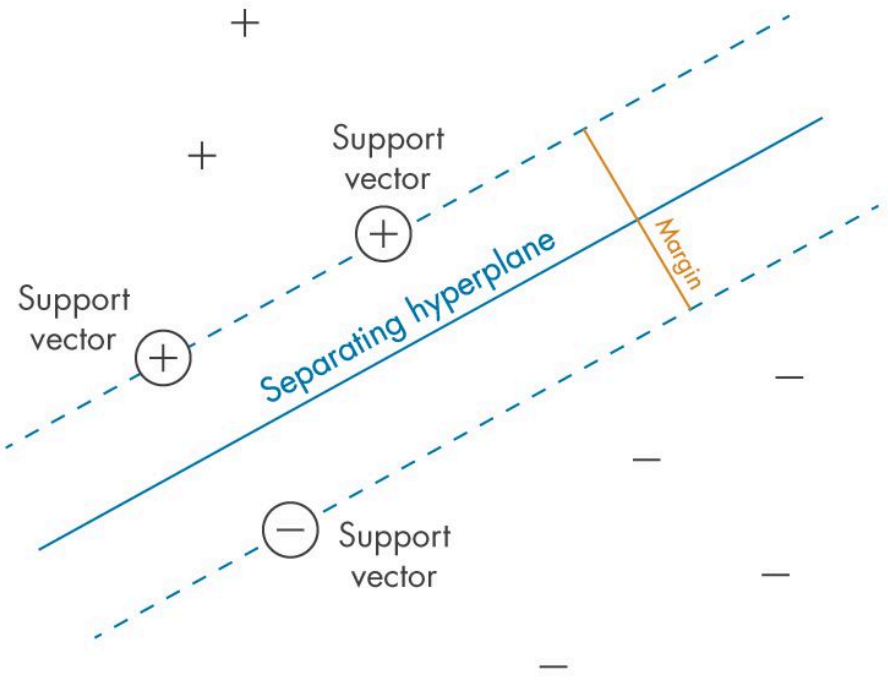

Figure 2 : Machine à vecteurs de support

Une fois l'hyperplan trouvé, les SVM peuvent être utilisées pour classer de nouvelles données en fonction de leur position par rapport à cet hyperplan.

Dans notre projet, nous définissons une classe SVMModel() qui importe le classifieur SVC() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont notamment le terme inverse de régularisation C, le type de noyau Kernel et gamma le coefficient du noyau.

7.3 Arbres de décision

Les arbres de décision sont des modèles d'apprentissage automatique utilisés pour résoudre des problèmes de classification et de régression.

Fonctionnement :

Les modèles d'arbres de décision prennent la forme d'une structure arborescente composée de nœuds, où chaque nœud représente une décision basée sur une caractéristique particulière. La décision de subdiviser ou non un nœud dépend d'une notion d'impureté (comme vu en cours). Si l'impureté est élevée alors on subdivise. Les feuilles de l'arbre contiennent les résultats de la classification ou de la prédiction.

Au fur et à mesure de l'apprentissage, les arbres de décision classifient les nœuds en utilisant les modèles de types "stumps" comme des classifieurs perpendiculaires à un axe. En combinant ces modèles on obtient des frontières de décision "crénelées". Ce type de modèle mène souvent au surapprentissage des données d'entraînement (nous y reviendrons dans le projet).

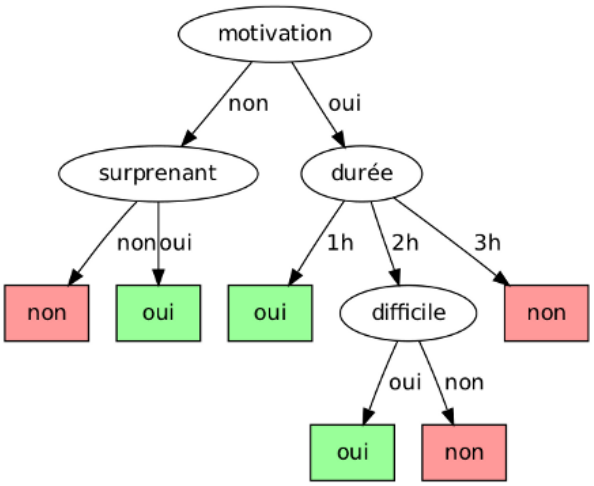

Figure 3 : Arbre de décision

Dans notre projet, nous définissons une classe DecisionTreeModel() qui importe le classifieur DecisionTreeClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le critère d'impureté (gini ou entropie) et la profondeur maximale de l'arbre.

7.4 Forêts aléatoires

Les forêts aléatoires sont une technique d'ensemble en apprentissage automatique qui combine plusieurs modèles d'arbres de décision pour améliorer la robustesse et la généralisation du modèle global. Comme énoncé précédemment, les arbres de décisions sont très souvent soumis au sur-apprentissage, les forêts aléatoires permettent de combiner les arbres de décisions à l'aide d'un vote majoritaire afin de réduire la variance des modèles.

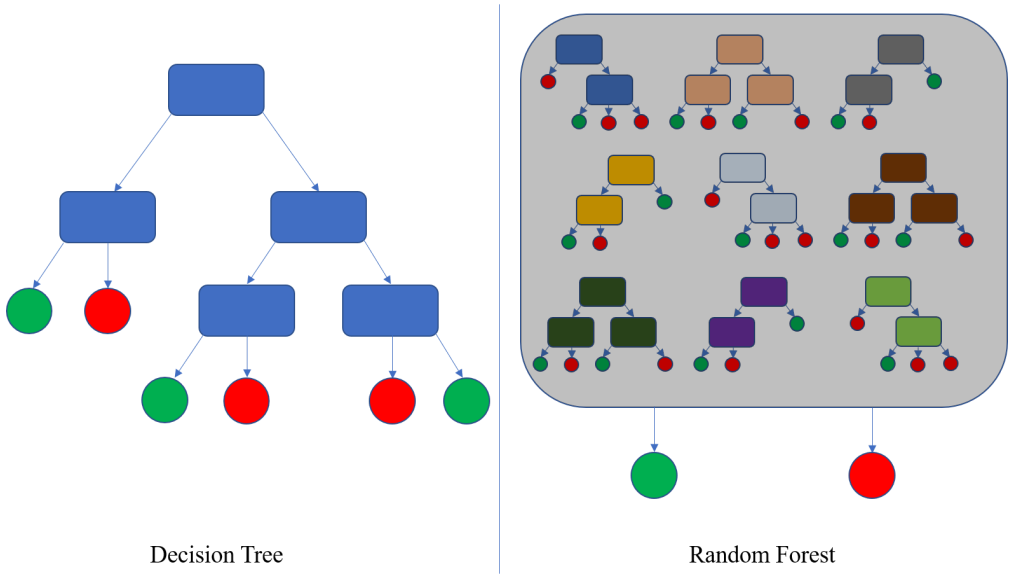

Figure 4 : Forêts aléatoire

Dans notre projet, nous définissons une classe RandomForestModel() qui importe le classifieur RandomForestClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le critère d'impureté (gini ou entropie), la profondeur maximale de l'arbre ainsi que le nombre d'arbres combinés pour effectuer la prédiction.

7.5 Perceptron multicouches

Le perceptron multicouche est un type d'architecture de réseau de neurones artificiels. C'est une extension du perceptron simple, introduisant plusieurs couches de neurones, d'où le terme « multicouche ». Ces réseaux sont également souvent appelés réseaux de neurones profonds lorsqu'ils ont plusieurs couches cachées.

Fonctionnement :

Les neurones sont les unités fondamentales du réseau, représentant des entités qui effectuent des opérations mathématiques sur les entrées.

Un perceptron multicouche est composé d'au moins trois couches : une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie. La couche d'entrée reçoit les données d'entrée, la couche de sortie produit la sortie finale du réseau, et les couches cachées effectuent des transformations intermédiaires. Chaque neurone est associé à une fonction d'activation qui détermine sa sortie en fonction des entrées.
Chaque connexion entre les neurones est associée à un poids qui ajuste l'impact de l'entrée sur la sortie du neurone. Ces poids sont appris pendant la phase d'entraînement du réseau.

Les données d'entrée sont propagées à travers le réseau de la couche d'entrée à la couche de sortie. Chaque neurone calcule sa sortie en utilisant les entrées pondérées et la fonction d'activation. C'est la forward pass.

L'algorithme de rétropropagation est ensuite utilisé pour ajuster les poids du réseau en fonction de l'erreur entre les prédictions du réseau et les valeurs réelles. Il utilise la dérivée de l'erreur par rapport aux poids pour mettre à jour les poids de manière à minimiser cette erreur. C'est la backward pass.

Une fois les poids déterminés, le perceptron multicouche peut effectuer les prédictions.

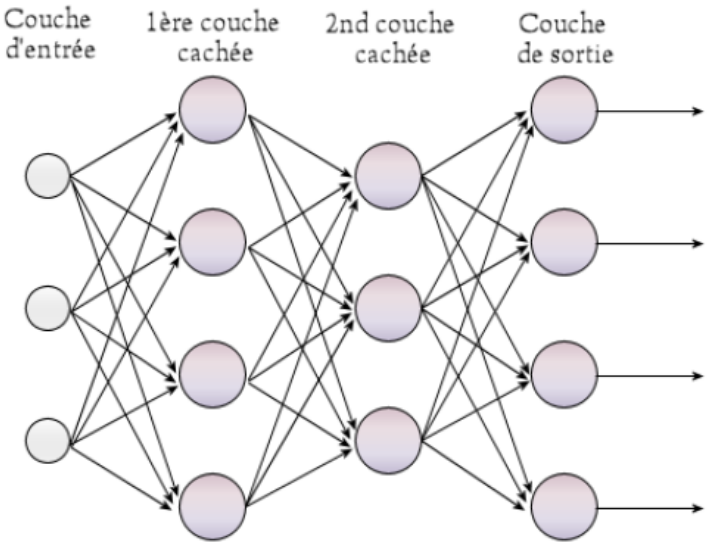

Figure 5 : Perceptron Multicouche

Dans notre projet, nous définissons une classe MLPModel() qui importe le classifieur MLPClassifier() de scikit-learn. De nombreux hyper-paramètres peuvent être testés, spécialement pour ce modèle, ce qui entraîne un algorithme grid search assez long. C'est pour cela que nous avons établi que la fonction d'activation que nous garderons est la ReLU car elle fait partie des plus performantes (à l'inverse de la sigmoid ou tanh qui mène à la disparition du gradient, c'est-à-dire que les poids ne sont plus actualisés au bout d'une certaine itération). D'autres choix on été fait comme une valeur de momentum égale à 0.9 ou adam comme solveur pour la descente de gradient.

Ces choix nous amènent à ne tester que ces deux hyperparamètres qui nous semblaient cruciaux à faire varier : le terme de régularisation (alpha) et les différentes couches cachées du réseau (hidden_layers_sizes). Cela nous permet d'éviter des temps trop longs lors de l'algorithme grid search.

7.6 Classification par algorithme des K plus proches voisins

L'algorithme des plus proches voisins est une méthode d'apprentissage supervisé utilisée à la fois pour la classification et la régression.

Fonctionnement :

Comme explicité précédemment, cet algorithme repose sur le principe que les données similaires ont tendance à se regrouper dans l'espace. Il fonctionne en trouvant les k voisins les plus proches d'un point donné dans un espace multidimensionnel, et en utilisant ces voisins pour effectuer une classification. Si la majorité des voisins sont présents dans la classe A alors le point sera classé dans la classe A.

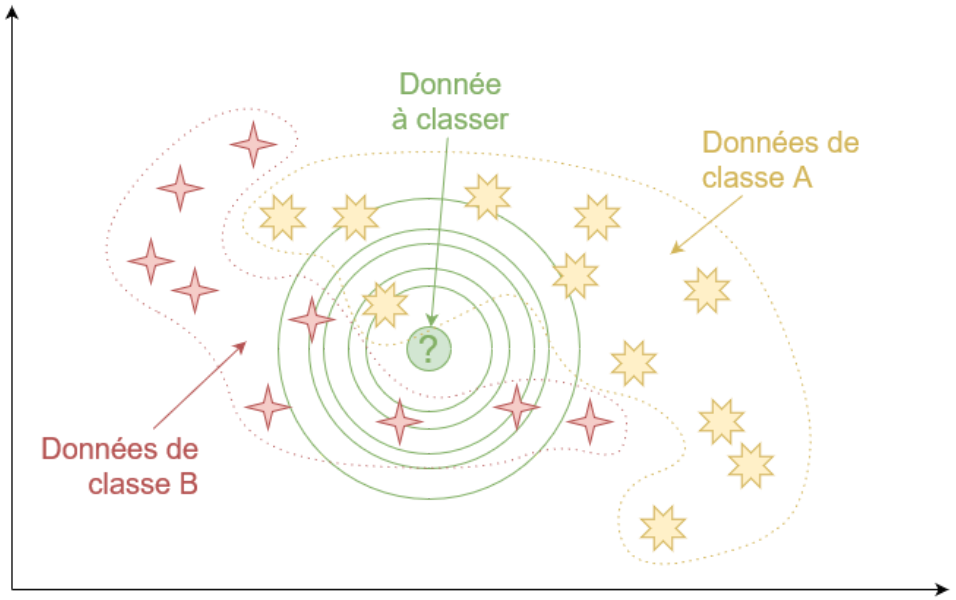

Figure 6 : K plus proches voisins

Dans notre projet, nous définissons une classe KNNModel() qui importe le classifieur KNNClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le nombre de voisins pris en compte lors de l'algorithme ainsi que la fonction de mesure des poids :

- uniforme, dans ce cas tous les points de chaque quartier sont pondérés de manière égale.
- distance, dans ce cas les voisins les plus proches auront une plus grande influence que les voisins plus éloignés.