MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique
Filière : Informatique

RAPPORT DE PROJET

Option :**Ingénierie des Systèmes d'Information**

THÈME

Système d'Aide au Diagnostic de la Maladie Coronarienne, Utilisation d'une Méthode de Bagging avec Régression Logistique

Etudiantes : Benhammou Mekkia
Belkaious Hayet

Devant le jury composé de :

Mme B. Kaid Slimane	Université de Mostaganem	Présidente
Mr K. Sehaba	Université de Mostaganem	Examinateur
Mr F. Henni	Université de Mostaganem	Encadreur

Année Universitaire 2020-2021

Dédicaces

«Une des clés du succès est la confiance en soi, une des clés de la confiance en soi est la préparation»

Arthur Ashe

À notre famille et à tous nos amis

Remerciements

Tout d’abord, nous tenons à remercier « ALLAH », Qui nous a donné la force, la volonté et le courage pour terminer ce travail.

Nos premiers remerciements à Monsieur Henni Fouad qui a accepté d’être notre encadreur, qui nous a guidées, et surtout pour la confiance qu’il ne cesse de nous témoigner.

Nos remerciements vont également aux membres de jury qui ont eu l’amabilité de lire et évaluer ce travail.

Un grand merci à nos parents pour leurs encouragements, soutient et patience.

Tout simplement à tous ceux et celles qui méritent nos remerciements.

MERCI ...

Résumé

En apprentissage automatique (Machine Learning, ML) un modèle ensembliste consiste en un ensemble de modèles « entrainés » (t.q. des réseaux de neurones, des arbres de décision…etc).

En classification, un modèle ensembliste combine les prédictions d'un ensemble de modèles pour une prédiction plus précise. Le bagging est une méthode ensembliste qui consiste à agréger les prédictions d'un nombre importants de modèles. Des recherches récentes ont montré que les modèles qui suivent un principe de bagging (tels que les forêts aléatoires) sont très performants.

L'objectif du projet est d'abord de faire un tour sur des méthodes ensemblistes (Bagging, Boosting, Stacking) ensuite de mettre l'accent sur le bagging.

Le principe de bagging sera appliqué à un dataset réel en utilisant le module Scikit Learn (Sklearn) sous Python. Des expérimentations seront menées afin d'évaluer les résultats obtenus.

Mots-clés:

Apprentissage automatique, méthodes Ensembliste, Bootstrapping, Bagging, Boosting, Stacking, Système d'aide à la décision, Maladie coronarienne, Sklearn

Abstract :

In machine learning (ML) an ensemble model consists of a set of "trained" models (such that neural networks or decision trees). In classification, an ensemble model combines the predictions of a set of models for a more precise prediction. Bagging is an ensemble method which consists in aggregating the predictions of a large number of models. Recent researches have shown that models that follow a bagging principle (such as random forests) are very efficient. The objective of this project is first to provide an overview of ensemble methods (Bagging, Boosting, Stacking) then to focus on bagging.

The principle of bagging will be applied to a real dataset using the Scikit Learn (Sklearn) module under Python. Experiments will be carried out to assess the results obtained.

Keywords:

Machine learning, Ensemble methods, Bootstrapping, Bagging, Boosting, Stacking, Decision making system, Coronary artery disease, Sklearn

Liste des figures

Liste des tableaux

Liste des abréviations

IA : Intelligence Artificielle

AA: Apprentissage Automatique

XGBoost : eXtreme Gradient Boosting

SCV: Stratified Cross-Validation

RF: Random Forest

CART: Classification And Regression Trees

CHAID : Chi-Squared Automatic Interaction Detection

MCO: Moindres Carrés Ordinaires

CAD : Coronary Artery Disease

Table des matières

Introduction Générale

Les systèmes d'aide à la décision médicale (SADM) sont des outils informatiques capables de traiter l'ensemble des caractéristiques d'un patient donné afin de générer les diagnostics probables de son état clinique (aide au diagnostic) ou les traitements qui lui seraient adaptés (aide à la thérapeutique). Le terme SADM recouvre aujourd'hui un ensemble d'outils variés, plus ou moins complexes.

Dans le domaine médical, les connaissances des experts ne sont pas basées uniquement sur des règles mais également sur des connaissances théoriques et sur leur expérience acquise. Les données mémorisées qui concernent les patients antérieurs peuvent être exploitées pour construire des modèles de prédiction qui peuvent aider les médecins dans l'établissement d'un diagnostic.

Les maladies cardiovasculaires ont été estimées comme étant la première cause de mortalité dans le monde. La maladie des artères coronaires, le type de maladie cardiaque le plus fréquent, fait référence au rétrécissement des artères qui fournissent du sang au cœur, provoqué par une accumulation d'une substance collante appelée plaque.

Le présent projet rentre dans le cadre des systèmes d'aide à la décision dans le domaine médical, et plus particulièrement pour diagnostiquer la maladie coronarienne avant qu'elle ne provoque des problèmes cardiaques sérieux. La prédiction est basée sur l'utilisation d'un dataset expérimental qui a déjà été utilisé dans plusieurs travaux de recherche. Il s'agit d'utiliser une méthode de classification pour diagnostiquer la présence ou l'absence de la maladie coronarienne en se basant sur un ensemble d'attributs.

La classification, comme étant une tâche qui répond au besoin d'un large type d'applications, nécessite d'être au cœur du datamining distribué. Dans le cadre de ce projet, nous mettons le point particulièrement sur les approches des méthodes ensemblistes (Bagging, Boosting et Stacking). Ces méthodes combinent un nombre important d'estimateurs afin de produire une estimation plus précise.

Afin d'aboutir à divers ensembles d'apprentissage qui seront utilisés pour construire les classificateurs de base, plusieurs techniques peuvent être appliquées, parmi les plus utilisées, on présente la technique de bagging. Ensuite, on s'intéresse à un bagging avec la régression logistique comme estimateur de base. Le bagging a pour but de réduire la variance de l'estimateur, et permettre la production de meilleures performances prédictives pour une estimation plus précise. La régression logistique est une méthode simple et efficace largement utilisée par les analystes de données et les scientifiques.

Ce mémoire est organisé comme suit : le chapitre 1 est consacré à la présentation du principe des méthodes ensemblistes. Le chapitre 2 met l'accent particulièrement sur le

bagging et ses caractéristiques et présente également la méthode de régression logistique que nous projetons d'utiliser comme estimateur de base dans un modèle basé sur le bagging. Le chapitre 3 est consacré à la construction de notre modèle en utilisant le module « Scikit-Learn » sous Python. Les expérimentations seront présentées également dans ce chapitre. Une conclusion générale vient clôturer ce mémoire.

Chapitre 1

Les Méthodes Ensemblistes

1. Introduction :

L'apprentissage automatique (AA), ou machine-learning, est une forme d'intelligence artificielle (IA) qui permet à un système d'apprendre à partir des données et non à l'aide d'une programmation explicite. Cependant, l'apprentissage automatique n'est pas un processus simple. Au fur et à mesure que les algorithmes ingèrent les données de formation, il devient possible de créer des modèles plus précis basés sur ces données. Un modèle de machine-learning est le résultat généré lorsque vous entraînez votre algorithme.

En classification, un modèle ensembliste combine les prédictions d'un ensemble de modèles pour une prédiction plus précise.Dans ce chapitre, on présente le principe des méthodes ensembliste (Bagging, Boosting, Stacking) qui consiste à agréger les prédictions d'un nombre importants de modèles.

2. Le Datamining :

2.1 Définition :

Le terme Datamining désigne l'analyse de données depuis différentes perspectives et le fait de transformer ces données en informations utiles, en établissant des relations entre les données ou en repérant des patterns. Ces informations peuvent ensuite être utilisées par les entreprises pour augmenter un chiffre d'affaires ou pour réduire des coûts [1].

2.2 Fonctionnement du Datamining :

Les technologies informatiques ont évolué de manière à ce que les systèmes transactionnels et les systèmes analytiques soient séparés. Le datamining assure la jonction entre les deux. Les logiciels de datamining analysent les relations et les patterns des données de transactions stockées en se basant sur des requêtes d'utilisateurs. Plusieurs types de logiciels analytiques sont disponibles : statistiques, Machine Learning, et réseaux neuronaux. En général, on dénombre quatre types de relations [1]:

- **Classe :**

Les données stockées sont utilisées pour localiser les données en groupes prédéterminés.

- **Cluster :**

Les données sont regroupées par rapport à des relations logiques ou aux préférences des clients. Par exemple, les données peuvent être minées pour identifier des segments de marché ou des affinités de clients.

- **Associations**:

Les données peuvent être minées pour identifier des associations.

- **Patterns séquentielles :**

Les données sont minées pour anticiper les patterns de comportements et les tendances.

2.3 Les objectifs des méthodes de Datamining :

On peut regrouper les objectifs des méthodes de Data Mining en quatre grandes fonctions :

- **Classifier:**

On examine les caractéristiques d'un nouvel objet pour l'affecter à une classe prédéfinie. Les classes sont bien caractérisées et on possède un fichier d'apprentissage avec des exemples pré classés. On construit alors une fonction qui permettra d'affecter à telle ou telle classe un nouvel individu.

- **Estimer:**

La classification se rapporte à des événements discrets (par exemple : le patient a été ou non hospitalisé). L'estimation, elle, porte sur des variables continues (par exemple : la durée d'hospitalisation). Dans ce contexte, on parle aussi de régression.

- **Segmenter:**

Il s'agit de déterminer quelles observations vont naturellement ensemble sans privilégier aucune variable. On segmente une population hétérogène en un certain nombre de sous-groupes plus homogènes (les clusters). Dans ce cas, les classes ne sont pas prédéfinies.

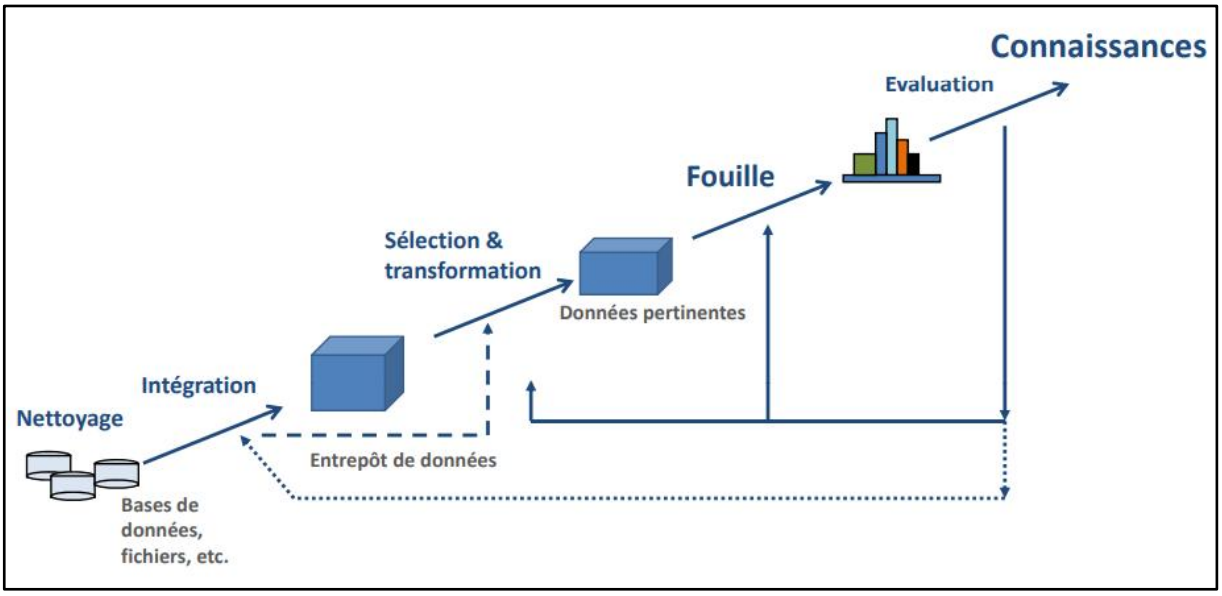

Figure1.1 *:* Processus du datamining [2]

- **Prédire :**

Cette fonction est proche de la classification ou de l'estimation, mais les observations sont classées selon un comportement ou une valeur estimée futurs. Les techniques précédentes

peuvent être adaptées à la prédiction au moyen d'exemples d'apprentissage où la valeur à prédire est déjà connue. Le modèle, construit sur les données d'exemples et appliqué à de nouvelles données, permet de prédire un comportement futur.

3. Limites des modèles uniques

L'Overfitting (sur-ajustement) et l'Underfitting (sous-ajustement) sont les causes principales des mauvaises performances des modèles prédictifs générés par les algorithmes deMachine Learning.

3.1 L'Overfitting

Désigne le fait que le modèle prédictif produit par l'algorithme de Machine Learning s'adapte bien au*Training Set.* Le modèle prédictif capturera tous les "aspects" et détails qui caractérisent les données du *Training Set.* Dans ce sens, il capturera toutes les fluctuations et variations aléatoires des données du *Training Set*. En d'autres termes, le modèle prédictif capturera les corrélations généralisables et le bruit produit par les données. Dans ce cas, on dit aussi que le modèle est « trop » taillé sur le *Training Set*, et ses prédictions sur de nouvelles données ne seront pas bonnes.

3.2 L'Underfitting

Sous-entend que le modèle prédictif généré lors de la phase d'apprentissage, s'adapte mal au *Training Set.* Le modèle prédictif n'arrive même pas à capturer les corrélations du *Training Set.* Par conséquent, le coût d'erreur en phase d'apprentissage reste grand. Bien évidemment, le modèle prédictif ne se généralisera pas bien, non plus sur les données qu'il n'a pas encore vues. Finalement, le modèle ne sera viable car les erreurs de prédictions seront grandes.

4 Les méthodes ensemblistes :

4.1 Définition :

Une méthode ensembliste modélise l'apprentissage réalisé par un ensemble d'algorithmes qui est très utilisé en « data-science ». La force des techniques d'apprentissage d'ensemble réside dans le fait qu'elles combinent les prédictions de plusieurs modèles simples d'apprentissage automatique.

4.2 Les différents types de techniques d'apprentissage d'ensemble :

Il existe des techniques d'apprentissage d'ensemble simples et avancées :

4.2.1 Techniques d'ensemble simples :

4.2.1.1 Vote majoritaire :

La méthode de vote majoritaire est généralement utilisée pour les problèmes de classification. Dans cette technique, plusieurs modèles sont utilisés pour faire des prédictions pour chaque point de données. Les prédictions de chaque modèle sont considérées comme un« vote ».La prédiction obtenue de la part de la majorité des modèles est utilisée comme prédiction finale.

4.2.1.2 Moyenne :

Dans cette méthode, la moyenne des prédictions de tous les modèles est utilisée pour faire la prédiction finale. La moyenne peut être utilisée pour faire des prédictions dans des problèmes de régression ou pour calculer des probabilités pour des problèmes de classification.

4.2.1.3 La Moyenne pondérée :

Il s'agit d'une extension de la méthode de calcul de la moyenne. Tous les modèles se voient attribuer des poids différents définissant l'importance de chaque modèle pour la prédiction.

4.2.2 Techniques d'ensemble avancées :

4.2.2.1 Le Bagging :

Proposé par Breiman en 1996, le principe du Bagging est de combiner les résultats de plusieurs modèles (par exemple, plusieurs arbres de décision) pour obtenir un résultat généralisé.

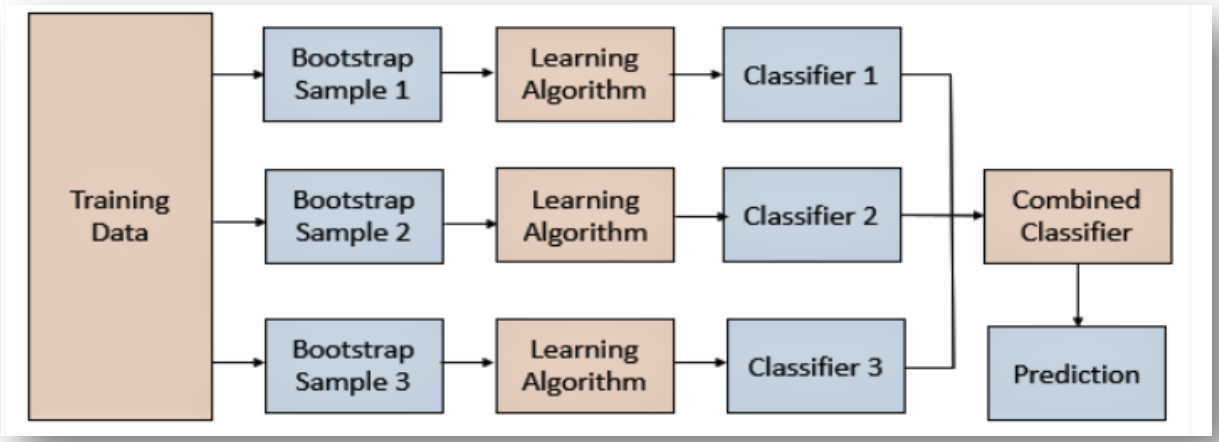

Figure1.2 : Principe du Bagging (Bootstrap Aggregation) [3]

-Algorithmes basés sur le bagging :

- Bagging meta-estimator
- Random forest

-Avantages et inconvénients du bagging :

Random forest est l'un des algorithmes bagging les plus populaires. Le bagging offre l'avantage de permettre à de nombreux apprenants faibles de combiner leurs efforts pour

surpasser un seul apprenant. Il aide également à réduire la variance, éliminant ainsi le sur-ajustement des modèles dans la procédure.

- Un inconvénient de bagging est qu'il introduit une perte d'interprétabilité d'un modèle.
- Le modèle résultant peut subir de nombreux biais lorsque la procédure appropriée est ignorée.
- Bien que bagging soit très précis, il peut être coûteux en calcul et cela peut décourager son utilisation dans certains cas.

4.2.2.2 Le Boosting :

Proposé par Freund et Shapire en 1996,le renforcement (Boosting)est un processus séquentiel, où chaque modèle suivant tente de corriger les erreurs du modèle précédent. Les modèles suivants dépendent du modèle précédent.

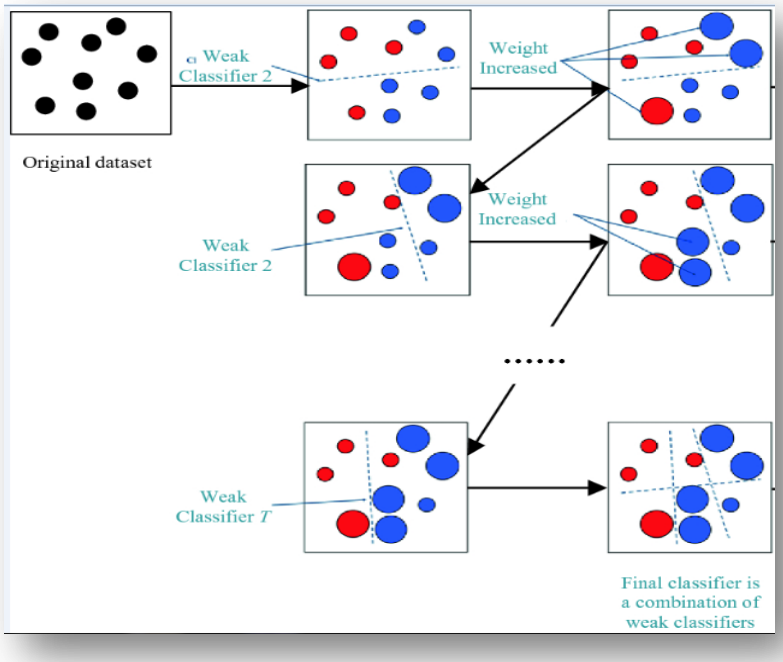

Figure 1.3 : Principe de la technique de Boosting [4]

-Fonctionnement du boosting :

a) Un sous-ensemble est créé à partir de l'ensemble de données d'origine.
b) Au départ, tous les points de données reçoivent des poids égaux.
c) Un modèle de base est créé sur ce sous-ensemble.
d) Ce modèle est utilisé pour faire des prédictions sur l'ensemble des données
e) Les erreurs sont calculées à l'aide des valeurs réelles et des valeurs prédites.
f) Les observations qui sont mal prédites reçoivent des poids plus élevés

g) Un autre modèle est créé et des prédictions sont effectuées sur l'ensemble de données.(Ce modèle tente de corriger les erreurs du modèle précédent)
h) De même, plusieurs modèles sont créés, chacun corrigeant les erreurs du modèle précédent.
i) Le modèle final (apprenant fort) est la moyenne pondérée de tous les modèles (apprenants faibles)

L'algorithme de Boosting combine un certain nombre d'apprenants faibles pour former un apprenant fort. Les modèles individuels ne fonctionneraient pas bien sur l'ensemble de données, mais ils fonctionnent bien pour une partie de l'ensemble de données. Ainsi, chaque modèle améliore en fait la performance de l'ensemble.

Algorithmes basés sur le Boosting :

- **AdaBoost :**

Le boosting adaptatif ou AdaBoost est l'un des algorithmes de boosting les plus simples. Habituellement, les arbres de décision sont utilisés pour la modélisation. Plusieurs modèles séquentiels sont créés, chacun corrigeant les erreurs du dernier modèle. AdaBoost attribue des poids aux observations qui sont incorrectement prédites.

Les étapes pour exécuter l'algorithme AdaBoost:

a) Audépart, toutes les observations de l'ensemble de données reçoivent des poids égaux.
b) Un modèle est construit sur un sous-ensemble de données.
c) En utilisant ce modèle, des prédictions sont faites sur l'ensemble de données.
d) Les erreurs sont calculées en comparant les prédictions et les valeurs réelles.
e) Lors de la création du modèle suivant, des poids plus élevés sont attribués aux points de données qui ont été prédits de manière incorrecte.
f) Les poids peuvent être déterminés à l'aide de la valeur d'erreur. Par exemple, plus l'erreur est élevée, plus le poids attribué à l'observation.
g) Ce processus est répété jusqu'à ce que la fonction d'erreur ne change pas ou que la limite maximale du nombre d'estimateurs soit atteinte.

- **GBM :**

Gradient Boosting ou GBM est un autre algorithme d'apprentissage automatique d'ensemble qui fonctionne à la fois pour les problèmes de régression et de classification.GBM utilise la technique de renforcement, combinant un certain nombre d'apprenants faibles pour former un apprenant fort.

- **XGBoost :**

XGBoost (eXtreme Gradient Boosting) est une implémentation avancée de l'algorithme de boosting de gradient. XGBoost s'est avéré être un algorithme de ML très efficace, largement utilisé dans les compétitions d'apprentissage automatique et les hackathons. XGBoost a une puissance prédictive élevée et est presque 10 fois plus rapide que les autres techniques d'amplification de gradient. Il comprend également une variété de régularisation qui réduit le sur-ajustement et améliore les performances globales.

- **CatBoost :**

CatBoost peut traiter automatiquement les variables catégorielles et ne nécessite pas de prétraitement approfondi des données comme les autres algorithmes d'apprentissage automatique.

4.2.2.3 Le Stacking :

Le Stacking (ou bien l'empilement) est une technique d'apprentissage d'ensemble qui utilise les prédictions de plusieurs modèles (par exemple arbre de décision) pour construire un nouveau modèle. Ce modèle est utilisé pour faire des prédictions sur l'ensemble des données.

Il utilise un algorithme de méta-apprentissage pour apprendre à combiner au mieux les prédictions de deux ou plusieurs algorithmes d'apprentissage automatique de base.

L'avantage du Stacking est qu'il peut exploiter les capacités d'une gamme de modèles performants sur une tâche de classification ou de régression et faire des prédictions qui ont de meilleures performances que n'importe quel modèle unique de l'ensemble [5].

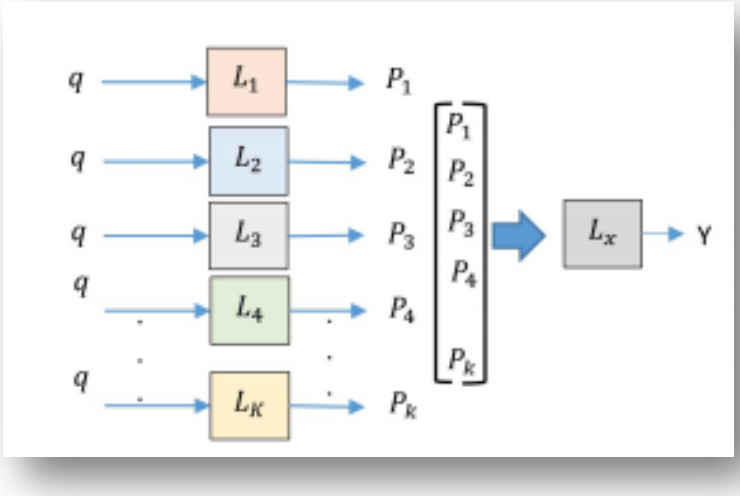

Figure 1.4 : Classificateur basé sur l'empilement (stacking) général [6]

5 Evaluation d'un modèle :

Evaluer la performance d'un modèle est une étape importante de tout projet de Machine Learning. Il faut pouvoir mesurer la capacité de généralisation du modèle sans introduire de biais, ni fuites de données. Il est commun de diviser le dataset en données d'entrainement, de validation et de test.

5.1 La Validation croisée :

La validation croisée (ou cross-validation, en anglais) est une méthode statistique qui permet d'évaluer la capacité de généralisation d'un modèle. Il s'agit d'une méthode qui est plus stable et fiable que celle d'évaluer la performance sur des données réservées pour cette tache (Hold-out Validation).

- Utiliser l'intégralité de notre jeu de données pour l'entraînement et pour la validation
- Découper les données en k *parties*
- Tout à tour, chacune des k parties est utilisée comme jeu de test et le reste est utilisé pour l'entraînement.

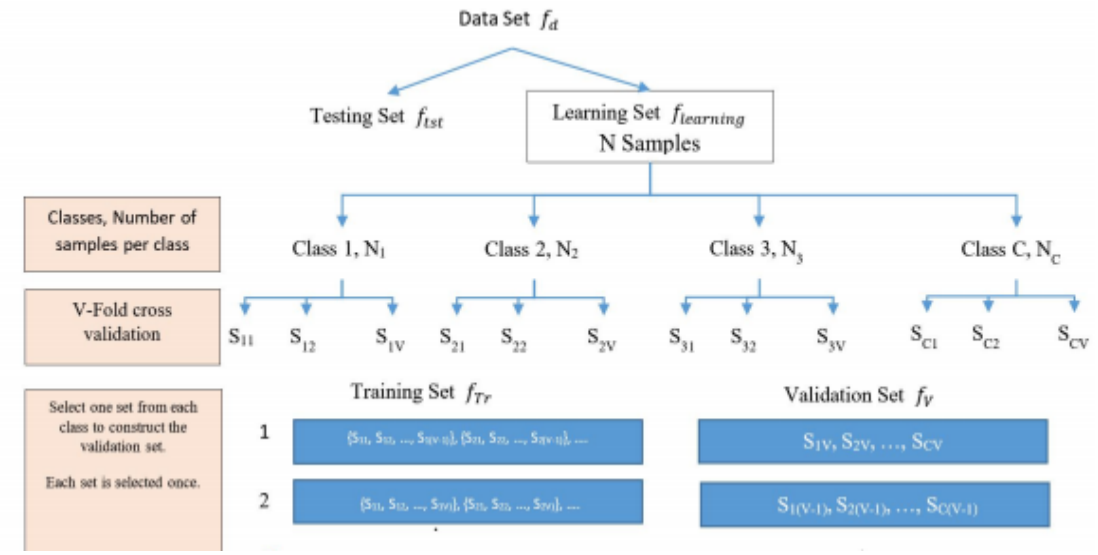

Figure 1.5 : Validation croisée standard [6]

-Les types de validation croisée :

- **Stratified Cross-Validation (SCV) :**

Pour bien comprend (SCV) en prend un exemple simple. Le vecteur y qui représente les labels dans un projet de Machine learning est comme suit : y = [0,0,0,1,1,1,2,2,2]. Il s'agit d'un problème de classification avec 3 classes. Malheureusement, les données de chaque classe sont alignées. Et donc, si on faisait un 3-fold cross-validation, notre modèle s'entraînera uniquement sur 2 classes et sera testé sur des données d'une classe qu'elle n'a jamais vu. On a là un gros risque de mauvaise performance qui est dû à la disposition de nos données. Nous souhaitons que chaque classe soit représenté dans chaque partie ou fold lors

d'une validation croisée. C'est pour cela qu'on opte pour un SCV dans ce cas. Avec un SCV, on aura donc les folds suivants : [0,1,2] [0,1,2] [0,1,2] au lieu de [0,0,0] [1,1,1] [2,2,2] lors d'un simple 3-fold cross-validation.

- **Leave-one-out cross-validation:**

Si nous choisissons de diviser le dataset en parties contenant uniquement un seul point. Autrement dit, nous ferons le test de notre modèle sur une seule observation à chaque itération. Cela signifie que si dans le dataset, nous avons 100 observations, nous entrainons et testons le modèle pour 100 itérations.

5.2 Performance d'un modèle :

La matrice de confusion résume les performances d'un modèle lorsqu'il est utilisé en prédiction. La structure de cette matrice pour un problème de classification à deux classes est la suivante :

		Classe prédite	
		Classe 1	Classe 0
Classe réelle	Classe 1	TP	FN
	Classe 0	FP	TN

Tableau 1.6 : Matrice de confusion

• TP (True Positive, Vrai positif) : Elément de la classe 1 correctement prédit ;

• TN (True Negative, Vrai négatif) : Elément de la classe 0 correctement prédit ;

• FP (False Positive, Faux positif) : Elément de la classe 1 mal prédit ;

• FN (False Negative, Faux négatif) : Elément de la classe 0 mal prédit.

A partir de la matrice de confusion, nous pouvons calculer les principales mesures de performance du classifieur. Il s'agit de :

- **Rappel** (ou la **sensitivité**) : Proportion des éléments bien classés parmi ceux qui sont positifs.

$$Rappel = \frac{TP}{TP + FN}$$

- **Spécificité** : Proportion des éléments bien classés parmi ceux qui sont négatifs.

$$Spécificité = \frac{TN}{TN + FP}$$

- **Précision** : C'est la probabilité qu'un individu prédit positif le soit réellement.

$$Précision = \frac{TP}{TP + FP}$$

- **Accuracy** : Proportion des éléments bien classés.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Chacune des mesures plus haut est une valeur entre 0 et 1. Il est clair que plus la mesure est proche de 1 plus le classifieur est meilleur. Il est courant de multiplier ces mesures par 100 pour les représenter sous forme de pourcentage.

A partir de la sensitivité et la précision, on calcul le « FScore » qui représente la moyenne harmonique entre la sensitivité et la précision (valeur entre 0 et 1).

$$FScore = \frac{2 \times Sensitivité \times Précision}{Sensitivité + Précision}$$

La courbe ROC (Receiver Operating Characteristic) peut ensuite être tracée. Elle représente le compromis (tradeoff) entre la sensitivité et la spécificité. Finalement, l'AUC (Area Under the roc Curve) est claculé. Il est largement admis que l'AUC résume globalement les performances du classifieur.

6. Conclusion

Ce chapitre a permis de passer en revue les objectifs du datamining avant de mettre l'accent sur les méthodes ensemblistes et ce qu'elles apportent par rapport aux modèles basés sur un algorithme unique. Nous continuons avec le principe de Bagging.

Chapitre 2

LeBagging

2.1 Introduction

Les techniques de classification sont une partie importante des applications d'apprentissage automatique et d'exploration de données .Environ 70% des problèmes en Data Science sont des problèmes de classification. Il existe de nombreux problèmes de classification. Dans ce chapitre, nous nous intéressons d'abord à donner plus de détails sur le Bagging, objet de ce rapport. Ensuite, nous introduisons la régression logistique en tant que modèle simple très utilisé dans les problèmes de classification. Notre intérêt à la régression logique s'explique par le fait que ce modèle peut être utilisé comme modèle de base dans un Bagging.

2.2 Bagging

L'utilisation du bagging est adaptée aux algorithmes à fortes variance qui sont ainsi stabilisés (réseaux neuronaux, arbres de décision pour la classification ou la régression…). Le bagging est une technique utilisée pour améliorer la classification notamment celle des arbres de décision, considérés comme des « classifieurs faibles », c'est-à-dire à peine plus efficaces qu'une classification aléatoire.

Le bagging a pour but de réduire la variance de l'estimateur, en d'autres termes de corriger l'instabilité des arbres de décision (le fait que de petites modifications dans l'ensemble d'apprentissage entraînent des arbres très différents). Pour ce faire, le principe du Bootstrap (ou tirage avec remise) est de créer de « nouveaux échantillons » par tirage au hasard dans l'ancien échantillon, avec remise. L'algorithme de base, par exemple l'arbre de décision, est entraîné sur ces sous-ensembles de données. Les estimateurs ainsi obtenus sont moyennés (lorsque les données sont quantitatives, cas d'un arbre de régression) ou utilisés pour un « vote » à la majorité (pour des données qualitatives, cas d'un arbre de classification). C'est la combinaison de ces multiples estimateurs « indépendants » qui permet de réduire la variance. Toutefois, chaque estimateur est entrainé avec moins de données. En pratique, le bagging donne d'excellents résultats.

2.2.1 Formulation de l'algorithme du Bagging

L'algorithme est simple à implémenter : il suffit de construire B estimateurs sur des échantillons Bootstrap et de les agréger. Le fait de considérer des échantillons Bootstrap introduit un aléa supplémentaire dans l'estimateur.

Algorithme du bagging :

Entrées :

- x l'observation à prévoir
- un estimateur de base (arbre de décision, KPPV, régression logistique, …..)
- d_n l'échantillon
- B le nombre d'estimateurs que l'on agrège.

Début :

Pour k = 1, …, B :

1. Tirer un échantillon Bootstrap d_n^k dans d_n
2. Ajuster l'estimateur sur cet échantillon Bootstrap : m_k

Fin

Sortie : L'estimateur$\widehat{m}_B$ (agrégation des estimateurs m_k, k=1, 2, .., B) – Moyenne pour un problème de régression et vote majoritaire pour un problème de classification.

2.2.2 La Technique Bootstrap

Le bootstrapping est une technique d'échantillonnage dans laquelle nous créons dessous-ensembles d'observations à partir de l'ensemble de données d'origine, avec remplacement (tirage avec remise).La technique de Bagging utilise ces sous-ensembles pour avoir une idée juste de la distribution. La taille des sous-ensembles créés pour le Bagging est inférieure à la taille de l'échantillon d'origine.

Quelques remarques sur le principe u Bagging :

- Plusieurs sous-ensembles sont créés à partir du jeu de données d'origine, en sélectionnant les observations avec remplacement.
- Un modèle de base (modèle faible) est créé sur chacun de ces sous-ensembles.
- Les modèles fonctionnent en parallèle et sont indépendants les uns des autres.
- Les prédictions finales sont déterminées en combinant les prédictions de tous les modèles.

2.2.3 Le classifieur de base

Les arbres de décisions ont été les premiers à être combinés dans un algorithme de Bagging. Plusieurs algorithmes permettant de construire des arbres de décision, dont : CART (Classification And Regression Trees),CHAID (Chi-Squared Automatic Interaction Detection) et C5.0 et ses prédécesseurs ID3 et C4.5. Dans le cas où le classifieur de base est un arbre de décision on parle de forêt aléatoire (Random Forest).

2.2.4 Cas particulier du Bagging - Random Forest(RF):

L'algorithme des « forêts aléatoires » (ou Random Forest, parfois aussi traduit par forêt d'arbres décisionnels) est un algorithme de classification/régression qui réduit la variance des prévisions d'un arbre de décision unique, améliorant ainsi les performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging.

L’algorithme des « forêts aléatoires » a été proposé par Leo Breiman et Adèle Cutler en 2001. Dans sa formule la plus classique, il effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et entraînés sur des sous-ensembles de données différents. Le nombre idéal d’arbres est un paramètre important : il est très variable et dépend du problème.

Concrètement, chaque arbre de la forêt aléatoire est entrainé sur un sous-ensemble aléatoire de données selon le principe du bagging, avec un sous-ensemble aléatoire de « features » (caractéristiques des données, ou descripteurs) selon le principe des « projections aléatoires ». Les prédictions sont ensuite moyennées lorsque les données sont quantitatives ou utilisés pour un vote pour des données qualitatives, dans le cas des arbres de classification. L’algorithme des forêts aléatoires est connu pour être un des classifieurs les plus efficaces « out-of-the-box » (c’est-à-dire nécessitant peu de prétraitement des données).

2.3 La Régression logistique

2.3.1 Définition :

La régression logistique [7] est un modèle statistique permettant d’étudier les relations entre un ensemble de variables qualitatives Xi et une variable qualitative Y. Il s’agit d’un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien. Un modèle de régression logistique permet aussi de prédire la probabilité D’occurrence d’un événement.

Notre intérêt pour la régression logistique est motivé par le fait qu’elle est permet de modéliser des variables binaires ou des sommes de variables binaires. Elle est très utilisée dans le domaine médical (guérison ou non d'un patient), en sociologie, en épidémiologie, en marketing quantitatif (achat ou non de produits ou services suite à une action) et en finance pour modélisation de risques (scoring).

2.3.2 Modèle de classification :

Un modèle de classification est un modèle qui classe chaque instance d’un jeu de données dans une catégorie. On distingue deux grands types de modèle de classification :

- ✓ Binaire : il n’y a que deux catégories possibles, par exemple : la potabilité d’un échantillon d’eau(potable ou non potable).
- ✓ Multi-classe : il y a au moins trois catégories, par exemple : la météo un jour donné (pluie, soleil ou neige).

La régression logistique peut s’appliquer à ces deux types de classification

2.3.3 Principe de la régression logistique :

Le principe du modèle de la régression logistique est de relier la survenance ou la non survenance d'un événement au niveau de variables explicatives. Par exemple, dans le domaine phytosanitaire, on cherche à évaluer à partir de quelle dose d'un agent chimique, un insecte sera neutralisé.

Pour bien comprendre la régression logistique, il faut introduire le concept de relation linéaire.

Le concept de relation linéaire :

Équation de régression linéaire:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots\cdots + \beta_n x_n$$

Où, y est la variable dépendante et x_1, x_2, ..., x_n sont des variables explicatives.

Régression linéaire Vs. Régression logistique :

La régression linéaire donne une sortie continue mais la régression logistique donne une sortie discontinue. Un exemple de production continue est le prix du logement et le prix de l'action. La régression linéaire est estimée à l'aide des moindres carrés ordinaires (MCO) tandis que la régression logistique est estimée à l'aide de l'approche de l'estimation du maximum de vraisemblance (MLE).

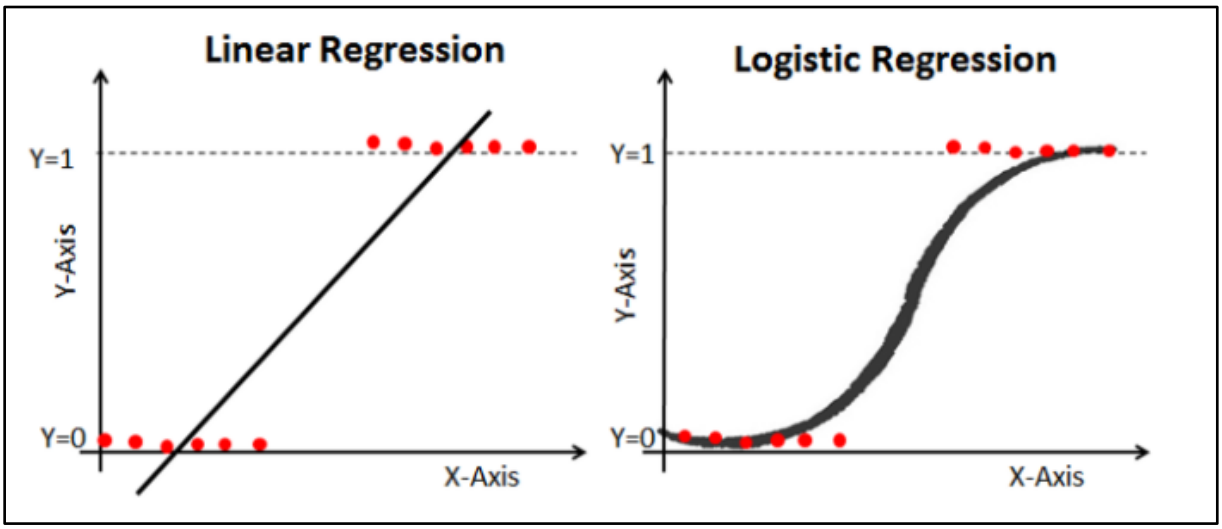

Figure 2.1 Régression linéaire Vs. Régression logistique [8]

La fonction sigmoïde :

La fonction sigmoïde également appelée fonction logistique donne une courbe en forme de (S)qui peut prendre n'importe quel nombre à valeur réelle et donner en retour une valeur comprise entre 0 et 1. Si la courbe va à l'infini positif, y prédit deviendra 1 et si le la courbe va à l'infini négatif, y prédit deviendra 0. Si la sortie de la fonction sigmoïde est supérieure à 0.5, nous pouvons classer le résultat comme 1 ou OUI et s'il est inférieur à 0.5, nous pouvons le classer comme 0 ou NON

Fonction sigmoïde : $f(x) = \frac{1}{1+e^{-(x)}}$

Propriétés de la régression logistique:

- La variable dépendante de la régression logistique suit la distribution de Bernoulli.
- L'estimation se fait par maximum de vraisemblance.

- Pas de carré R, la forme physique du modèle est calculée par Concordance, KS-Statistics.

- **2.3.4 Exemple :**

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0

Tableau 2.1 : jeux de donné [9]

Dans le jeu de données ci-dessus, chaque colonne détaille plusieurs informations recueillies sur un ensemble de patientes d'un hôpital. Ainsi, pour chaque patiente examinée, on dispose de plusieurs indicateurs médicaux, on retrouve notamment l'âge de la patiente et encore son taux de glucose dans le plasma sanguin.

La colonne qui se détache des autres ici est la colonne « *Outcome* » qui indique si la patiente examinée souffre de diabète. Cette variable est binaire, elle vaut 1 si la patiente souffre de diabète, 0 sinon.

2.3.5 Illustration de la régression logistique

La régression logistique en pratique est assez simple, en Python on utilise la classe « *LogisticRegression* » du module « *sklearn.linear_model* » comme un classificateur normal et que l'on entraîne sur des données déjà nettoyées et séparées en ensembles d'entraînement et de test.

2.3.6 Avantages et Désavantages

- **Avantages**

En raison de sa nature simple et efficace, la régression logistique ne nécessite pas de puissance de calcul élevée, facile à mettre en œuvre, facilement interprétable, largement utilisée par les analystes de données et les scientifiques. Elle ne nécessite pas non plus de mise à l'échelle des fonctionnalités.

La régression logistique fournit un score de probabilité pour les observations.

- **Désavantages :**

La régression logistique n'est pas capable de gérer un grand nombre de caractéristiques (variables catégorielles). Elle est vulnérable au sur-ajustement (overfitting). De plus, elle ne peut pas résoudre le problème non linéaire avec la régression logistique, c'est pourquoi elle nécessite une transformation de caractéristiques non linéaires.

La régression logistique ne fonctionne pas bien avec des variables indépendantes qui ne sont pas corrélées à la variable cible et très similaires ou corrélées les unes aux autres.

2.4 Un Bagging avec Régression logistique comme classifieur de base

La régression logistique est une technique très populaire. Les bases de la régression logistique, dans un premier temps : test de significativité des coefficients, intervalles de confiances, prédiction ; et dans un second temps : l'interprétation des coefficients sous forme d'odds-ratio, les différents types de codage des variables explicatives et leur impact sur l'interprétation, les stratégies de sélection de variables, résoudre le problème.

Ces caractéristiques motivent la conception d'un bagging comme modèle de base avec la régression logistique. Le bagging peut pallier à certains inconvénients de la régression logistique comme le sur-ajustement (overfitting).

2.5 Conclusion :

Le but de ce chapitre était d'abord de détailler le principe du Bagging, ensuite de présenter brièvement la régression logistique en tant que méthodologie simple, efficace et très populaire. Ce qui est très intéressant dans la régression logistique et de manière plus générale dans les modèles d'apprentissage automatique est que l'algorithme n'a pas besoin de connaître de règles théoriques liant l'entrée du modèle et sa sortie. Le modèle de machine learning apprend tout seul ces règles en se basant uniquement sur les données.

Chapitre 3

Un Bagging avec une Régression Logistique

3.1. Introduction

L'objectif de ce chapitre est d'expérimenter un bagging avec une régression logistique comme estimateur de base. Le modèle construit sera utilisé pour prédire l'absence ou la présence de la maladie coronarienne chez un patient. Le modèle sera construit en utilisant un dataset existant et qui a été utilisé dans de nombreux travaux de recherche.

3.2 La maladie du cœur

Le cœur est une pompe musculaire puissante qui bat environ 100000 fois par jour et propulse 10000 litres de sang dans le corps. Pour que ce muscle puisse accomplir un tel travail, il faut qu'il soit continuellement approvisionné en oxygène, c'est-à-dire en sang frais. Cet approvisionnement en sang est assuré par les artères coronaires, les artères nourricières du cœur (figure 3.1).

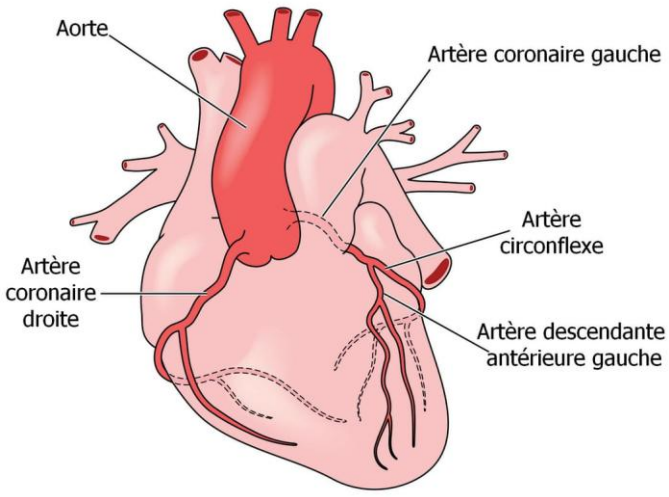

Figure 3.1 : Artères coronaires

Tout problème de santé qui affecte la structure ou le fonctionnement du cœur est une maladie du cœur. On croit parfois à tort qu'il n'en existe qu'un seul type, alors qu'en fait, les maladies du cœur sont nombreuses. Les causes de ces problèmes de santé sont d'ailleurs diverses.

On compte plusieurs types de maladies du cœur. Dans ce projet, nous nous intéressons à la maladie coronarienne.

3.3 La Maladie Coronarienne (CAD : Coronary Artery Disease)

Aussi appelée « athérosclérose », ou « durcissement des artères » c'est une maladie qui touche les artères ayant pour fonction d'alimenter le cœur en sang (artères coronaires). Elle est souvent causée par l'athérosclérose, une accumulation de plaques à l'intérieur de la paroi des artères. Cette accumulation rétrécit peu à peu l'intérieur des artères et ralentit le flot de sang.

Il existe plusieurs facteurs de risque de la maladie coronarienne; certains ne sont pas modifiables, mais d'autres le sont. La maladie coronarienne se développe de façon progressive au fil des ans et finit par se manifester par des symptômes tels que les douleurs thoraciques. Le diagnostic est établi au moyen de différents examens, comme l'électrocardiographie ou l'épreuve d'effort. Le traitement peut comprendre des changements au mode de vie, des médicaments et, à l'occasion, des interventions médicales ou chirurgicales [10].

3.4 Le dataset utilisé :

Attribut	Description	Type	Valeurs
Age	age en années	Entier	[29-77]
Sexe	Sexe	Discret	2= masculin 1= féminin
Douleur	Type de douleur thoracique	Discret	1= angine de poitrine typique 2= angine atypique 3= douleur non angorale 4= asymptomatique
Pression (Trestbps)	Pression artérielle au repos (en mm Hg à l'admission à l'hôpital)	Entier	[94-200]
Cholestérol	Cholestérol sérique en mg/dl	Entier	[126-564]
Glycémie (Fbs)	Glycémie à jeun (>120 mg/dl) (fastingbloodsugar)	Discret	2=vrai 1=faux
Electrocardio (Restecg)	Résultats de l'électrocardiographie au repos	Discret	1 = normal 2= présentant une anomalie de l'onde ST-T 3= affiche une hypertrophie ventriculaire gauche probable ou définitive selon les critères d'Estes.
Fréq_cardiaque (Thalach)	Fréquence cardiaque maximale atteinte	Entier	[71-202]
Angine (Exang)	Angine induite par l'exercice	Discret	2= oui 1= non
Depression (Oldpeak)	Dépression ST induite par l'exercice par rapport au repos	Réel	[0.00-62.00]
Pic (Slope)	Pente du segment ST d'exercice maximal	Discret	1= montée en flèche 2= plat 3= descente
Vaisseau (Ca)	Nombre des grands vaisseaux (0,3) colorés par la fluoroscopie	Discret	{1, 2, 3, 4}
Thal	Type de défaut	Discret	1 = normal 2 = défaut fixe 3 = défaut réversible
Cœur (Target)	Présence ou non de la maladie	Discret	2=oui 1=non

Tableau 3.1 : Description des attributs du dataset utilisé [13]

La propagation de la CAD et ses effets néfastes pour la santé a sonné l'alarme pour un diagnostic précoce qui permet de prédire la maladie dès son apparition, afin d'éviter une forme grave. Pour cela les recherches se sont orientées vers la conception de systèmes d'aide à la décision basés sur des données réelles. Ceci a donné lieu à la construction de plusieurs banques de données (ou dataset) par plusieurs équipes de recherches dans le domaine. La plupart de ces dataset sont accessibles et téléchargeables à travers le site « UCI – Machine Learning Repository » [11]. Parmi les datasets disponibles, nous avons téléchargé celui nommé « Statlog Dataset » [12]. Le dataset est décrit par 14 attributs et consiste en 270 patients dont 150 ne présentent pas la maladie CAD.

3.5 Le module Scikit Learn (Sklearn) de Python

Pour la mise en œuvre de notre système d'aide à la décision basé sur le bagging le module Scikit-Learn (ou Sklearn) de Python constitue un des meilleurs choix pour la plaforme logicielle. Sklearn met à notre disposition une multitude d'algorithmes d'apprentissage supervisé ou non-supervisé, dont :

- Linear Regression (régression linéaire).
- Logistic Regression (régression logistique).
- Decision Tree (arbre de décision).
- SVM (machines à vecteur de support).
- Naive Bayes (classification naïve bayésienne).
- KNN (Plus proches voisins).
- Dimensionality Reduction Algorithms.
- Gradient Boost & Adaboost.
- Réseaux de neuronnes.

Scikit-learn est un module Python intégrant les algorithmes classiques de l'apprentissage automatique. Il vise à fournir des solutions simples et efficaces pour les problèmes d'apprentissage, accessibles à tous et réutilisables dans divers contextes. Il existe quelques ensembles de données classiques intégrés avec Scikit-learn, par exemple "le dataset de fleurs des iris de Fisher" et "digits" qui est un dataset d'images de chiffres manuscrits [14].

3.6 La fonction « LogisticRegression »

Il est à rappeler que, dans le cadre de ce projet, nous envisageons d'expérimenter un bagging avec la « Régression logistique » comme classifieur de base. Nous commençons donc par détailler les paramètres du classifieur de base tel qu'il se présente dans le module Sklearn (Tableau 3.2).

3.6.1 Explication des paramètres de la régression logistique :

Ce sont les paramètres les plus couramment ajustés avec la régression logistique. Examinons plus en détail à quoi ils servent et comment changer leurs valeurs [15]:

penalty: (par défaut: « l2 ») Définit les normes de pénalisation. Certains objets du solveur ne prennent en charge que des paramètres de pénalisation spécifiques, ce qui doit être pris en compte.

- **l1** : pénalité supportée par les solveurs liblinear et saga
- **l2** : pénalité supportée par les solveurs newton-cg, sag, saga, lbfgs.
- **elasticnet:** pénalité uniquement supportée par le solveursaga.
- **none** : la régularisation des pénalités ne sera pas appliquée. Ne fonctionne pas avec le solveur liblinear.
-

solveur: (par défaut: « lbfgs ») Fournit des options pour choisir l'algorithme du solveur pour l'optimisation. Habituellement, le solveur par défaut fonctionne très bien dans la plupart des situations mais il est possible de spécifier un solveur particulier pour des occasions spécifiques telles que: des problèmes de classification avec des ensembles de données volumineux ou très volumineux.

Pour des cas particuliers, il est toujours judicieux de surveiller la façon dont le solveur travaille sur les données d'entraînement et de test en comparant différentes fonctions du solveur. Cela peut également aider à comprendre la finesse des différents solveurs sur un sujet très intéressant.

Paramètre	Valeurs possibles					Type	Valu Par
penalty	l1	l2	elasticnet	none			l2
dual	False	True				bool	False
tol						float	1,00E-04
fit_intercept	True	False				bool	True
random_state	RandomState instance					int	None
intercept_scaling						float	1
class_weight	dict	balanced					None
multi_class	auto	ovr	multinomial				auto
warm_start	False	True				bool	False
l1_ratio						float	None
max_iter						int	100
n_jobs						int	None
solvor	newton-cg	lbfgs	liblinear	sag	saga		lbfgs
verbose						int	0
C						float	1

Tableau 3.2 Paramètres du classifieur « LogisticRegression » sous « Sklearn »

- **lbfgs** : signifie BFGS à mémoire limitée (Limited-memory Broyden–Fletcher–Goldfarb–Shanno). Ce solveur ne calcule qu'une approximation de la Hessienne basée sur le gradient, ce qui le rend plus efficace sur le plan du calcul. D'un autre côté, son utilisation de la mémoire est limitée par rapport aux « bfgs » ordinaires, ce qui l'oblige à supprimer les gradients précédents et à n'accumuler que de nouveaux gradients comme le permet la restriction de mémoire.
- **liblinear** : Solveur plus efficace avec de petits ensembles de données. Seulement utile pour les problèmes ovr (one-versus-rest) et fonctionne pas avec les problèmes multi-classes contrairement aux autres solveurs . Ne fonctionne pas non plus avec les valeurs de pénalité« l2 » ou « None ».
- **newton-cg** : Solveur qui calcule explicitement la Hessienne ce qui peut être coûteux en calcul en grandes dimensions.
- **sag** : Signifie Stochastic Average Gradient Descent. Solveur plus efficace avec de grands ensembles de données.
- **saga** : Saga est une variante de Sag et il peut être utilisé avec la régularisation l1. C'est un solveur assez rapide et généralement le solveur de choix avec de très grands ensembles de données.

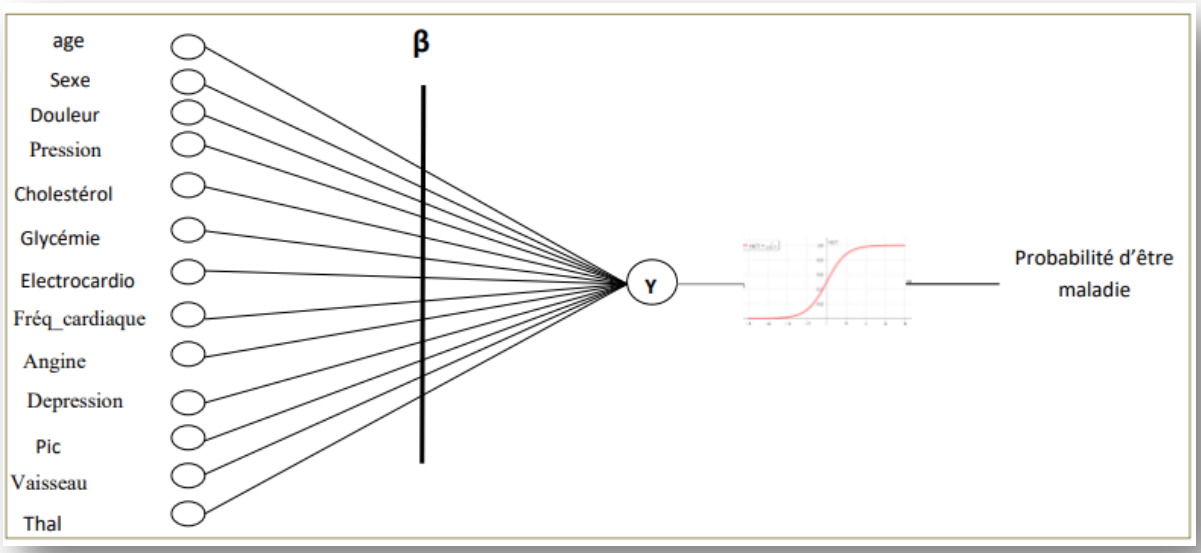

Figure3.2 :Récapitulation de la modélisation de la maladie

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_{13} X_{13}.$$

X_i : i^{eme} variable explicative : dans notre exemple, c'est une colonne parmi{Age ,sexe, douleur, pression, cholestérol, glycémie, electrocardio, fréquence cardiaque, angine, dépression, pic, vaisseau, Thal}.

β_i :les coefficients que le solveur doit trouver. Le solveur tente de trouver les valeurs des β_i qui permettent d'obtenir la meilleure sigmoïde pour les individus utilisés pour l'entrainement.

max_iter : (par défaut: 100) Ce paramètre définit le nombre maximum d'itérations autorisées pour la convergence du solveur.

dual: (par défaut: False) : Formulation double ou primale. La formulation double n'est implémentée que pour la pénalité l2 avec le solveur liblinear.

tol: (par défaut: 0,0001) Ce paramètre définit la tolérance ; en d'autres termes permet de définir le critère d'arrêt des itérations pour le solveur.

C: (par défaut: 1.0) Ce paramètre signifie la force de la régularisation et prend une valeur flottante positive (plus petit est le C et plus forte la régularisation sera).

fit_intercept: (par défaut: True) Concernant la fonction de décision, règle si une constante doit être ajoutée.

random_state : (par défaut: none) Ajuste la valeur de départ aléatoire.

- **none :**« seed » sera le module aléatoire de numpy : numpy.random.
- **int** : la valeur initiale sera générée en fonction de la valeur entière donné par le générateur de nombres aléatoires RandomState : random_state sera le générateur de nombres aléatoires (seed).

3.6.2 Plus de paramètres d'optimisation de la régression logistique pour un réglage fin :

De plus, ces paramètres peuvent être utilisés pour une optimisation supplémentaire, pour éviter le sur-ajustement (over-fitting) et effectuer des ajustements en fonction des impuretés:

warm_start : (par défaut: False) Ce paramètre n'est utile qu'avec des solveurs autres que liblinear. Il permet d'utiliser la solution de la formation précédente comme initialisation, d'où le terme démarrage à chaud.

- **False :** la solution précédente sera rejetée.
- **True :** la solution précédente sera réutilisée pour l'ajustement d'initialisation.

L1_ratio : (par défaut = None) Si le solveur est saga et que la pénalité est sélectionnée comme élastique, ce paramètre peut offrir une optimisation supplémentaire.

- **l1_ratio = 0 :** la pénalité sera égale à l2.
- **l1_ratio = 1** : la pénalité sera égale à l1.
- **0 <l1_ratio <1 ,**la pénalité sera une combinaison de l1 et l2, la fraction l1_ratio définira le poids de l1 dans le mix.

class_weight :

- dict
- équilibré
- (par défaut = none)

Poids associés aux classes sous la forme {class_label: weight}. Sinon, toutes les classes sont censées avoir un poids unitaire.

Le mode «équilibré» utilise les valeurs de y pour ajuster automatiquement les poids inversement proportionnels aux fréquences de classe dans les données d'entrée comme n_samples / (n_classes * np.bincount (y)).

multi_classe : (par défaut: 'auto')

- **auto :** Sélectionne automatiquement entre ovr (one-versus-rest) et multinomial. Si le solveur est liblinear, ovr sera sélectioné. De plus, si les données sont binaires, ovr sera sélectionné. Dans d'autres cas, le multinomial sera sélectionné automatiquement.
- **ovr :** fait que chaque problème binaire est adapté à chaque étiquette.
- **multinomial :** la distribution de probabilité sera ajustée avec la perte multinomiale. (ne fonctionne pas avec le solveur liblinear)

verbose : (par défaut: 0) signifie les informations imprimées pendant l'exécution de l'algorithme d'apprentissage automatique lorsqu'elles sont disponibles.

Dans les cas de régression logistique, disponible uniquement lorsque le solveur est soit liblinear, soit lbfgs .

- **0 :** pas de verbosité, les informations ne seront pas affichées.
- **1 :** Un peu de verbosité, certaines informations seront affichées.
- **2 :** Plus de verbosité, plus d'informations seront affichées.

n_jobs : (par défaut: none)ce paramètre signifie que les cœurs de processeur sont autorisés à fonctionner en parallèle.

Ne fonctionne que lorsque le solveur n'est pas liblinear et que multi_class est "ovr ".

- **none :** Un seul cœur de processeur fonctionnera
- **-1 :** tous les cœurs de processeur seront attribués lorsque cela est possible.
- **int :** les cœurs de processeur seront autorisés à travailler en parallèle en fonction de la valeur entière attribuée lors de la régression logistique.

3.6.3 Expérimentation de la régression logistique

Nous passons maintenant à l'expérimentation de la régression logistique sur notre dataset qui concerne la maladie coronarienne. Le but étant de construire un modèle de prédiction qui permet de diagnostiquer cette maladie avec la meilleure précision. Les expériences menées ont été axées principalement sur le choix du solveur, le nombre d'itérations qui permettent au solveur de converger et la pénalité utilisée. Notons que si le nombre d'itération n'est pas suffisant le solveur risque de ne pas converger, et donc les résultats donnés ne sont pas optimisé.

Il faut rappeler qu'avant l'expérimentation, le dataset a été répartis aléatoirement en ensemble de d'entrainement (training set) et ensemble de test avec une proportion de 20% pour le test. Ensuite les solveurs ont été passés en revue. Pour chaque expérience une cross-validation est appliquée et les mesures de performances sur l'ensemble de test sont calculées. Les résultats de ces expériences sont portés sur le tableau 3.3.

Les résultats du tableau 3.3 montrent clairement que les solveurs « newton-cg » avec la pénalité « l1 », « saga » et « sag » sont trop gourmands en temps d'exécution puisqu'il leur faut 10000 itérations pour assurer la convergence. Le solveur « lbfgs » (solveur par défaut) nécessite 1000 itérations. Nous pouvons remarquer que pour le dataset utilisé, les solveurs « liblinear » et « newton-cg » avec les pénalités « l2 » ou « none » sont plus intéressants puisqu'ils nécessitent moins d'itérations avec des performances intéressantes (la précision de la prédiction avoisine les 90%). Les courbes ROC qui correspondent aux meilleurs résultats sont présenteés dans la figure 3.3.

solveur	penalty	max_iter	Accuracy	AUC	cross_val
liblinear	l2	100	88,89	0,94	81,88
	l1	350	88,89	0,94	82,34
newton-cg	l2	100	88,89	0,93	82,36
	none	100	85,19	0,92	85,41
	l2	10000	87,04	0,95	82,84
saga	l1	10000	87,04	0,95	82,38
	elasticnet	10000	87,04	0,95	82,84
	none	10000	87,04	0,95	82,84
sag	l2	10000	87,04	0,95	82,36
	none	10000	87,04	0,95	82,36
lbfgs	l2	1000	88,89	0,93	82,36

Tableau 3.3 : Résultats de l'expérimentation avec le classifier « LogisticRegression »

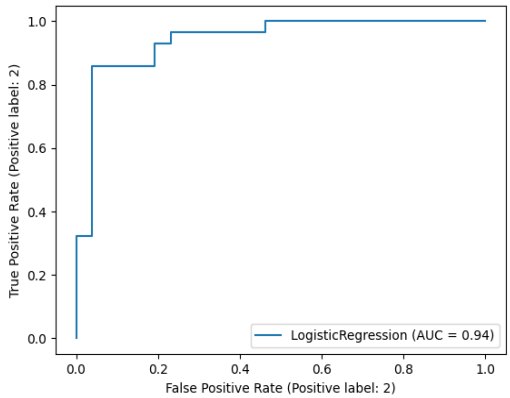

Figure 3.3 (a) : Courbe ROC pour le solveur « liblinear »

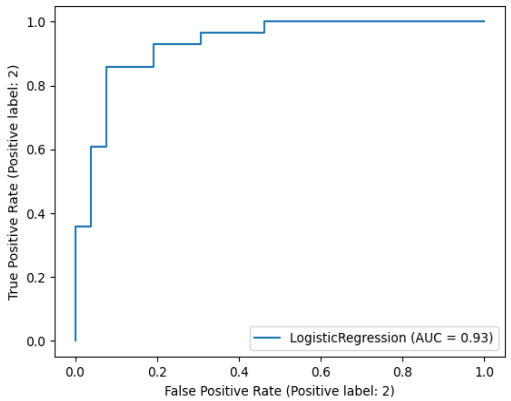

Figure 3.3 (b) : Courbe ROC pour le solveur « newton-cg », pénalité='l2'

3.7 La fonction « BaggingClassifier »

Un classificateur Bagging est un méta-estimateur d'ensemble qui ajuste les classificateurs de base chacun sur des sous-ensembles aléatoires de l'ensemble de données d'origine, puis agrège leurs prédictions individuelles (soit par vote, soit par moyenne) pour former une prédiction finale. Un tel méta-estimateur peut typiquement être utilisé comme un moyen de réduire la variance d'un estimateur boîte noire (par exemple, un arbre de décision ou une régression logistique), en introduisant la randomisation dans sa procédure de construction et en en faisant ensuite un ensemble.

La fonction « BaggingClassifier » du module « Sklearn » est paramétrable. Le tableau 3.4 présente les principaux paramètres qui permettent d'ajuster les résultats du classifieur d'ensemble « BaggingClassifier » [16].

Paramètres			Type	
base_estimator	none			*None*
n_estimators			int	10
max_samples			float/int	1.0
max_features			float/int	1.0
bootstrap	*False*	True	bool	True
bootstrap_features	*False*	True	bool	False
oob_score	*False*	*True*	bool	False
warm_start	*False*	*True*	bool	False
n_jobs			int	None
random_state	none	instance RandomState	int	None
Verbose			int	0

Tableau3.4 : Les paramètres de la fonction « BaggingClassifier » du module « Sklearn »

base_estimator : (par défaut = none). L'estimateur de base à ajuster sur des sous-ensembles aléatoires de l'ensemble de données. Dans le cadre de ce projet l'estimateur de base est la régression logistique .

n_estimators : (par défaut = 10). Le nombre d'estimateurs de base dans l'ensemble.

max_samples : (par défaut = 1.0). Le nombre d'échantillons à tirer pour former chaque estimateur de base (avec remplacement par défaut). Avec la valeur par défaut (1.0), l'estimateur est entrainé sur la totalité de l'ensemble d'entrainement.

max_features : (par défaut = 1.0). Le nombre de colonne à tirer pour entraîner chaque estimateur de base (sans remplacement par défaut).

bootstrap : (par défaut = True). Si les échantillons sont prélevés avec remplacement. Si faux, un échantillonnage sans remplacement est effectué.

bootstrap_features : (par défaut = False). Si les fonctions sont dessinées avec remplacement.

oob_score : (par défaut = False). Indique s'il faut utiliser des échantillons hors sac pour estimer l'erreur de généralisation (Uniquement disponible si bootstrap = True).

warm_start : (par défaut = False). Lorsqu'il est défini sur True, réutilisez la solution de l'appel précédent pour ajuster et ajouter plus d'estimateurs à l'ensemble, sinon, ajustez simplement un tout nouvel ensemble.

n_jobs : (par défaut = none). Le nombre de travaux à exécuter en parallèle. None signifie 1 sauf dans un joblib.parallel_backend contexte. -1 signifie utiliser tous les processeurs.

random_state : (par défaut = none). Contrôle le rééchantillonnage aléatoire du jeu de données d'origine (par échantillon et par fonctionnalité). Si l'estimateur de base accepte un random_state attribut, un germe différent est généré pour chaque instance de l'ensemble. Passez un int pour une sortie reproductible sur plusieurs appels de fonction.

Verbose : (par défaut = 0). Contrôle la verbosité lors de l'ajustement et de la prédiction.

3.7.1 Expérimentation du Bagging :

Nous passons maintenant à l'expérimentation du bagging avec la régression logistique comme estimateur de base. Nous avons expérimenté uniquement les solveurs qui ont donné de bons résultats dans l'expérimentation précédente. Les résultats de cette deuxième phase d'expérimentation sont récapitulés dans le tableau 3.5.

solver	penalty	max_iter	max_samples	n_estimators	max_features	Accuracy	AUC	cross_val
liblinear	l2	100	0,4	40	0,8	**90,74**	**0,95**	84,2
	l1	350	0,6	20	0,8	**90,74**	**0,93**	83,74
newton-cg	l2	100	0,7	40	0,8	88,89	0,94	83,74
	none	100	0,6	60	0,7	88,89	0,94	82,84
saga	l2	10000	0,6	50	0,8	87,04	0,95	82,84
sag	none	10000	0,6	50	0,8	**90,74**	**0,95**	82,38

Tableau 3.5 : Résultats de l'expérimentation du Bagging

L'examen des résultats du tableau 3.5 montre d'abord que l'intuition de départ était correcte : à savoir que le bagging permet d'améliorer les résultats obtenus en utilisant un seul estimateur. Ensuite, il est clair que les meilleurs résultats ont été obtenus en utilisant le solveur « liblinear », puis le solveur « sag » avec pénalité='none'. Puisque ce dernier nécessite un nombre très important d'itérations pour assurer la convergence, nous allons

retenir uniquement les résultats obtenus avec un bagging régression logistique et le solveur « liblinear ». Avec uniquement 40 estimateurs de base, chacun d'eux entrainé sur 40% de l'ensemble d'entrainement et en utilisant uniquement 80% des attributs, la précision de la prédiction a dépassé 90%. Les courbes ROC des deux meilleurs classifieurs sont représentés dans la figure 3.4.

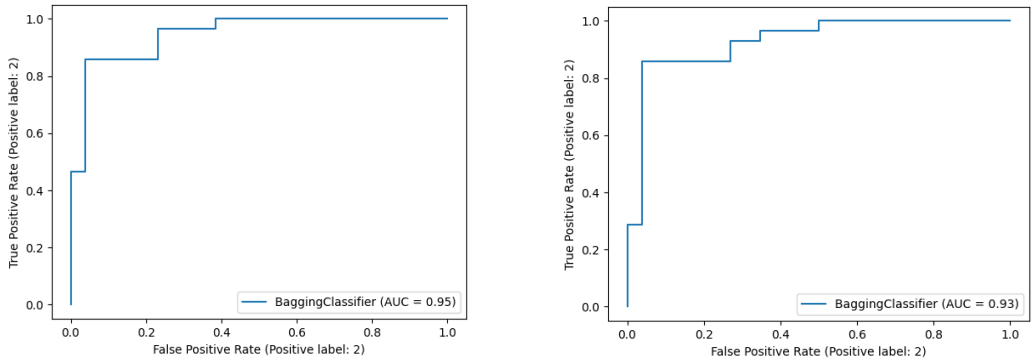

Figure 3.4: Courbes ROC des deux meilleurs classifieurs obtenus par bagging

3.8 Le module joblib :

3.8.1 Définition :

Joblib est un ensemble d'outils pour fournir un pipeline léger en Python. En particulier, joblib propose :

- Une mise en cache transparente sur disque des valeurs de sortie et réévaluation paresseuse (modèle mémorisé) ;
- Un calcul parallèle simple et facile
- Une journalisation et un suivi de l'exécution.

Joblib est optimisé pour être rapide et robuste en particulier sur les données volumineuses et dispose d'optimisations spécifiques pour les tableaux Numpy.

3.8.2 Enregistrement de modèle avec joblib :

Joblib fournit des utilitaires pour enregistrer et charger des objets Python qui utilisent efficacement les structures de données Numpy. Cela peut être utile pour certains algorithmes d'apprentissage automatique qui nécessitent beaucoup de paramètres ou stockent l'ensemble de données complet (comme K-Nearest Neighbors).

En répertorie certaines considérations importantes lors de la finalisation de modèles d'apprentissage automatique[17].

- **Version Python :**Il est souvent nécessaire de noter la version (majeure et mineure) de Python lors de l'utilisation de joblib. Un modèle sérialisé doit être désérialisé en utilisant la même version.

- **Versions de la bibliothèque** : On doit également utiliser la même version de bibliothèque, en particulier celle de Sklearn, sinon le modèle déserialisé risque de ne pas fournir les mêmes résultats.

L'utilisation de joblib nous a permis de sauvegarder sur disque un modèle obtenu avec Sklearn. Ceci est particulièrement important lorsqu'il s'agit d'algorithmes d'apprentissage qui utilisent le bagging :

- D'abord, puisque le bagging est basé sur un tirage aléatoire, si nous générons plusieurs fois un modèle avec exactement les mêmes paramètres, nous obtenons des modèles différents (en termes de performances).
- Ensuite, la génération d'un modèle basé sur le bagging nécessite un temps de calcul important. Si un mdèle est sauvegardé (on dit aussi sérialisé) avec joblib, il n'est plus nécessaire de le regénérer, il suffit de le lire et l'appliquer sur de nouvelles données.

3.9 Conclusion :

Ce chapitre nous a permis d'expérimenter un bagging avec comme estimateur de base la régression logistique. L'expérimentation a permis de construire un modèle de diagnostic de la maladie coronarienne. Les résultats obtenus, en termes de performances, sont encourageants. De plus, ça confirme l'idée selon laquelle un bagging qui utilise plusieurs classifieurs faibles donne de meilleurs résultats qu'un classifieur fort.

Conclusion Générale

Ce projet s'inscrit dans le cadre de l'intelligence artificielle, et plus précisément, dans la catégorie des méthodes d'apprentissage automatique. Ces méthodes visent à extraire de la connaissance à partir de masses de données importantes. L'objectif principal de ce projet était d'expérimenter un algorithme de bagging basé sur une régression logique afin de produire un modèle efficace d'aide à la décision pour le diagnostic de la maladie coronarienne.

La recherche documentaire nous a permis de découvrir et comprendre les méthodes ensemblistes (Bagging, Boosting et Stacking), avant de se concentrer sur les fondements et les principes du bagging qui consiste à agréger les prédictions d'un nombre importants de modèles. Des recherches récentes ont montré que les modèles qui suivent un principe de bagging (tels que les forêts aléatoires) sont très performants.

Ce projet a été une occasion immense d'apprentissage tant sur le plan théorique que pratique. En effet, l'étude des méthodes ensemblistes nous a ouvert un grand horizon vers la conception de modèles de prédiction qui combinent plusieurs estimateurs faibles. Plusieurs travaux de recherche confirment l'idée selon laquelle la combinaison de plusieurs estimateurs faibles permet d'obtenir un estimateur efficace. Sur le plan pratique, nous avons pu découvrir le module « Scikit-Learn » et utilisé quelques unes des méthodes qu'il contient. C'est l'un des modules les plus utilisés actuellement dans le domaine d'apprentissage automatique. Ceci a renforcé notre formation technique.

Au bout de ce projet, nous avons pu construire et paramétrer un modèle de bagging basé sur la régression logistique et l'appliquer à la prédiction de la maladie coronarienne. Il faut noter que la paramétrisation d'un modèle de bagging est un processus très long, vu le nombre de paramètres, et aussi le caractère aléatoire dans le choix de l'ensemble de données à partir duquel un estimateur est construit. Nous avons pu obtenir plusieurs modèles avec des performances très intéressantes (précision supérieure à 90%). De plus, les modèles obtenus ont été sauvegardés en utilisant le module « joblib » ce qui permet de les réutiliser sans avoir à les régénérer.

Une suite logique de ce travail est de le renforcer par une interface simple et facile d'utilisation par un médecin. En effet, l'exploitation des modèles obtenus pour l'aide au diagnostic de la maladie coronarienne, passe par la construction d'une interface (desktop et/ou mobile) qui permet au médecin de suivre l'évolution d'un malade et de déclencher une alerte lorsque cela est nécessaire.

Une autre direction à explorer serait d'expérimenter d'autres estimateurs de base (SVM, Réseaux Bayésiens, …) en utilisant du bagging, et pourquoi pas penser à combiner plusieurs estimateurs de base dans le même algorithme de bagging, ce qui n'est pas permis actuellement dans « Sklearn ».

Bibliographie

[1] Bastien L, Data Mining. [en ligne]. 31 janvier 2018,Disponible sur <<https://www.lebigdata.fr/data-mining-definition-exemples>> consulté en Janvier 2021.

[2] Data Mining [en ligne] .Disponible sur << http://silico.biotoul.fr/site/images/archive/e/e6/20141016073026!Data_Mining_-_Intro.pdf>>, consulté en Janvier 2021

[3] Bootstrap Aggregation,disponible sur : https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789136609/5/ch05lvl1sec29/bootstrap-aggregation, consulté en Février 2021.

[4] Data Science Team , Gradient Boosting, disponible sur: <<https://datascience.eu/fr/apprentissage-automatique/gradient-boosting-ce-que-vous-devez-savoir/>>, consulté en Février 2021.

[5] AISHWARYA SINGH, A Comprehensive Guide to Ensemble Learning (with Python codes), JUNE 18, 2018

[6] Haddad, B. M. BagStack Classification for Data Imbalance Problems with Application to Defect and Labeling in Semiconductor Units. Thèse de PhD, Arizona State University, US, 2019.

[7] Adrien R , La régression logistique, Qu'est-ce que c'est ? disponible sur : <<https://datascientest.com/regression-logistique-quest-ce-que-cest#:~:text=La%20r%C3%A9gression%20logistique%20est%20un,logistique%20comme%20fonction%20de%20lien.>>, consulté en Février 2021.

[8] Navlani, A. Understanding Logistic Regression and Building Model. Disponible sur: <<https://python.plainenglish.io/understanding-logistic-regression-and-building-model-in-python-1752a7e562a8>>, consulté en Février 2021.

[9] Mariaccia,H. La régression logistique Expliquée à ma Grand-Mère. Disponible sur : <<https://medium.com/tell-ia/la-r%C3%A9gression-logistique-expliqu%C3%A9e-%C3%A0-ma-grand-m%C3%A8re-52a2ab30788>>; consulté en Février 2021.

[10] Maladie coronarienne (Athérosclérose). Disponible sur <<https://www.ottawaheart.ca/fr/maladie-du-c%C5%93ur/maladie-coronarienne-ath%C3%A9roscl%C3%A9rose>> consulté en avril 2021.

[11] Dheru Due, UC Irvine Machine Learning Repository .Disponible sur

<<https://archive.ics.uci.edu/ml/index.php>> consulté en avril 2021

[12] Sarmento, R. Statlog (Heart) Data Set, Disponible sur <<https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29>>, consulté en Avr, 2021

[13] F. Atmani. Intégration du Datamining et du RàPC dans les Systèmes d'Aide à la Décision. Mémoire de Master ISI, Université de Mostaganem, 2020.

[14] Introduction à l'apprentissage automatique avec scikit-learn. 23/01/2018.Disponible sur

<<https://www.vneuron.com/2018/01/23/introduction-a-lapprentissage-automatique-avec-scikit-learn/>>.consulté en avril 2021.

[15] sklearn.linear_model.LogisticRegression . Disponible sur <<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>> consulté en avril 2021.

[16] sklearn.ensemble.BaggingClassifier . Disponible sur <<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>> consulté en mai 2021.

[17] Jason Brownlee, Save and Load Machine Learning Models in Python with scikit-learn, June 8, 2016. Disponible sur <<https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/>> consulté en juin 2021.