MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DELARECHERCHESCIENTIFIQUE

UNIVERSITE ABDELHAMID IBN BADIS -

MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique

Filière : Informatique

RAPPORT DE PROJET DE FIN D'ETUDE

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

THEME :

Etude de Quelques Algorithmes de Boosting dans l'Apprentissage Automatique

Etudiantes : Boudia Souad
Achacha Yasmine

Devant la commission de jury :

Mme Kaid Slimane Bouchra	FSEI, Université de Mostaganem, Présidente
Mr Habib Zahmani Mohammed	FSEI, Université de Mostaganem, Examinateur
Mr Henni Fouad	FSEI, Université de Mostaganem, Encadreur
Mr Korichi Mokhtar El-Amine	FSEI, Université de Mostaganem, Co-Encadreur

Année Universitaire 2021-2022

Dédicace

On a le grand plaisir de dédier ce modeste travail à :
Qui nous sont très chers au monde nos parents, chère marie et enfants pour leur soutiennent durant toutes notre années d'étude: nos amies et proches: A tous ceux qui par leur sourire, leur gentillesse et espoir nous ont encouragés à poursuivre nos études.

Boudia Souad et Achacha Asma

Remercîment

Premièrement, nous remercions notre Dieu le tout puissant qui nous a dotées de la merveilleuse faculté de raisonnement, qui nous a donné la fois, la santé, le courage et la volonté pour terminer ce travail.

Nos intentions vont à M^{r} Henni Fouad et M^{r} Korichi Mokhtar El Amine, notre encadreur et Co-encadreur pour leur aide précieuse, leur patience et l'intérêt qu'ils portaient à notre travail.

Nous remercions également les membres du jury, Mme Kaid Slimane et Mr Habib Zahmani d’avoir accepté de lire et d'évaluer notre travail.

Résumé

En apprentissage automatique (Machine Learning, ML), une méthode ensembliste consiste à combiner plusieurs algorithmes d'apprentissage automatique en réduisant la variance et le biais des modèles de base pour obtenir des prédictions de meilleure qualité.

Dans ce contexte, Il existe deux grandes classes de méthodes ensemblistes : les méthodes séquentielles telles que le Boosting et les méthodes parallèles telles que le Bagging. D'un autre côté, le « Stacking » peut être considéré comme étant une généralisation du Bagging.

L'objectif de ce projet est d'étudier le principe du Boosting et présenter quelques algorithmes de « Boosting ».

Mots-clés:

Apprentissage Automatique, Méthodes ensemblistes, Bagging, Boosting, Stacking, AdaBoost , Gradient Boosting , LigitBoost

Abstract

In machine learning (ML), ensemble methods consist of combining multiple machine learning algorithms, reducing the variance and bias of the basic models to obtain better quality predictions.

In this context, there are two main ensemble methods, the sequential and the parallel. With the sequential set method (such as "Boosting"), the models are trained in sequence, and the parallel set method (such as "Boosting"), the models are trained simultaneously (at the same time). On the other hand, "Stacking", can be viewed as a generalization of the Bagging principle.

The objective of this project is to study the foundations of “Boosting” algorithms and present some algorithms that use the Boosting principle.

Keywords:

Machine learning Ensemble methods, Bagging, Boosting, Stacking, AdaBoost, GradientBoosting, LigitBoost.

Liste des figures

Liste des tableaux

Liste des abréviations

Abréviation	Expression Complète
AUC	Area Under the roc Curve
EVC	Epreuve de Vérification des Connaissances
IA	Intelligence Artificielle
ROC	Receiver Operating features Curve
ML	Machine Learning
MART	Multiple Additive Regression Trees
ACC	Accuracy
CHD	Coronary Heart Disease
GaussianNB	Gaussian Naive Bayes

Table des matières

Introduction Générale

L’apprentissage automatique (Machine Learning, ou ML) n’est pas une technologie nouvelle mais elle prend une tout autre ampleur avec l’émergence de l’intelligence artificielle. Apprentissage automatique, ou «machine learning» est une démarche d’analyse des données qui automatise la création de modèles analytiques et favorise les données prédictives.

Le Machine Learning, grâce à son caractère autonome, présente de nombreux avantages. Ses atouts peuvent s’appliquer à une multitude de secteurs d’activité. Le premier avantage est qu’il permet de traiter une grande quantité de données très rapidement. Grâce à cela, il pourra formuler des prédictions précises. Ces prédictions peuvent être utilisées dans les domaines de la santé, des finances, de la « supply-chain » ou encore de la maintenance industrielle.

Les modèles de Machine Learning sont classés en deux grandes catégories : apprentissage supervisé qui consiste à apprendre à une fonction à faire correspondre une entrée à une sortie continue (régression) ou discrète (classification) en se basant sur des exemples connus, ou apprentissage non supervisé qui est utilisé pour tirer des conclusions et trouver des tendances à partir de données d’entrée sans étiquettes. L’Overfitting (sur-ajustement) et l’Underfitting (sous-ajustement) sont les causes principales des mauvaises performances des modèles prédictifs générés par les algorithmes de Machine Learning. Lorsqu'on entraîne un modèle sur des données étiquetées, on émet l'hypothèse qu'il doit également fonctionner sur de nouvelles données.

La classification, comme étant une tâche qui répond au besoin d’un large type d’applications, nécessite d’être au cœur du datamining. Dans notre projet, un point particulier à été mis sur les approches des méthodes ensemblistes (Bagging, Boosting et Stacking). Ces méthodes combinent plusieurs modèles d'apprentissage automatique, pour obtenir des prédictions de meilleure qualité.

Plusieurs techniques sont appliquées sur l'ensemble d'apprentissage pour construire des classificateurs de base, la technique de Boosting est parmi les méthodes d'ensembles séquentielles les plus abouties. Le succès du Boosting a ouvert la voie à la proposition de plusieurs algorithmes qui s'appuient sur le même principe mais diffèrent au niveau des caractéristiques. On peut citer : AdaBoost, GradientBoosting et LogitBoost.

L'objectif de ce projet est d'abord de comprendre le principe des méthodes ensemblistes avant de mettre l'accent sur le Boosting. Ensuite de choisir quelques algorithmes de Boosting et les expérimenter sur des Datasets existants. Puisqu'il y a un nombre important d'algorithmes qui appliquent le principe du Boosting, nous avons fait le choix d'étudier trois algorithmes : Adaboost, GrandienBoosting et LogitBoost. L'étude expérimentale comporte l'utilisation de différents algorithmes comme estimateur de base et la paramétrisation des algorithmes traités à travers les hyperparamètres.

Ce rapport est organisé comme suit : dans le chapitre I, les principes de base du contexte de ce travail sont définis : le machine learning et l'introduction du principe des méthodes ensemblistes. Le chapitre II est dédié aux différents algorithmes de Boosting. Le chapitre III aborde avec plus de détails les trois algorithmes considérés dans ce projet. Le chapitre IV, quant à lui, présente quelques expérimentations faites en utilisant les trois algorithmes de Boosting sur deux datasets expérimentaux.

Chapitre 1 Les Méthodes Ensemblistes

1.1 Introduction

L'apprentissage automatique est un sous-ensemble de l'intelligence artificielle « IA », il est axé sur la création des systèmes qui apprennent et améliorent les performances, en se basant sur des données qu'ils traitent. Les algorithmes d'apprentissage automatique entrent en jeu pour optimiser, fluidifier, et sécuriser cette dernière [5]. L'objectif de ce chapitre est de présenter succinctement les objectifs des algorithmes du machine learning avant de mettre l'accent sur l'émergence et l'utilité des méthodes ensemblistes qui combinent plusieurs modèles afin d'améliorer la prédiction.

1.2 Machine Learning

1.2.1 Définition

La définition de l'apprentissage automatique a connu une progression durant plusieurs années, cela est dû au développement continu dans ce domaine :

- Le terme « Apprentissage automatique » a été inventé par l'informaticien Américain « Arthur Samuel », dans le début de l'année 1959. Il a créé le premier programme qui permet aux ordinateurs de jouer et d'apprendre le jeu de dames sans être explicitement programmé [8].

- En 1997, l'informaticien américain « Tom Michael Mitchell » introduit une nouvelle définition de l'apprentissage automatique. Il a considéré qu'un programme apprend d'une expérience E, par rapport à une classe de tâches T, et avec une mesure de performance P [8].

- Avec le temps, la définition de l'apprentissage automatique a commencé à prendre une dimension mathématique et statistique. Selon les auteurs dans [9], l'apprentissage automatique est essentiellement une forme de statistiques

appliquées, mettant davantage l'accent sur l'utilisation de l'ordinateur pour estimer statistiquement les fonctions compliquées et un accent moindre sur la démonstration des intervalles de confiance autour de ces fonctions.

Ces définitions peuvent varier en fonction de l'angle étudié, mais elles sont toutes orientées vers une seule direction, qui est définie comme suit : l'apprentissage automatique est la science ou l'art de la programmation des ordinateurs afin qu'ils puissent apprendre des données [8].

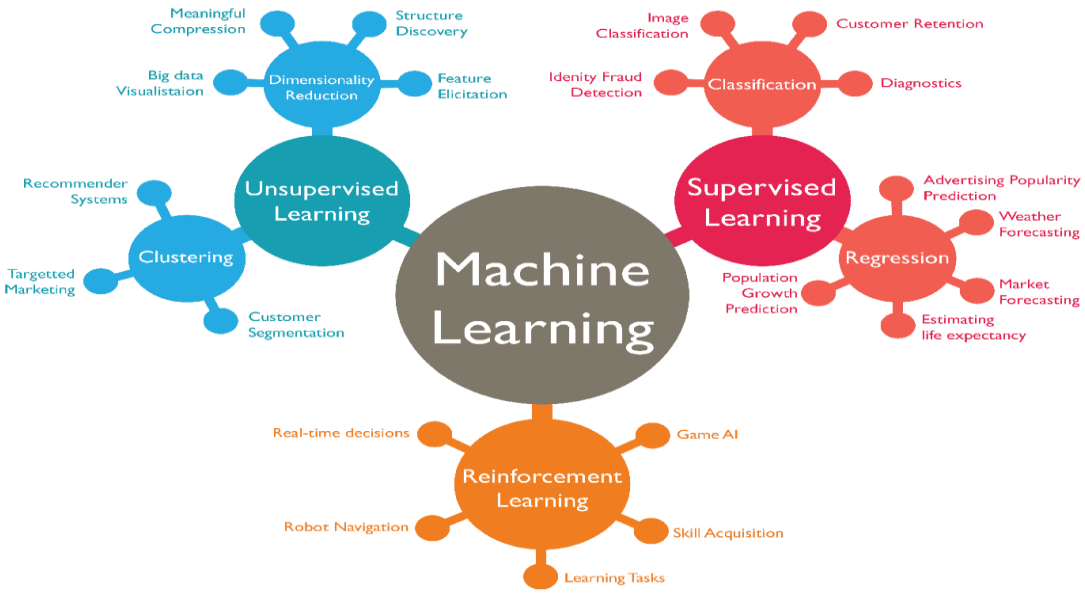


Figure1.1- Illustration Du Machine Learning

1.2.2 Objectif

L'objectif du machine Learning est de reconnaître, parmi des données, des structures souvent trop difficiles à détecter ou à mesurer manuellement. À partir de ces structures, on peut chercher à classifier des individus, des objets, à prédire la valeur d'une variable à un certain horizon, à expliquer l'apparition ou non d'une caractéristique.

Le machine Learning est par exemple utilisé pour :

- Prédire la quantité de ventes d'un produit sur les trois prochains mois, en fonction des ventes observées sur le même produit dans le passé.

●Créer une classification de la clientèle en fonction de caractéristiques sociodémographiques et des achats antérieurs.

●Attribuer un score à une personne indiquant si celle-ci est sur le point de se désabonner d'un produit ou non.

●Reconnaître automatiquement un chiffre manuscrit ou un visage.

Au-delà d'une ou deux variables, il devient difficile de voir à l'œil nu des structures dans les données, c'est à ce niveau que l'apprentissage statistique trouve tout son intérêt. Il en est de même lorsque le nombre d'individus (d'observations) augmente.

1.2.3 Démarche

Les algorithmes utilisés permettent - dans une certaine mesure - à un système piloté par ordinateurs ou assisté par ordinateur d'adapter ses analyses et comportements en réponse en se fondant sur l'analyse de données empiriques provenant d'une base de données ou de capteurs.

La difficulté réside dans le fait que l'ensemble de tous les comportements possibles compte tenu de toutes les entrées possibles devient rapidement trop complexe à décrire dans les langages de programmation disponibles, de sorte qu'on confie en quelque sorte à des programmes le soin d'apprendre de manière à auto-améliorer le système d'analyse ou de réponse (commande adaptative), ce qui est une des formes que peut prendre l'intelligence artificielle

Ces programmes, selon leur degré de perfectionnement, intègrent des capacités en probabilités et statistiques, traitement de données et éventuellement d'analyse de données issues de capteurs, de reconnaissance (reconnaissance vocale, reconnaissance de forme, d'écriture, etc.), de datamining et d'informatique théorique.

1.3 Limites des modèles uniques

Considérons que nous concevons un modèle d'apprentissage automatique. Un modèle est dit être un bon modèle d'apprentissage automatique s'il généralise correctement toutes les nouvelles données d'entrée du domaine du problème. Cela nous aide à faire des prédictions dans les données futures que le modèle de données n'a jamais vues. Maintenant, supposons que nous voulions vérifier dans quelle mesure notre modèle d'apprentissage automatique apprend et se généralise aux nouvelles données. Pour cela, nous avons le sur-apprentissage (overfitting) et le sous-apprentissage (underfitting), qui sont principalement responsables des mauvaises performances des algorithmes d'apprentissage automatique.

Avant de prolonger plus avant, comprenons deux termes importants (Figure 1.2) :

- **Biais :** sont les hypothèses simplificatrices faites par un modèle pour rendre la fonction cible plus facile à apprendre.

  - **Biais faible** : suggère moins d'hypothèses sur la forme de la fonction (ou du modèle) cible.
  - **Biais élevé** : suggère plus d'hypothèses sur la forme de la fonction cible.

- **Variance :** La variance est la quantité que l'estimation de la fonction cible changera si des données d'entraînement différentes étaient utilisées.

  - **Faible variance :** suggère de petites modifications de l'estimation de la fonction cible avec des modifications de l'ensemble de données d'apprentissage.

  - **Variance élevée :** suggère des modifications importantes de l'estimation de la fonction cible avec des modifications de l'ensemble de données d'apprentissage.

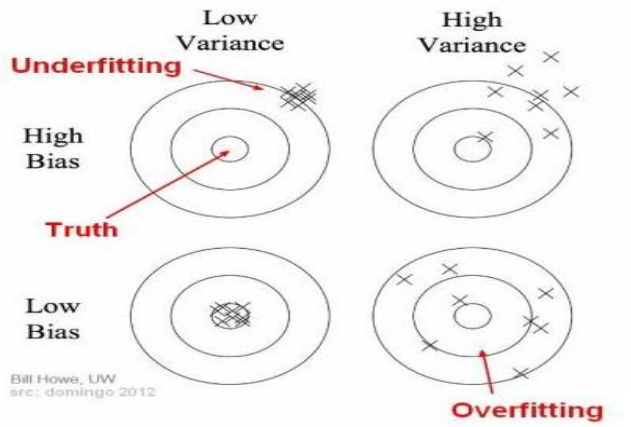


Figure 1.2- Biais et Variance à l'aide du diagramme de Bulls Eye [30]

1.3.1 Underfitting (Sous-Ajustement)

Un modèle statistique ou un algorithme d'apprentissage automatique est considéré comme présentant un sous-ajustement lorsqu'il ne peut pas capturer la tendance sous-jacente des données. Le Sous-Ajustement détruit la précision de notre modèle d'apprentissage automatique. Son occurrence signifie simplement que notre modèle ou l'algorithme ne correspond pas assez bien aux données utilisées pour l'entraînement. Cela se produit généralement lorsque nous avons moins de données pour construire un modèle précis et également lorsque, par exemple, nous essayons de construire un modèle linéaire avec des données non linéaires. Dans de tels cas, les règles du modèle d'apprentissage automatique sont trop simples et flexibles pour être appliquées à des données aussi minimales et, par conséquent, le modèle fera probablement beaucoup de prédictions erronées. Le sous-apprentissage peut être évité en utilisant plus de données et en réduisant également les caractéristiques par sélection de caractéristiques (Figure 1.3).

- En résumé : **Underfittingα Biais élevé et faible variance**

1.3.2 Overfitting (Sur-Ajustement)

Un modèle statistique est dit en Sur-Ajustement lorsque nous l'entraînons avec beaucoup de données. Lorsqu'un modèle est entraîné avec autant de données, il commence à apprendre du bruit et des entrées de données inexactes dans notre ensemble de données. Ensuite, le modèle ne catégorise pas correctement les données, en raison de trop de détails et de bruit. Les causes du Sur-Ajustement sont les méthodes non paramétriques et non linéaires, car ces types d'algorithmes d'apprentissage automatique ont plus de liberté pour créer le modèle basé sur l'ensemble de données et peuvent donc vraiment créer des modèles irréalistes. Une solution pour éviter le sur apprentissage consiste à utiliser un algorithme linéaire si nous avons des données linéaires ou à utiliser des paramètres tels que la profondeur maximale si nous utilisons des arbres de décision (Figure 1.3).

- En résumé : **Overfitting α Variance élevée et biais faible**

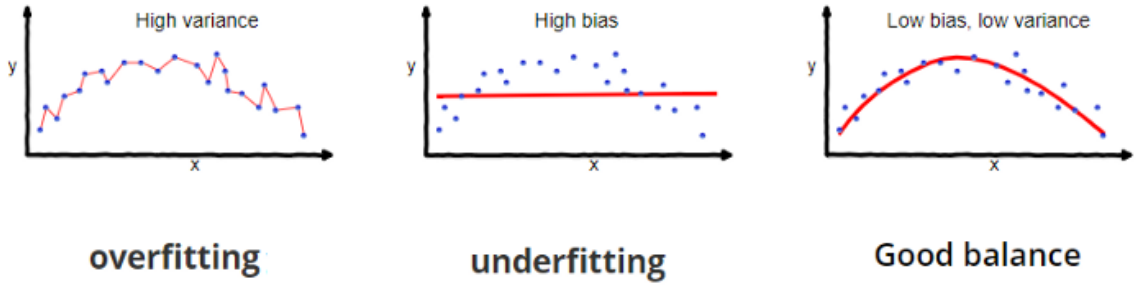


Figure 1.3- Courbe Sur-Ajustement, Sous-Ajustement Et Bon Ajustement [28]

1.4 Méthodes Ensemblistes

1.4.1 Principe général

Une Méthode Ensembliste selon la définition de Dietrich[6], est un ensemble de classificateurs individuels qui sont divers, mais précis, et dont les décisions sont combinées par moyenne ou par vote (la décision la plus populaire) pour donner une décision beaucoup plus précise. Selon Dietrich[6], un classificateur est précis si son taux

d’erreur est plus faible qu’une classification aléatoire, et deux classificateurs sont divers s’ils font, généralement, des erreurs différentes sur de nouvelles données.

Notons que les méthodes ensemblistes peuvent être appliquées à tous les algorithmes instables, ceux pour lesquels un petit changement dans les données d'entraînement induit un grand changement dans le classificateur final.

1.4.2 Différentes méthodes ensemblistes

1.4.2.1 Le Bagging

Le Bagging, abréviation de Bootstrap Aggregating a été proposé par Breiman [2] comme étant une méthode permettant de générer plusieurs versions d'un prédicteur et de les combiner pour obtenir un prédicteur agrégé. L'agrégation fait une moyenne sur les versions lors de la prédiction d'un résultat numérique et fait un vote de pluralité lors de la prédiction d'une classe. Les multiples versions sont formées en faisant des répliques (Bootstrap, ou tirage aléatoire avec remise) de l'ensemble d'apprentissage et en les utilisant comme nouveaux ensembles d'apprentissage (Figure 1.4). Des tests sur des ensembles de données réelles et simulées utilisant la classification et les arbres de régression et la sélection de sous-ensembles dans la régression linéaire montrent que le Bagging peut donner des gains substantiels en précision.

L'élément vital est l'instabilité de la méthode de prédiction. Si la perturbation de l'ensemble d'apprentissage peut entraîner des changements dans le prédicteur construit, alors le Bagging peut améliorer la précision.

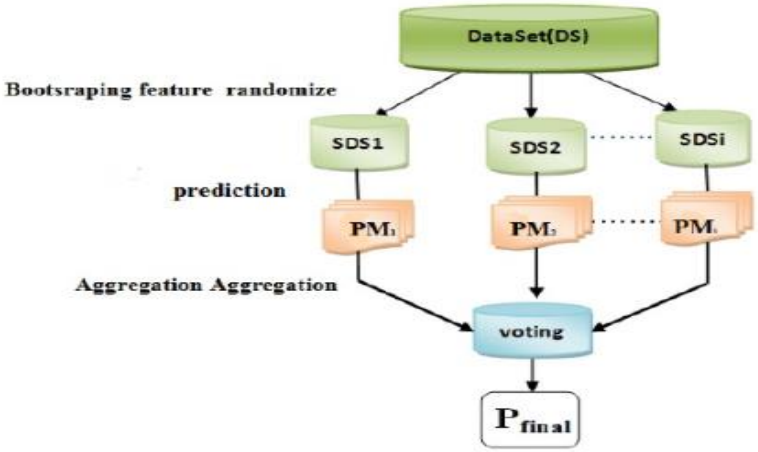


Figure 1.4- Principe Du Bagging [14]

1.4.2.2 Le Boosting

L'idée de base est de construire un nouveau classificateur en tenant compte de la performance d'une série de classificateurs précédents dans un processus séquentiel. L'ensemble d'apprentissage d'origine est renforcé par des poids qui seront ajustés à chaque étape dans l'objectif d'amplifier (Boost) les exemples mal classés (Figure 1.5). Les poids des exemples bien classés -par le dernier modèle construit-seront alors décrémentés, et les poids des exemples mal classés seront incrémentés, permettant ainsi au système de prêter plus d'attention aux exemples mal classés. Les modèles sont combinés par vote à majorité pondéré où la pondération est déterminée par la précision de prédiction de chaque classificateur [29].

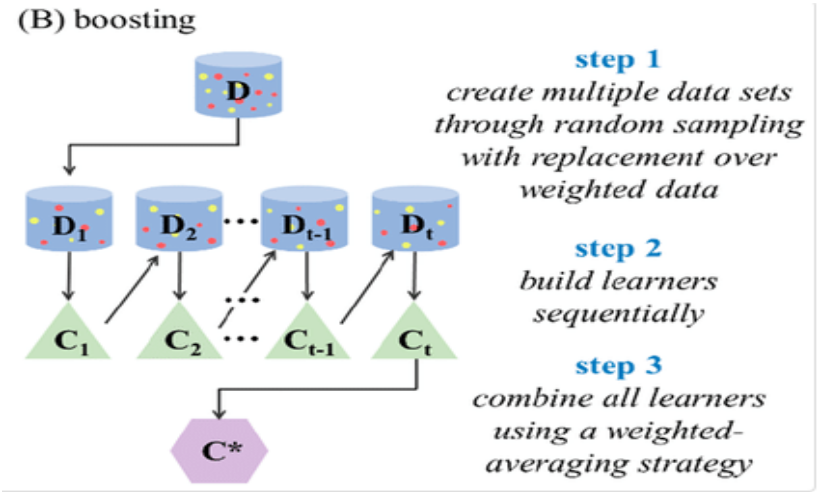


Figure 1.5- Principe Du Boosting [13]

1.4.2.3 Le Stacking

Inventé par David Wolpert [37], c'est une méthode qui permet de combiner plusieurs classificateurs de base. La première phase consiste à induire N classificateurs Cl_i, à partir de N ensembles de données $\{S_1, S_2, ..., S_N\}$. Le test est ensuite fait sur un ensemble d'évaluation $T = \{t_1, t_2, ..., t_N\}$ indépendant des ensembles d'apprentissage S_i. Dans la deuxième phase, un nouvel ensemble de données M est formé par les valeurs calculées $Cl_i(t_j)$ (ou bien les prédictions de chaque modèle sur l'ensemble d'évaluation) et la vraie classe de l'instance t_j, classe(t_j). Chaque instance de M sera de la forme $<Cl_1(tj), Cl_2(tj), \ldots, Cl_N(t_j)$, classe($t_j$)>. Dans la dernière étape, un classificateur global (Generalizer) est construit à partir de M. Les classificateurs de base peuvent être construits avec des algorithmes différents (arbres de décision, réseaux de neurones, ….) selon les contraintes du problème (Figure 1.6).

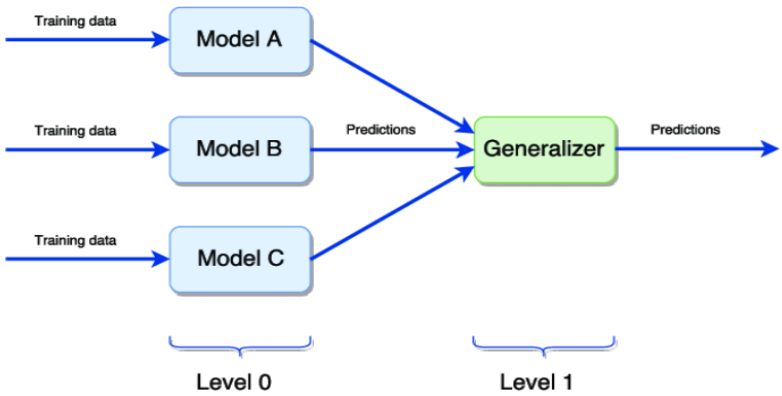


Figure 1.6- Principe Du Stacking

1.5 Evaluation d'un modèle

Un modèle est entraîné sur des données historiques disponibles afin de permettre de faire des prédictions sur de nouvelles données. Avant de passer à la prédiction, il faut d'abord s'assurer des performances des modèles générés. Pour cela, plusieurs mesures d'évaluation d'un modèle sont proposées. Les paragraphes qui suivent résument les principales évaluations de modèles.

1.5.1 Validation Croisée (Cross-Validation)

On parle en général de validation croisée à K blocs (ou K- Fold Cross Validation) pour désigner une technique d'évaluation d'un algorithme de Machine Learning. Cela consiste à découper le Dataset d'entraînement (Training Dataset) en K sous-ensembles (ou K Folds) puis prendre un des K sous-ensembles comme Dataset de validation (Validation Set) et les K-1 restants comme Dataset d'entraînement (Training Set). On répète l'opération sur toutes les combinaisons possibles (Figure 1.7). On obtient K mesures de performance dont la moyenne représente la performance de l'algorithme.

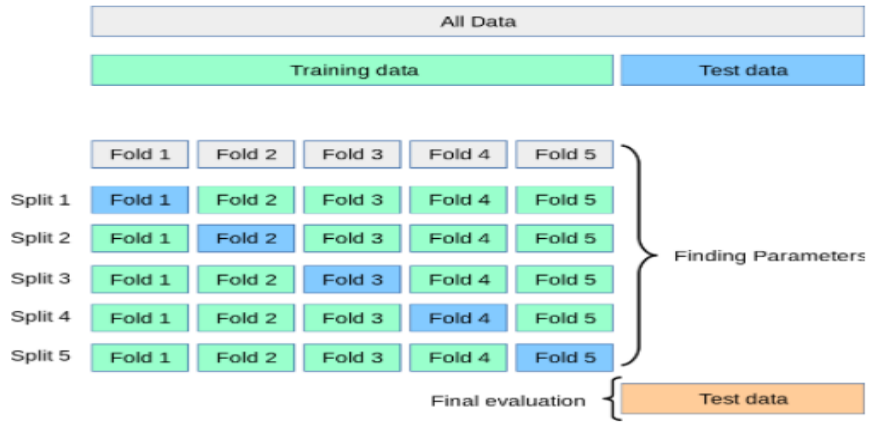


Figure 1.7- Exemple de 5-fold cross validation

1.5.2 Mesures de performance

Matrice de Confusion est un tableau croisé entre les valeurs réelles et les prédictions. Cette matrice permet d'identifier quartes catégories de résultats (Tableau 1.1) :

- **Vrais positifs (TP :** True Positive) : le nombre d'observations positives que le modèle a correctement prédites comme positives.
- **Faux positif (FP :** False Positive) : le nombre d'observations négatives que le modèle a incorrectement prédites comme positives.
- **Vrai négatif (TN :** True Négative) : le nombre d'observations négatives que le modèle a correctement prédites comme négatives.
- **Faux négatif (FN :** False Négative) : le nombre d'observations positives que le modèle a incorrectement prédites comme négatives.

Tableau 1.1- Matrice de Confusion

		Classe réelle	
Classe Prédite			
	TP	*FP*	
	FN	*TN*	

L'indicateur le plus simple est **l'Accuracy** : il indique le pourcentage de bonnes prédictions. C'est un très bon indicateur parce qu'il est très simple à comprendre.

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

Pour compléter l'Accuracy, on calcule également le **Recall** (en Français, le rappel) :

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

Enfin, un 3ème indicateur vient compléter l'accuracy et le recall, c'est la **Precision** :

$$Precision = \frac{TP}{TP + FP}$$

La F-mesure est la mesure de la moyenne harmonique de la précision et du rappel. Elle permet de mesurer la capacité du modèle à donner toutes les solutions pertinentes et à refuser les autres.

$$F_1 = \frac{2 * Precision * Recall}{Précision + Recall}$$

La Spécificité évalue la capacité du modèle à prédire un non-évènement comme étant un non-évènement.

$$Specificity = \frac{TN}{FP + TN}$$

La courbe **ROC** (Receiver Operating Curve) est un tracé des performances du modèle (un tracé du taux de vrais positifs et du taux de faux positifs) à tous les seuils de classification. L'AUC (Area Under the roc Curve) est la mesure de toute la zone bidimensionnelle sous la courbe ROC et, en tant que telle, est une mesure de la performance du modèle à tous les seuils de classification possibles (Figure 1.8).

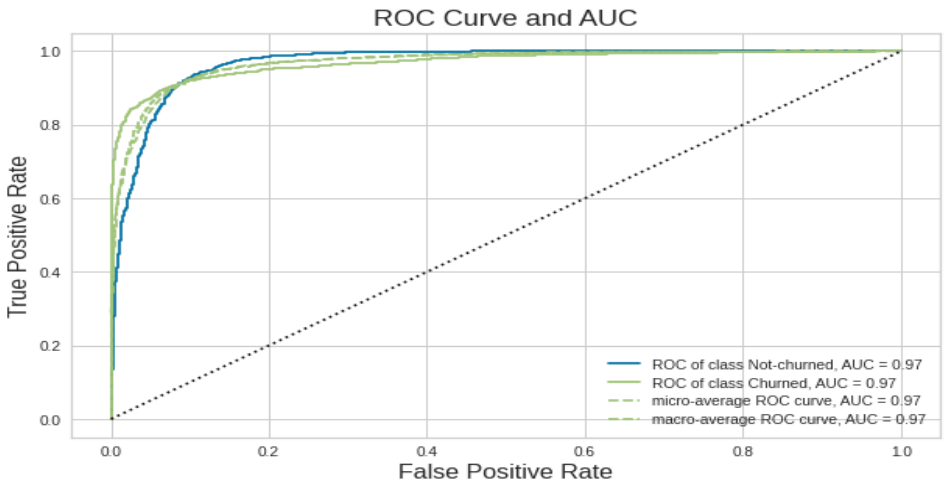


Figure 1.8- Exemple de courbe ROC et AUC [15]

1.6 Conclusion

Dans ce chapitre, nous avons défini brièvement le Machine Learning en tant que domaine de recherche en pleine expansion. Ensuite, ce chapitre a présenté les limites des modèles uniques de prédiction. Ces limites ont donné naissance aux méthodes ensemblistes : Bagging, Boosting et Stacking. Le principe de chacune de ces trois familles de méthode a été brièvement exposé. Le chapitre présente ensuite les principales mesures de performance utilisées pour évaluer les modèles de prédiction. Le chapitre suivant expose avec plus de détails le principe du Boosting et quelques algorithmes qui appliquent ce principe.

Chapitre 2 Boosting, Méthode Ensembliste Séquentielle

2.1 Introduction

Les méthodes d'ensembles séquentiels fonctionnent d'une manière itérative et récurrente, le Boosting est parmi les méthodes d'ensembles séquentielles les plus abouties. Le principe de base des algorithmes de Boosting a été exposé dans le chapitre précédent. Le succès obtenu par le premier algorithme de cette classe, AdaBoost[33], a donné lieu à la proposition de plusieurs algorithmes qui appliquent ce même principe mais avec des caractéristiques différentes. Le but de ce chapitre est d'exposer trois principaux algorithmes de Boosting.

2.2 Quelques Algorithmes de Boosting

2.2.1 AdaBoost

2.2.1.1 Historique

Une illustration claire du Boosting est donnée par AdaBoost [33] : un algorithme d'usage très courant en apprentissage automatique, dû à sa simplicité et à ses performances et dont le pseudocode est détaillé ci-dessous. L'expression d'AdaBoost est une concaténation de "Adaptative Boosting". Il construit un vote de majorité itérativement de manière adaptative. À la fin de l'algorithme, chaque classifieur h_t est pondéré par une valeur α_t calculée dans la boucle de l'algorithme. La classification d'un nouvel exemple se fait en utilisant un vote de majorité pondéré [12].

2.2.1.2 Principe

Tout d'abord, nous commençons par résumer les propriétés clés d'AdaBoost (Figure 2.1) :

1. AdaBoost utilise des souches de décision (en anglais « Stump ») comme estimateurs de base, qui peuvent être entraînées extrêmement rapidement, même avec un

grand nombre de fonctionnalités. Un « Stump » est un arbre composé uniquement de deux niveaux : la racine et ses deux fils. Les souches de décision sont des apprenants faibles. On peut comparer cela au Bagging, qui utilise des arbres de décision plus profonds, qui sont de bons apprenants.

2. AdaBoost garde une trace des poids sur des exemples de formation individuels. Ceci permet à AdaBoost d'assurer la diversité des ensembles en repondérant les exemples de formation.
3. AdaBoost garde une trace des pondérations sur les estimateurs de base individuels. Ceci est similaire aux méthodes de combinaison qui pondèrent chaque classificateur différemment.

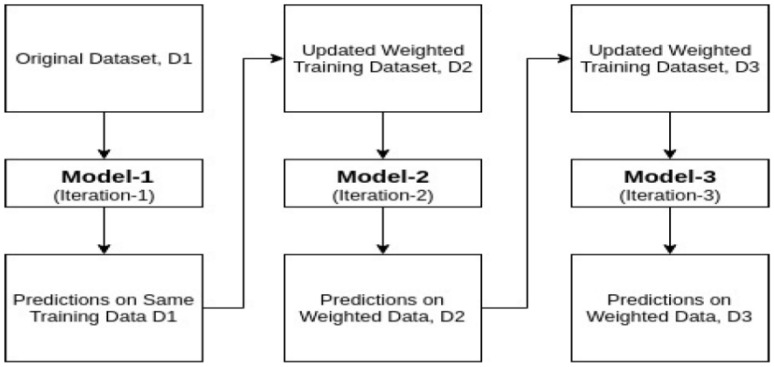


Figure 2.1- Principe AdaBoost [28]

2.2.1.3 Algorithme AdaBoost

Voici la formulation algorithmique d'AdaBoost pour apprendre un classificateur d'ensemble à partir d'apprenants faibles.

$$y=H(x) \text{ tel que } y\in\{-1,1\}.$$

Initialisation

Tout d'abord, initialiser les poids d'observation pour qu'ils soient égaux pour tous les exemples d'entraînement $w_{(i)}=\frac{1}{N}$, i=1,2,……,n ; où N est le nombre total de points de données.

Entraîner la séquence de classificateurs faibles

Répétez les étapes suivantes dans cet ordre pour m=1 à M :

1. Ajuster un classificateur $H_m(x)$ aux données d'apprentissage en utilisant les poids $w_i^{(m)}$.

2. Nous allons créer un Stump pour chacune des caractéristiques, puis calculer l'indice de Gini de chaque arbre. L'arbre avec l'indice de Gini le plus bas sera notre premier Stump.

3. Nous allons maintenant calculer la "quantité de dire" ou "l'importance" ou "l'influence" pour ce classificateur H_m en classant les points de données à l'aide de cette formule :

$$Performence\ du\ Stump(\alpha) = \frac{1}{2}\, log\ log\ \frac{1 - erreurtotale}{erreurtotale}$$

4. Nous devons mettre à jour les poids car si les mêmes poids sont appliqués au modèle suivant, la sortie reçue sera la même que celle reçue dans le premier modèle.
Les mauvaises prédictions auront plus de poids tandis que les poids des prédictions correctes seront diminués. Maintenant, lorsque nous construisons notre prochain modèle après la mise à jour des poids, une plus grande préférence sera accordée aux points avec des poids plus élevés.
Après avoir trouvé l'importance du classificateur et l'erreur totale, nous devons enfin mettre à jour les poids et pour cela, nous utilisons la formule suivante :

$$Nonveau\ poids\ d'échantillon = Ancien\ poids * e^{\pm\ Performence\ du\ Stump(\alpha)}$$

- Performance du Stump (α) sera négative lorsque l'échantillon est correctement classé.
- Performance du Stump (α) sera positive lorsque l'échantillon est mal classé.

5. Nous devons maintenant créer un nouvel ensemble de données pour voir si les erreurs ont diminué ou non.

6. Classificateur final de sortie

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \propto_m H_m(x)\right)$$

2.2.2 Gradient Boosting

2.2.2.1 Historique

L'idée de gradient Boosting est née de l'observation de Leo Breiman[4] selon laquelle le Boosting peut être interprété comme un algorithme d'optimisation sur une fonction de coût appropriée.Des algorithmes explicites de renforcement du gradient de régression ont ensuite été développés par Jérôme H. Friedman [10][11], simultanément avec la perspective plus générale de renforcement du gradient fonctionnel de Llew Mason, Jonathan Baxter, Peter Bartlett et Marcus Frean[26][27]. Les deux derniers articles ont introduit la vision des algorithmes de Boosting en tant qu'algorithmes itératifs de descente de gradient fonctionnel. C'est-à-dire des algorithmes qui optimisent une fonction de coût sur l'espace des fonctions en choisissant de manière itérative une fonction (hypothèse faible) qui pointe dans la direction du gradient négatif. Cette vision du gradient fonctionnel du Boosting a conduit au développement d'algorithmes de Boosting dans de nombreux domaines de l'apprentissage automatique et des statistiques au-delà de la régression et de la classification.

2.2.2.2 Principe

L'algorithme de Gradient Boosting a beaucoup de points communs avec AdaBoost. Tout comme AdaBoost, il s'agit d'un ensemble de "Weak Learners", créés les uns après les autres, formant un "Strong Learner". De plus, chaque "Weak Learner" est entraîné pour corriger les erreurs des "Weak Learners" précédents. Néanmoins, contrairement à AdaBoost, les "Weak Learners" ont tous autant de poids dans le système de vote, peu importe leur performance.

Nous allons détailler le fonctionnement de l'algorithme (Figure 2.2) :

- Le premier “Weak Learner” (w_1) est très basique, il s’agit tout simplement de la moyenne des observations. Il est donc très peu efficace, mais il va servir de base au reste de l’algorithme.
- Par la suite, nous calculons l’écart entre cette moyenne et la réalité que nous appelons premier résidu.
- Ainsi, le second “Weak Learner” est entraîné pour prédire le premier résidu.

Les prédictions du second “Weak Learner” sont ensuite multipliées par un facteur inférieur à 1.

L’idée derrière cette multiplication est que plusieurs petits pas sont plus précis que quelques grands pas. La multiplication réduit donc la taille des “pas” pour augmenter la précision. L’objectif étant “d’écarter” petit à petit les prédictions du modèle de la moyenne, pour les rapprocher de la réalité.

- A partir de ce moment, la création des “Weak Learners” suit toujours le même pattern :
  - A partir des dernières prédictions, on calcule les nouveaux résidus (écart entre la réalité et la prédiction).
  - On entraîne le nouveau “Weak Learner” pour prédire ces résidus.
  - On multiplie les prédictions de ce “Weak Learner” par un facteur inférieur à 1.
  - On obtient de nouvelles prédictions, souvent légèrement meilleures que les précédentes.

Pour demander la prédiction de Gradient Boosting sur une observation, il suffit d’interroger chaque “Weak Learner”. La prédiction du “Strong Learner” sera la somme de toutes les réponses des “Weak Learners”.

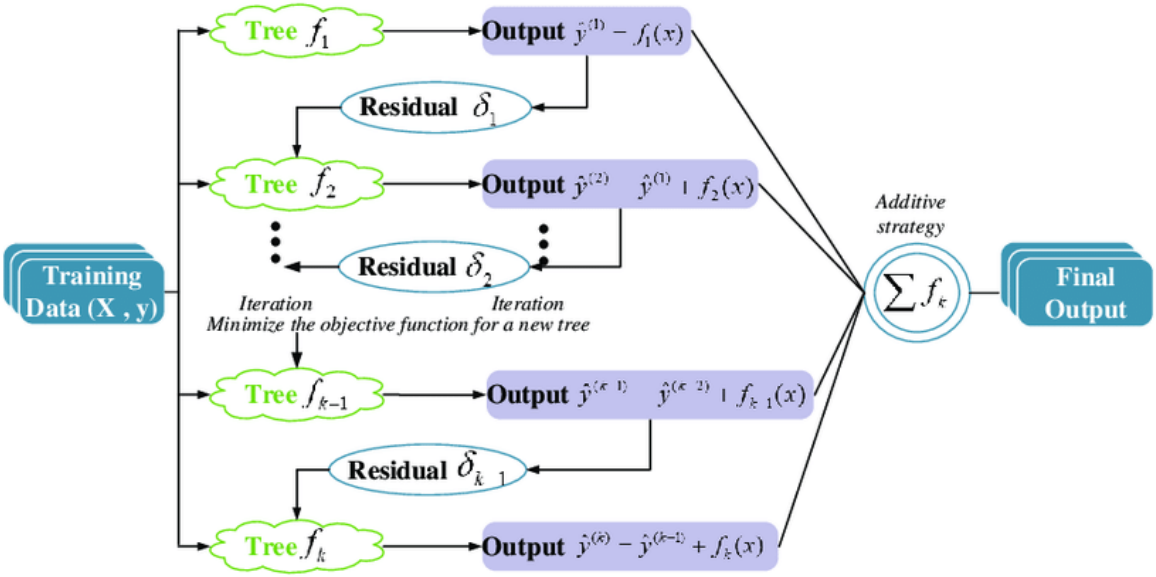


Figure 2.2- Principe du Gradient Boosting [29]

2.2.2.3 Algorithme de Gradient Boosting

1. **Entrées**

• $d_n = (x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $0 < \lambda \leq 1$

• M le nombre d'itérations.

Initialisation : $g0(\cdot) = argminc \frac{1}{n} \sum_{i=1}^{n} l(y_i, c)$

2. **Pour** m = 1 à M

2.1. Calculer l'opposé du gradient $\frac{-\partial}{\partial g(xi)} \ell (y_i, g_m(x_i))$ et l'évaluer aux points $g_{m-1}(x_i)$:

$$U_i = \frac{-\partial}{\partial g(xi)} \ell (y_i, g_m(x_i)) \Big| g(xi)=gm-1(xi), \; i = 1, \ldots, n.$$

2.2. Ajuster la règle faible de régression g_m sur l'échantillon $(x_1, U_1), \ldots, (x_n, U_n)$.

2.3. Mise à jour : $\hat{g}_m(x) = \hat{g}_m - l(x) + \lambda g_m(x)$.

3. **Sortie** : La règle $\hat{g}_M(x) = g_m(x)$.

2.2.3 LogitBoost - Booster avec la perte logistique

2.2.3.1 Historique

LogitBoost est un algorithme de Boosting formulé par Jerome Friedman, Trevor Hastie et Robert Tibshirani. L'article original place l'algorithme AdaBoost dans un cadre statistique [5]. Le développement de LogitBoost a été motivé par le désir d'apporter des fonctions de perte à partir de modèles de classification établis (tels que la régression logistique) dans le framework AdaBoost. De cette manière, le cadre général de renforcement peut être appliqué à des paramètres de classification spécifiques afin de former des ensembles boostés avec des propriétés similaires à ces classificateurs.

2.2.3.2 Principe

●Fonctions de perte logistique vs exponentielle

Premièrement, LogitBoost optimise la perte logistique

$$L(x; \alpha t) = log\ (1 + e^{(-\alpha t * y * ht(x))})$$

Pour tout x ∈ X(l'ensemble d'apprentissage) , on désire prédire la probabilité (h) que x appartienne à la classe t ∈ Y(classe binaires),

La perte logistique pénalise les erreurs différemment de la perte exponentielle.

●La régression comme algorithme d'apprentissage faible pour la classification

Deuxièmement, LogitBoost fonctionne avec des prédictions de probabilités P(x).La probabilité de prédire un exemple d'entraînement x comme un exemple positif est donnée par :

$$P(y = 1|x) = \frac{1}{1 + e^{-F(x)}}$$

Tandis que la probabilité de prédire x comme un exemple négatif est donnée par

$$P(y = 0 \mid x) = 1 - P(y = 1 \mid x).$$

Ceci influence directement notre choix d'estimateur de base.

●Troisièmement, LogitBoost fonctionne avec des probabilités de prédiction continues. Par conséquent, il utilise n'importe quel algorithme de régression comme algorithme d'apprentissage de base.

2.2.2.3 Algorithme LogitBoost

En rassemblant tous ces éléments, l'algorithme LogitBoost effectue les étapes suivantes au sein de chaque itération. La probabilité $P(y_i = 1|x_i)$ est abrégée P_i.

1. Calculer la réponse de travail, ou combien la probabilité de prédiction diffère de la vraie étiquette,

$$Z_i = \frac{Y_i - P_i}{P_i(1 - P_i)}$$

2. Mettre à jour les poids de l'exemple, $D_i = P_i\ (1 - P_i)$.

3. Entraînez un Stump de régression faible $h_t(x)$ sur les exemples pondérés (x_i, z_i, D_i).

4. Mettre à jour l'ensemble, $F_t+1(x) = F_t(x) + h_t(x)$

5. Mettre à jour les exemples de probabilités.

$$P_i = \frac{1}{1 + e^{-F_{t+1}(x)}}$$

6. Sortie du classificateur :

$$[(F(x))] = \sum_{k=1}^{k} fk(x)$$

A ce stade, le signe [F (x)] est une fonction qui a deux possibilités classes de sortie :

Sign [F(x)] ={1 , $si\ F(x) < 0$ -1, $si\ F(x) \geq 0$

Comme nous pouvons le voir à l'étape 3 ci-dessus, LogitBoost, comme AdaBoost, est un ensemble additif. Cela signifie qu'il regroupe les estimateurs de base et combine leurs prédictions de manière additive.

2.3 Conclusion

Ce chapitre a présenté quelques-uns des algorithmes de Boosting. Il existe évidemment d'autres algorithmes proposés dans le cadre du Boosting. Nous pouvons citer XGBoost et CatBoost. Tous ces algorithmes utilisent le principe de Boosting mais avec des caractéristiques différentes.

Chapitre 3 Présentation Détaillée De Trois Algorithmes De Boosting

3.2 Introduction

Dans ce chapitre, une petite recherche documentaire a été faite sur le module Scikit-learn : une bibliothèque très riche d'apprentissage automatique dans laquelle sont implémentés plusieurs algorithmes parmi eux ceux du Boosting (AdaBoost, GradientBoosting et LogitBoost). Nous utiliserons certains classificateurs tels que Support Vector Machine (SVM), Naïve Bayes ou encore DécisionTree comme estimateur de base dans l'algorithme « AdaBoost ».

3.3 Présentation du module scikit-learn (Sklearn) de Python

C'est une bibliothèque d'apprentissage statistique dans le langage de programmation Python basée sur d'autres bibliothèques : NumPy, SciPy et Matplotlib. Au début, Scikit-Learn (Sklearn) était un projet « Google Summer of Code » de David Cournapeau en 2007. En 2010, l'INRIA, l'Institut Français de Recherche en Informatique et en Automatisme, a commencé à développer ce projet et a publié la première version le 1er février 2010 [34]. Le projet repose aujourd'hui sur un effort mondial en code source ouvert rassemblant plus de 200 contributeurs.

Sklearn est la bibliothèque la plus utilisée et la plus robuste pour l'apprentissage automatique en Python. Elle fournit une sélection d'outils efficaces pour l'apprentissage automatique et la modélisation statistique, notamment la classification, la régression, le regroupement et la réduction de la dimensionnalité via une interface de cohérence en Python.

Elle comporte divers algorithmes ensemblistes de type Boosting tels que: “Adaptive Boosting (AdaBoost)”, “Gradient Boosting” , “XGBoost”, ”CatBoost”, ”LogitBoost”,” Multiple Additive Regression Trees (MART)” et ” LightGBM”.

Dans le cadre de ce projet, nous nous intéressons à l’étude de trois algorithmes de Boosting : Adaptive Boosting (AdaBoost) qui est la base du Boosting, GradientBoosting qui utilise la descendante du gradient et enfin LogitBoost basé sur la régression logistique. Ces trois algorithmes utilisent le même principe mais diffèrent au niveau de quelques caractéristiques.

L’environnement logiciel sur lequel a été construit ce travail se base sur la version 3.10.3 du langage Python et l’utilisation de plusieurs modules dont : Sklearn 1.0.2, Numpy 1.22.3, Scipy 1.8.0, Pandas 1.3.5 at Matplotlib 3.5.1.

3.4 Différents algorithmes de Boosting utilisés pour la classification

3.4.2 AdaBoostClassifier

3.4.2.3 Définition

Le classificateur AdaBoost est un méta-estimateur qui commence par ajuster un classificateur sur l'ensemble de données originales, puis ajuste des copies supplémentaires du classificateur sur le même ensemble de données, mais où les poids des instances incorrectement classées sont ajustés de sorte que les classificateurs suivants se concentrent davantage sur les données mal-classées. Les sections suivantes présentent les fonctions associées aux trois algorithmes de Boosting. Bien que ces algorithmes soient prêts à l’emploi, un grand travail d’ajustement est nécessaire afin d’obtenir les meilleures prédictions. L’ajustement se fait au niveau des principaux paramètres de la fonction associée à l’algorithme de Boosting considéré. Dans ce document, nous présenterons uniquement les principaux paramètres que nous avons fait varier afin d’améliorer les prédictions de chaque algorithme utilisé.

3.4.2.4 Fonction AdaBoostClassifier

Class sklearn.ensemble.AdaBoostClassifier (base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None) [24]

Paramètres

- **base_estimator** : object, default=None
  - L'estimateur de base à partir duquel l'ensemble Boosté est construit. La prise en charge de la pondération des échantillons est requise, ainsi que les attributs classes_ et n_classes_ appropriés.
  - Si aucun, alors base_estimator = DecisionTreeClassifier (max_depth= 1)

- **n_estimators** : int, par défaut=50

Le nombre maximal d'estimateurs auquel l'amplification est terminée. En cas d'ajustement parfait, la procédure d'apprentissage est arrêtée prématurément.

- **learning_rate** : float, par défaut =1.0

Pondération appliquée à chaque classifieur à chaque itération de Boosting. Un taux d'apprentissage plus élevé augmente la contribution de chaque classifieur. Il existe un compromis entre les paramètres learning_rate et n_estimators.

- **algorithm**: {'SAMME', 'SAMME.R'}, default='SAMME.R'
  - Si 'SAMME.R', utilisez l'algorithme de Boosting réel SAMME.R. base_estimator doit prendre en charge le calcul des probabilités de classe.
  - Si 'SAMME', utilisez l'algorithme de Boosting discret SAMME. L'algorithme SAMME.R converge généralement plus rapidement que SAMME, obtenant une erreur de test inférieure avec moins d'itérations de Boosting.

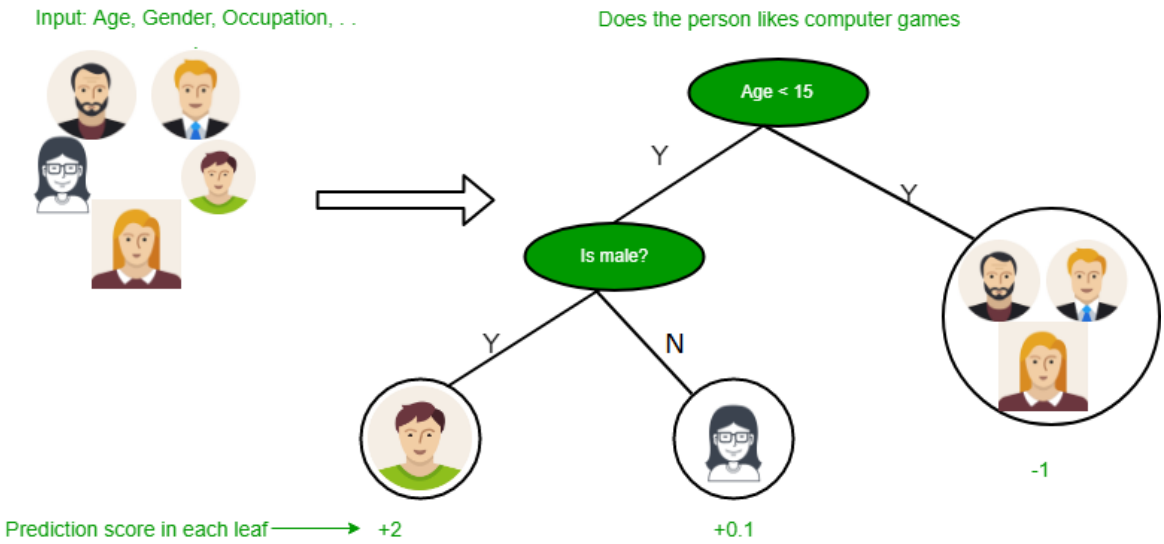
3.4.2.5 Les classifieur de base utilisés Dans la fonction AdaBoostClassifier

Il existe différents classificateurs dans les méthodes d'apprentissage supervisé qui sont utilisés pour l'analyse de Dataset, certains d'entre eux sont les suivants :

3.4.2.5.1 Arbre de Décision (DecisionTreeClassifier)

L'apprentissage par arbre de décision est une méthode classique en apprentissage automatique. Son but est de créer un modèle qui prédit la valeur d'une variable-cible depuis la valeur de plusieurs variables d'entrée. Une des variables d'entrée est sélectionnée à chaque nœud intérieur (ou interne, nœud qui n'est pas terminal) de l'arbre selon une méthode qui dépend de l'algorithme. Chaque arête vers un nœud-fils correspond à un ensemble de valeurs d'une variable d'entrée, de manière que l'ensemble des arêtes vers les nœuds-fils couvrent toutes les valeurs possibles de la variable d'entrée (figure 3.1).

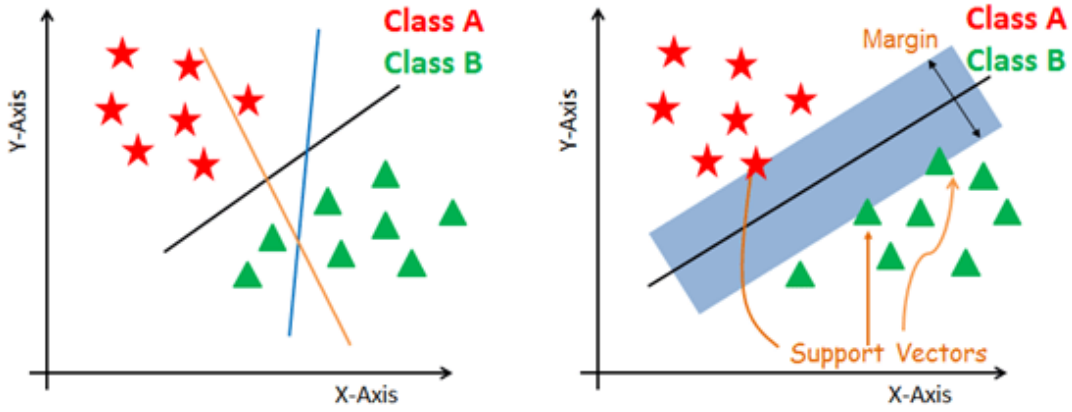
Chaque feuille (ou nœud terminal de l'arbre) représente soit une valeur de la variable-cible, soit une distribution de probabilité des diverses valeurs possibles de la variable-cible. La combinaison des valeurs des variables d'entrée est représentée par le chemin de la racine jusqu'à la feuille.



- Figure 3.1- Exemple de classification par arbre de décision [25]

3.4.2.5.2 Les Machines à Vecteurs de Support (Support Vector Machines SVM)

SVM est une méthode dont l'objectif principal est de trouver des séparateurs linéaires qui séparent entre les différentes classes. SVM trouve un hyperplan avec la plus grande marge possible [35]. SVM prend en entrée un grand nombre d'exemples et les affecte à différentes classes. Les données formées qui sont des exemples considérés comme des points dans l'espace (figure 3.2). Le point que nous avons obtenu appartient à l'une des classes qui sont pré-séparées et la distance obtenue est appelée hyperplan. Quand nous avons une nouvelle donnée d'entrée, elle sera assignée à l'une des classes après une séparation par la marge. Plus la marge est grande, plus l'erreur de généralisation du classificateur est faible [31].



- Figure 3.2- Exemple de classification SVM [36]

3.4.2.5.3 Le Classifieur : Gaussian Naive Bayes

L'algorithme des réseaux bayésiens classifie les instances données (classes dans notre cas) par la construction d'un graphe orienté, où des modèles de risque seront représentés par les nœuds et leurs indépendances sous forme de liens [16] (figure 3.3). Il peut être utilisé sous différentes variantes. L'une des plus populaires est la Naïve Bayes Network que nous avons choisie pour notre étude qui applique des assomptions d'indépendances moins fortes que le classificateur Naïve Bayes.

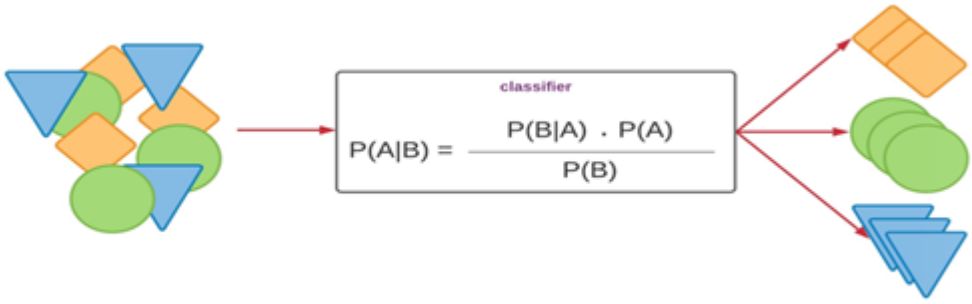


Figure 3.3- Exemple de classification GaussianNB [21]

3.4.3 GradientBoostingClassifier

3.4.3.3 Définition

L'algorithme Gradient Boosting construit un modèle additif de manière progressive; il permet l'optimisation de fonctions de perte différentiables arbitraires. À chaque étape, n_classes_ arbres de régression sont ajustés sur le gradient négatif de la fonction de perte de déviance binomiale ou multinomiale. La classification binaire est un cas particulier où un seul arbre de régression est induit.

3.4.3.4 Fonction GradientBoostingClassifier

class sklearn.ensemble.GradientBoostingClassifier (*, loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0) [22]

Paramètres

- **loss :** {'log_loss', 'deviance', 'exponential'}, default='log_loss'

La fonction de perte à optimiser « log_loss » fait référence à la déviance binomiale et multinomiale, la même que celle utilisée dans la régression logistique. C'est un bon

choix pour la classification avec des résultats probabilistes. Pour la perte « exponentielle », l'amplification de gradient récupère l'algorithme AdaBoost.

●**learning_rate :** float, par défaut=0.1

Le taux d'apprentissage réduit la contribution de chaque arbre de learning_rate. Il existe un compromis entre learning_rate et n_estimators.

●**n_estimators** : int, par défaut=100

Le nombre d'étapes de Boosting à effectuer. Le Boosting de gradient est assez robuste au sur-ajustement, donc un grand nombre se traduit généralement par de meilleures performances.

●**Subsample :** float, par défaut=1.0

La fraction d'échantillons à utiliser pour ajuster les apprenants de base individuels. S'il est inférieur à 1.0, cela entraîne une amplification du gradient stochastique. Subsample interagit avec le paramètre n_estimators. Le choix d'un Subsample < 1,0 entraîne une réduction de la variance et une augmentation du biais.

●**Criterion :** {'friedman_mse', 'mse', 'mae'}, default='friedman_mse'

La fonction pour mesurer la qualité d'un split. Les critères pris en charge sont « friedman_mse » pour l'erreur quadratique moyenne avec le score d'amélioration de Friedman, « mse » pour l'erreur quadratique moyenne et « mae » pour l'erreur absolue moyenne. La valeur par défaut de 'friedman_mse' est généralement la meilleure car elle peut fournir une meilleure approximation dans certains cas.

●**min_samples_split :** int ou float, par défaut=2

Le nombre minimum d'échantillons requis pour diviser un nœud interne :

- Si int, alors considérez min_samples_split comme le nombre minimum.

- Si float, alors min_samples_split est une fraction et ceil (min_samples_split * n_samples) est le nombre minimum d'échantillons pour chaque division

●**min_samples_leaf :** int ou flottant, par défaut=1

Le nombre minimum d'échantillons requis pour être à un nœud feuille. Un point de partage à n'importe quelle profondeur ne sera pris en compte que s'il laisse au moins min_samples_leaf échantillons d'apprentissage dans chacune des branches gauches et droite. Cela peut avoir pour effet de lisser le modèle, notamment en régression.

- Si int, alors considérez min_samples_leaf comme le nombre minimum.

- Si float, alors min_samples_leaf est une fraction et ceuil (min_samples_leaf * n_samples) est le nombre minimum d'échantillons pour chaque nœud.

●**min_weight_fraction_leaff :** float, par défaut=0.0

La fraction pondérée minimale de la somme totale des poids (de tous les échantillons d'entrée) devant être à un nœud feuille. Les échantillons ont le même poids lorsque sample_weight n'est pas fourni.

●**max_depth :** int, défaut=3

Profondeur maximale des estimateurs de régression individuels. La profondeur maximale limite le nombre de nœuds dans l'arbre. Ajustez ce paramètre pour de meilleures performances ; la meilleure valeur dépend de l'interaction des variables d'entrée.

●**min_impurity_decrease :** float, par défaut=0.0

Un nœud sera dédoublé si ce dédoublement induit une diminution de l'impureté supérieure ou égale à cette valeur.

●**min_impurity_splitfloat** : default=None

Seuil pour l'arrêt précoce de la croissance de l'arbre. Un nœud se divise si son impureté est supérieure au seuil, sinon il est une feuille.

●**Initestimator or 'zero'**: default=None

Objet estimateur utilisé pour calculer les prédictions initiales. init doit fournir fit et predict_proba. Si "zéro", les prédictions brutes initiales sont mises à zéro. Par défaut, un DummyEstimator prédisant les a priori des classes est utilisé.

●**random_state :** int, instance RandomState ou None, default=None

Contrôle la graine aléatoire donnée à chaque estimateur Tree à chaque itération de Boost. De plus, il contrôle la permutation aléatoire des caractéristiques à chaque fractionnement. Il contrôle également le fractionnement aléatoire des données d'apprentissage pour obtenir un ensemble de validation si n_iter_no_change n'est pas None. Passez un int pour une sortie reproductible sur plusieurs appels de fonction.

●**max_features :** {'auto', 'sqrt', 'log2'}, int ou float, default=None

Le nombre de fonctionnalités à prendre en compte lors de la recherche de la meilleure répartition :

- Si int, alors considérez les fonctionnalités max_features à chaque divison.
- Si float, alors max_features est une fraction et int (max_features * n_features) les fonctionnalités sont prises en compte à chaque fractionnement.
- Si 'auto', alors max_features= sqrt (n_features).
- Si 'sqrt', alors max_features= sqrt (n_features).
- Si 'log2', alors max_features=log2 (n_features).
- Si aucun, alors max_features=n_features.

●**Verbose** : int, default=0

Activer la sortie détaillée. Si 1, il imprime les progrès et les performances de temps en temps (plus il y a d'arbres, plus la fréquence est faible). S'il est supérieur à 1, il imprime la progression et les performances de chaque arbre.

- **max_leaf_nodes :** int, par défaut=Aucun

Cultivez des arbres avec max_leaf_nodes de la manière la plus efficace. Les meilleurs nœuds sont définis comme une réduction relative des impuretés. Si Aucun, alors nombre illimité de nœuds feuilles.

- **warm_start :** bool, défaut=Faux

Lorsqu'il est défini sur True, réutilisez la solution de l'appel précédent pour ajuster et ajouter plus d'estimateurs à l'ensemble, sinon, effacez simplement la solution précédente.

- **validation_fraction :** float, défaut=0.1

La proportion de données d'entraînement à mettre de côté comme ensemble de validation pour l'arrêt précoce. Doit être compris entre 0 et 1. Utilisé uniquement si n_iter_no_change est défini sur un entier.

- **n_iter_no_change :** int, par défaut=Aucun

n_iter_no_change est utilisé pour décider si un arrêt précoce sera utilisé pour terminer l'entraînement lorsque le score de validation ne s'améliore pas. Par défaut, il est défini sur Aucun pour désactiver l'arrêt anticipé. S'il est défini sur un nombre, il mettra de côté la taille validation_fraction des données d'entraînement en tant que validation et terminera l'entraînement lorsque le score de validation ne s'améliore pas dans tous les nombres d'itérations n_iter_no_change précédents. La division est stratifiée.

- **Tol :** float, par défaut=$1e^{-4}$

Tolérance pour l'arrêt précoce. Lorsque la perte ne s'améliore pas d'au moins tol pour n_iter_no_change itérations (si défini sur un nombre), l'apprentissage s'arrête.

- **ccp_alpha :** flottant non négatif, par défaut=0.0

Paramètre de complexité utilisé pour l'élagage coût-complexité minimal. Le sous-arbre avec la plus grande complexité de coût inférieure à ccp_alpha sera choisi. Par défaut, aucun élagage n'est effectué. Voir Élagage de complexité de coût minimal pour plus de détails.

3.4.4 LogitBoostClassifier

3.4.4.3 Définition

Un classificateur LogitBoost est un méta-estimateur qui ajuste un modèle additif minimisant une fonction de perte logistique.

3.4.4.4 La fonction LogitBoostClassifier

class logitBoost.LogitBoost (base_estimator=None, n_estimators=50, weight_trim_quantile=0.05, max_response=4.0, learning_rate=1.0, bootstrap=False, random_state=None).[23]

Paramètres

- **base_estimator (objet, facultatif (par défaut = None))**

L'estimateur de base à partir duquel le classificateur LogitBoost est construit. Cela devrait être un régresseur. Si aucun estimateur de base n'est spécifié, une souche de décision est utilisée.

- **n_estimators (int, optionnel (default=50))**

Le nombre d'estimateurs par classe dans l'ensemble.

- **weight_trim_quantile (float, optionnel (default=0.05))**

Seuil pour l'ajustement du poids. La distribution des poids a tendance à devenir très biaisée dans les itérations de renforcement ultérieures, et les observations avec des poids faibles contribuent peu à l'ajustement de l'estimateur de base à cette itération. À chaque itération de renforcement, les observations dont le poids est inférieur à ce quantile de la distribution des poids de l'échantillon sont supprimées des données pour ajuster l'estimateur de base (pour cette itération uniquement) afin d'accélérer le calcul.

- **max_response (float, optionnel (default=4.0))**

Valeur de réponse maximale à autoriser lors de l'ajustement des estimateurs de base (pour la stabilité numérique). Les valeurs seront coupées à l'intervalle [max_response, max_response].

- **learning_rate (float, optionnel (default=1.0))**

Le taux d'apprentissage réduit la contribution de chaque classifieur par learning_rate lors de l'ajustement.

- **bootstrap (bool, optionnel (default=False))**

  - Si True, chaque itération de Boost forme l'estimateur de base à l'aide d'un échantillon bootstrap pondéré des données d'apprentissage.

  - Si False, chaque itération d'amplification entraîne l'estimateur de base à l'aide de l'échantillon d'apprentissage complet (pondéré). Dans ce cas, l'estimateur de base doit prendre en charge la pondération d'échantillon au moyen d'un paramètre sample_weight dans sa méthode fit ().

- **random_state (int, RandomState instance ou None, optionnel (default=None))**

- Si int, random_state est la graine utilisée par le générateur de nombres aléatoires.

- Si instance RandomState, random_state est le générateur de nombres aléatoires.

- Si aucun, le générateur de nombres aléatoires est l'instance RandomState utilisée par numpy.random.

3.5 Conclusion

Dans ce chapitre, nous avons décrit les trois fonctions AdaBoostClassifier, GradientBoostingClassifier et LogitBoostClassifier telles qu'elles sont implémentées dans la bibliothèque Scikit-Learn. Ces fonctions seront expérimentées sut des Datasets afin de mettre en évidence l'apport du Boosting dans l'élaboration de modèles de prédiction efficaces. Ceci fera l'objet du chapitre suivant.

Chapitre 4 Expérimentations Et Résultats

4.2 Introduction

Après avoir passé en revue les trois algorithmes de Boosting choisis dans le cadre de ce projet, il est temps maintenant de les tester sur des Datasets expérimentaux. Ces Datasets seront d'abord présentés avant de passer à la description de l'environnement logiciel utilisé comme support de cette étude expérimentale.

Le reste du chapitre sera consacré à la présentation des résultats obtenus par les différents algorithmes de Boosting. Les principaux paramètres de chaque algorithmes (hyperparameters) sont ajustés afin d'améliorer la puissance prédictive. Les résultats des expérimentations sont présentés dans des tableaux récapitulatifs.

4.3 Datasets utilisées et Description des Attributs

4.3.2 Statlog Dataset

Statlog est un Dataset expérimental utilisé pour la construction de modèles de prédiction de la maladie du cœur (maladie coronarienne). Les données de ce Dataset ont été récoltées par l'université de California Irvine, USA. Il se compose de 13 variables prédictives médicales (qui ont été extraites d'un ensemble plus vaste de 75) concernant 270 patients dont 120 sont atteints par la maladie, et d'une variable cible à prévoir Absence (1) ou Présence (2) de la maladie coronarienne (tableau 4.1) [7].

Le cœur est une pompe musculaire qui fait circuler le sang dans tout le corps à travers un réseau de vaisseaux appelés « artères ». La maladie coronarienne est une maladie qui touche les artères ayant pour fonction d'alimenter le cœur en sang (artères coronaires, figure 4.1). Elle est souvent causée par l'athérosclérose, une accumulation de plaques à l'intérieur de la paroi des artères (mauvais cholestérol). Cette accumulation rétrécit peu à peu l'intérieur des artères et ralentit le flux 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 [17].

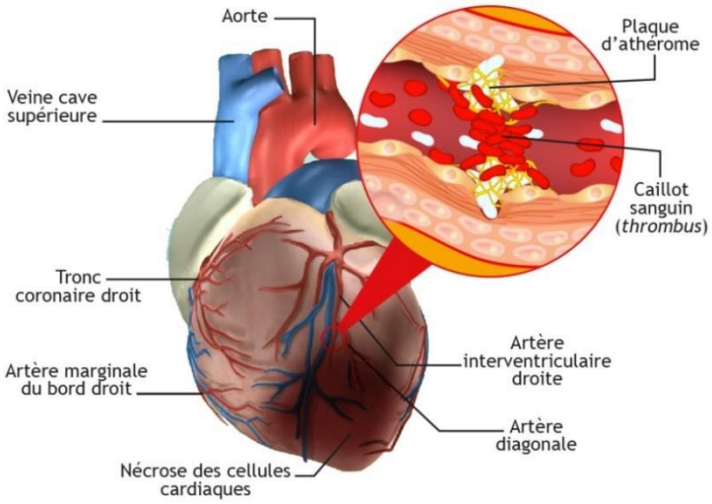


Figure 4.1- Maladie Coronarienne

4.3.3 Description des variables

Tableau 4.1- Description des attributs du Dataset Statlog [7]

Attributs	Description	Type	Valeur
Age (age)	âge du patient en année.	Entier	[29-77]
Sexe (gender)	patient homme ou femme.	Discret	2= masculin 1= féminin
Douleur (cp)	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 (chol)	Cholestérol sérique en mg/dl.	Entier	[126-564]
Glycémie (fbs)	Glycémie à jeun (>120 mg/dl) (fasting blood sugar).	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
Dépression (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
class	variable de classe indique Absence (1) ou présence (2) de maladie coronarienne.	Discret	2=oui 1=non

4.3.4 Breast Cancer Dataset

Ce Dataset concerne l'un des cancers les plus répandus au monde : le cancer du sein. C'est un ensemble de données extrait à l'université de Wisconsin, USA. Les données sont collectées à partir du référentiel d'apprentissage automatique de l'UCI qui est accessible au public. Le Dataset comporte les enregistrements de 569 patients avec 212 diagnostics Malins, et 357 diagnostics Bénins. Il se compose de 10 variables prédictives médicales et d'une variable cible « Diagnosis » (M = malignant, B = benign) (tableau 4.2) [19].

Le cancer du sein résulte d'un dérèglement de certaines cellules qui se multiplient et forment le plus souvent une masse appelée tumeur. Il en existe différents types qui n'évoluent pas de la même manière. Certains sont « agressifs » et évoluent très rapidement, d'autres ont une évolution plus lente. Les cellules cancéreuses peuvent rester dans le sein. Elles peuvent aussi se propager dans d'autres organes ce qui est une situation encore plus menaçante. On parle alors de métastases. Dans la majorité des cas, le développement d'un cancer du sein prend plusieurs mois, voire plusieurs années (figure 4.2).

Le cancer du sein est le cancer le plus fréquent chez la femme. Il représente plus du tiers de l'ensemble des nouveaux cas de cancer chez la femme. Lorsqu'une anomalie est découverte lors d'un examen de dépistage ou qu'une personne présente des symptômes, plusieurs examens doivent être réalisés. C'est l'examen anatomopathologique des tissus prélevés au niveau de l'anomalie qui établit le diagnostic de cancer du sein. Ce prélèvement au niveau de l'anomalie est le plus souvent réalisé par micro ou macro-biopsies à travers la peau.

Différents types de traitements peuvent être utilisés pour soulager un cancer du sein : la chirurgie, la radiothérapie, l'hormonothérapie, la chimiothérapie et les thérapies ciblées [18].

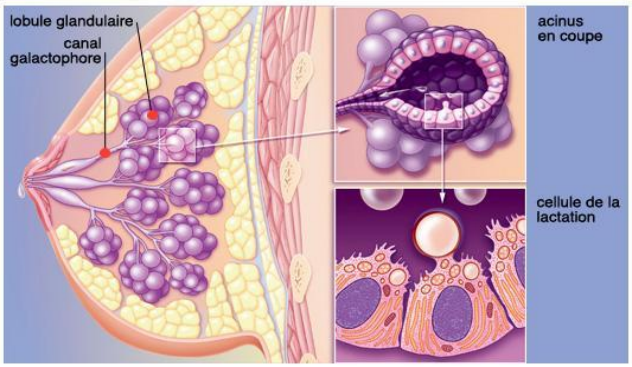


Figure 4.2- Cancer du Sein

4.3.5 Description des variables

Dix caractéristiques à valeur réelle (moyenne, SE et pire) sont calculées pour chaque noyau cellulaire :

Tableau 4.2- Description des attributs du Dataset Breast Cancer [7]

Attributs	Description	Types
Id	Numéro du noyau cellulaire	Entier
Radius	moyenne des distances du centre aux points du périmètre	Réel
Texture	écart type des valeurs d'échelle de gris	Réel
Perimeter	Périmètre du noyau cellulaire	Réel
Area	Surface du noyau cellulaire	Réel
Smoothness	variation locale des longueurs de rayon	Réel
Compactness	(périmètre^2 / surface - 1.0)	Réel
Concavity	sévérité des parties concaves du contour	Réel
concave points	nombre de parties	Réel
	concaves du contour	
Symmetry	Symétrie du noyau cellulaire	Réel
fractal_dimension	« approximation du littoral » - 1	Réel
Diagnosis	Diagnostic de présence ou absence du cancer (1 = malin, 0 = bénin)	Discret

4.4 Expérimentations Et Résultats

4.4.2 Prétraitement des données

Tout d'abord, nous avons procédé à la conversion des sources de données du format original (.xls) au format « .csv » plus couramment utilisé. Ensuite, les données catégorielles dont les valeurs étaient exprimées par des constantes textuelles ont été encodées par des constantes entières. La première colonne de chaque Dataset a été supprimée car elle contient les index des individus et ne fait pas partie des caractéristiques à prendre en considération par le modèle de prédiction.

4.4.3 Génération d'un modèle de prédiction

Un modèle de prédiction se base sur un algorithme du machine learning. L'algorithme est entrainé en utilisant des données disponibles et le modèle obtenu est utilisé pour faire de la prédiction (diagnostic) sur de nouvelles données. Ceci résume le processus de génération d'un modèle de prédiction (ou modèle d'aide à la décision). Ce processus passe par des étapes clés :

a. Train/Test Split

Le fractionnement du Dataset est essentiel pour une évaluation impartiale des performances de prédiction. Dans la plupart des cas, il suffit de diviser aléatoirement le jeu de données en deux sous-ensembles (Figure 4.3) :

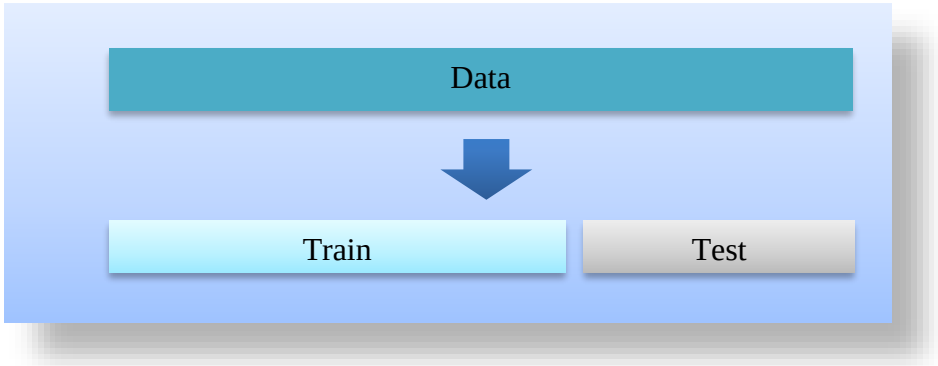


Figure 4.3- Répartition des données en Train/Test

- **Train set :** ensemble d'entraînement est appliqué pour entraîner ou ajuster le modèle.
- **Test set :** est nécessaire pour une évaluation impartiale du modèle final.

Nous avons utilisé la méthode « train_test_split » importé de la bibliothèque Sklearn, son code standard est donné dans la Figure 4.4 :

Test_size = 0.2 à l'intérieur de la fonction indique le pourcentage des données qui doivent être conservées pour le test, et le reste 80% pour l'entrainement.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 2 )
```

Figure 4.4 - Fractionnement de l'ensemble de données

b. Générer le modèle de prédiction

Dans cette étape nous avons utilisé la méthode standard « fit () », pour générer n'importe quel modèle à partir d'un algorithme de classification. Importée de la bibliothèque Sklearn, son code standard est donné par : modèle.fit (X_train, y_train)

c. Gross Validation

Pour évaluer les performances de chacun des trois algorithmes étudié, nous avons calculé la Cross Validation dont le code est donné dans la Figure 4.5

```
from sklearn.model_selection import cross_val_score
scores = evaluate_model (model)
```

Figure 4.5 – Evalué un model en utilisant CV

d. Mesures de Performance

Déjà mentionnées dans le chapitre1, ces mesures sont utilisées pour évaluer les performances du modèle de classification. Elles montrent à quel point on peut se fier à ce modèle de prédiction.

La matrice de confusion matrix peut être obtenue à travers la bibliothèque Sklearn, son code standard est donné dans la Figure 4.6

```
sklearn.metrics.confusion_matrix(y_true, y_pred, *, labels=None, sample_weight=None, normalize=None)
```

Figure 4.6 - Obtenir la matrice de confusion d'un modèle généré

A partir de la matrice de confusion, les principales mesures de performance du modèle peuvent être calculées : Accuracy, Precision, Specificity, Recall et F1-Score. Après cela, la courbe ROC peut être obtenue et l'AUC calculé.

4.4.4 Hyper-paramétrage

Dans un modèle de Machine Learning, on appelle hyper-paramètres les paramètres de l'algorithme qui permettent d'ajuster les performances du modèle pendant la phase d'apprentissage. Une fois les hyper-paramètres déterminés, l'algorithme fixe les autres paramètres optimaux à partir des données d'apprentissage.

La technique pour optimiser les hyper-paramètres est la méthode RandomizedSearchCV, qui implémente une méthode "fit" et une méthode "score". Son code standard est mentionné dans la Figure 4.7.

```
class sklearn.model_selection.RandomizedSearchCV(estimator, param_distributions, *, n_iter=10, scoring=None,
n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan,
return_train_score=False)
```

Figure 4.7- RandomizedSearchCV Avec Sklearn

Cette méthode implémente également "score_samples", "predict", "predict_proba", "decision_function", "transform" et "inverse_transform" s'ils sont implémentés dans l'estimateur utilisé. Les paramètres de l'estimateur utilisé pour appliquer ces méthodes sont optimisés par une recherche à validation croisée sur les paramètres. Le paramètre CV

détermine la stratégie de fractionnement de la validation croisée, l'entrée « None » pour utiliser la validation croisée 5 fois par défaut.

Contrairement à Grid SearchCV, toutes les valeurs de paramètre ne sont pas testées, mais un nombre fixe de paramètres est échantillonné à partir des distributions spécifiées. Le nombre de paramétrages essayés est donné par n_iter.

4.4.5 Expérimentations et Résultats

Le tableau ci-dessous, montre les résultats des performances des trois classifieurs (SVM, DécisionTree, GaussianNB) sur les deux Datasets utilisés. Ces algorithmes sont utilisés comme estimateur de base dans l'algorithme AdaBoost avec les deux Datasets. Les paramètres de chaque classifieur sont pris par default.

Tableau 4.3 - Résultats des Performances des 3 Algorithmes (SVM, DecisionTree, GaussianNB) sur les deux Datasets utilisés

Dataset	Classifier	Accuracy	Precison	Recall	F1-score
Statlog	SVM	0.71	0.73	0.69	0.69
	DécisionTree	0.79	0.79	0.79	0.79
	GaussianNB	0.88	0.88	0.88	0.88
Breast Cancer	SVM	0.95	0.96	0.93	0.94
	DécisionTree	0.91	0.92	0.90	0.91
	GaussianNB	0.93	0.95	0.91	0.92

D'après le tableau ci-dessus, le modèle GaussianNB a donné la meilleure Accuracy qui égal à 88% suivi de DécisionTree avec une Accuracy égale à 79%, et enfin SVM avec 71% en utilisant «Statlog » comme Dataset.

Pour la Dataset « Breast Cancer », SVM a bien ajusté le modèle suivi de GaussianNB et de DécisionTree, avec une Accuracy égale à 95%, 93%, et 91% respectivement.

4.4.5.3 AdaBoost

Cette section fournit les tableaux qui présentent quelque résultats des expériences pour chaque ensemble de données, en utilisant l'algorithme AdaBoost avec les paramètres suivant :

- Base_estimator : SVM, DécisionTree, GaussianNB.
- N_estimator : de 50 jusqu'à 500 itérations.
- Learning rate : entre 0.1 et 1.
- Algorithme : 'SAMME' et 'SAMME.R'.

Tableau 4.4 - Expérimentation d'AdaBoost sur le Dataset « Statlog » avec « DecisionTree» comme base estimator

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy (Acc)
Statlog	DecisionTree	Learning_rate = 0.1 Algorithme = 'SAMME'	50	0.89
			100	0.90
			150	0.90
			200	0.90
		N_estimator = 100 Algorithme = 'SAMME'	Learning_rate	Accuracy (Acc)
			0.1	0.90
			0.2	0.89
			0.3	0.89
			0.8	0.90
		N_estimator = 100 Learning_rate = 0.1	Algorithme	Accuracy (Acc)
			SAMME	0.90
			SAMME.R	0.89

Les meilleurs résultats sont obtenus avec les paramètres suivants :

N_estimator = 100, Learning rate = 0.1, Algorithm = 'SAMME' avec Accuracy = 0.90.

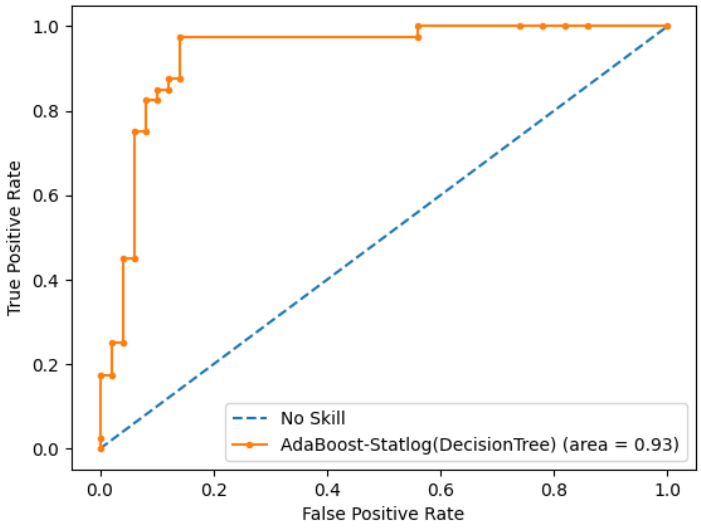


Figure 4.8 - Courbe ROC « AdaBoostClassifier » avec base estimator « DecisionTree » (sur le Dataset « Statlog »).

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy(Acc)
Statlog	Support Vector Machine (SVM) SVC (Kernel='Linear', C=0.02)	Learning_rate = 0.1 Algorithme = 'SAMME'	50	0.76
			100	0.76
			150	0.76
			200	0.76
		N_estimator = 100 Algorithme = 'SAMME'	Learning_rate	Accuracy(Acc)
			0.1	0.76
			0.3	0.74
			0.6	0.73
			1.0	0.72

Tableau 4.5 - Expérimentation d'AdaBoost Du Dataset sur le Dataset « Statlog » avec « SVM » comme base estimator

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 100, Learning rate = 0.1, Algorithm = 'SAMME' avec Accuracy = 0.76.

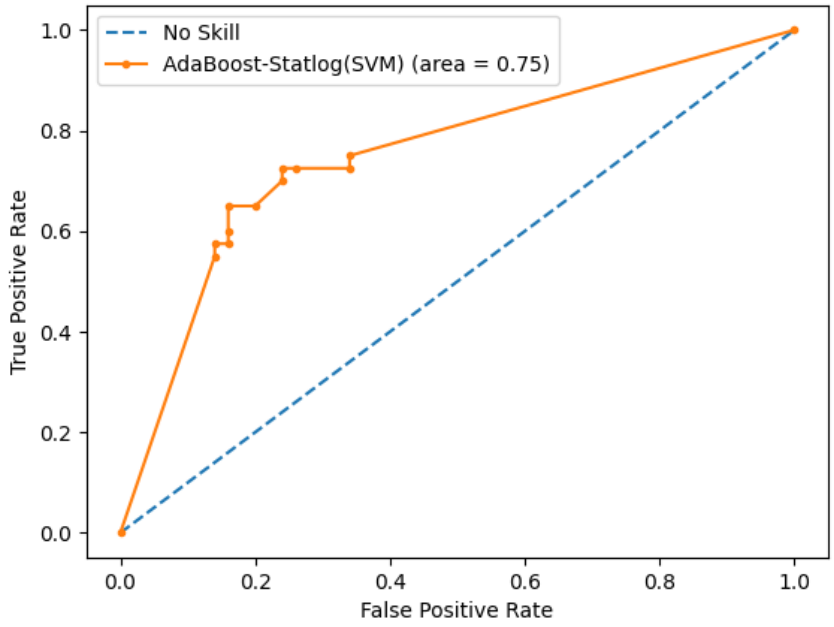


Figure 4.9 – Courbe ROC « AdaBoostClassifier » avec base estimator « SVM » (sur le Dataset « Statlog »).

Tableau 4.6 – Expérimentation d'AdaBoost Du Dataset sur le Dataset « Statlog » avec « GaussianNB » comme base estimator

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy (Acc)
Statlog	GaussianNB	Learning_rate = 0.1 Algorithme = 'SAMME'	50	0.88
			100	0.88
			150	0.88
			200	0.88
		N_estimator = 100 Algorithme = 'SAMME'	Learning_rate	Accuracy (Acc)
			0.1	0.87
			0.2	0.88
			0.4	0.84
			0.8	0.82
		N_estimator = 100 Learning_rate = 0.2	Algorithme	Accuracy (Acc)
			SAMME	0.88
			SAMME.R	0.81

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 100, Learning rate = 0.2, Algorithm = ‘SAMME’ avec Accuracy = 0.88.

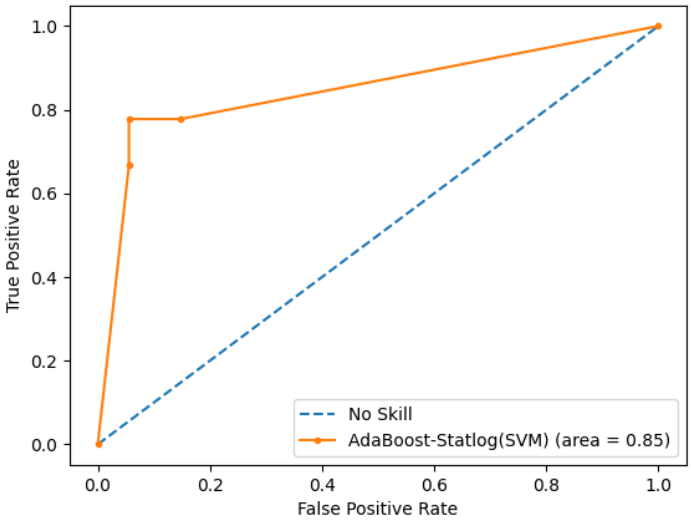


Figure 4.10 – Courbe ROC « AdaBoostClassifier » avec base estimator « GaussianNB » (sur le Dataset « Statlog »).

Tableau 4.7 – Expérimentation d’AdaBoost Du Dataset sur le Dataset « Breast Cancer » avec « DecisionTree » comme base estimator

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy (Acc)
Breast Cancer	DecisionTree	Learning_rate = 0.1 Algorithme = ’SAMME’	50	0.96
			100	0.98
			150	0.97
			200	0.97
		N_estimator = 100 Algorithme = ’SAMME’	Learning_rate	Accuracy (Acc)
			0.6	0.97
			0.8	0.97
			0.9	0.98
			1.0	0.97
		N_estimator = 100 Learning_rate = 0.1	Algorithme	Accuracy (Acc)
			SAMME	0.98
			SAMME.R	0.97

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 100, Learning rate = 0.9, Algorithm = ‘SAMME’ avec Accuracy = 0.98.

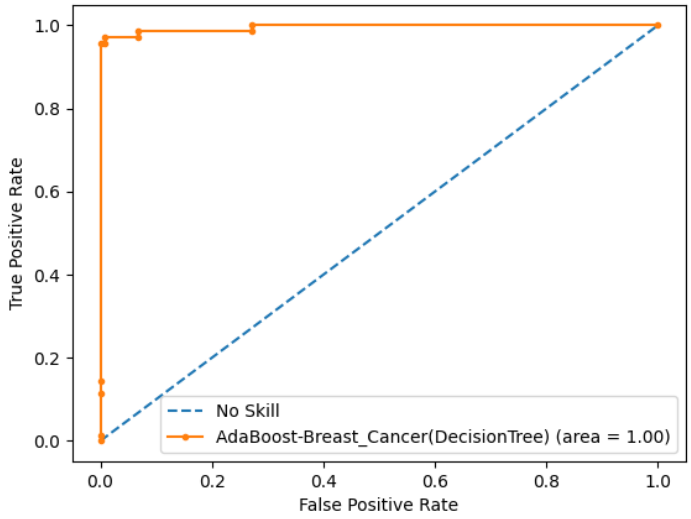


Figure 4.11- Courbe ROC « AdaBoostClassifier » avec base estimator « DecisionTree » (sur le Dataset « Breast Cancer »).

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy(Acc)
Breast Cancer	Support Vector Machine (SVM) SCV(Kernel=’Linear’, C=0.02)	Learning_rate = 0.9 Algorithme=’SAMME’	50	0.98
			100	0.98
			150	0.98
			200	0.98
		N_estimator = 50 Algorithme=’SAMME’	Learning_rate	Accuracy(Acc)
			0.1	0.97
			0.6	0.97
			0.9	0.98
			1.0	0.98

Tableau 4.8 – Expérimentation d’AdaBoost sur le Dataset « Breast Cancer » avec « SVM » comme base estimator

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 50, Learning rate = 0.9, Algorithm = ‘SAMME’ avec Accuracy = 0.98.

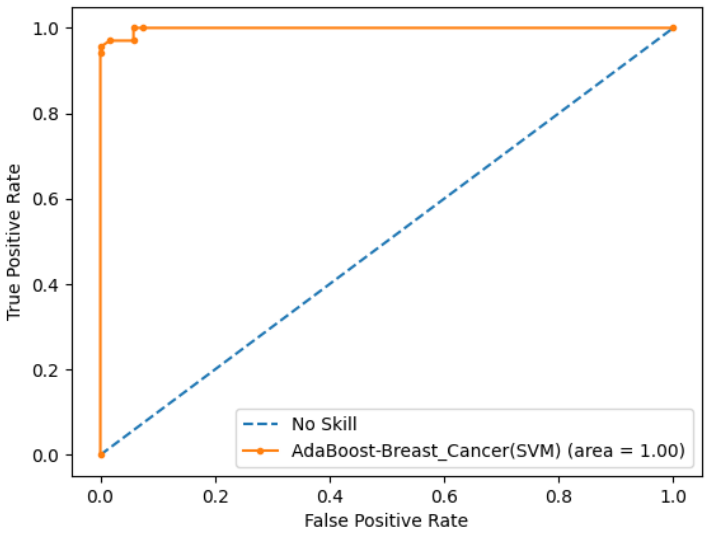


Figure 4.12 - Courbe ROC « AdaBoostClassifier » avec base estimator «SVM» (sur le Dataset « Breast Cancer »).

Tableau 4.9- Expérimentation AdaBoost sue le Dataset «Breast Cancer » avec « GaussianNB» comme base estimator

Dataset	Base estimator	Paramètres par default	N_estimator	Accuracy (Acc)
Breast Cancer	GaussianNB	Learning_rate = 0.8 Algorithme = 'SAMME'	50	0.97
			100	0.97
			150	0.97
			200	0.97
		N_estimator = 150 Algorithme = 'SAMME'	Learning_rate	Accuracy (Acc)
			0.3	0.96
			0.7	0.96
			0.8	0.97
			1.0	0.97
		N_estimator = 150 Learning_rate = 0.8	Algorithme	Accuracy (Acc)
			SAMME	0.97
			SAMME.R	0.92

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 150, Learning rate = 0.8, Algorithm = 'SAMME' avec Accuracy = 0.97.

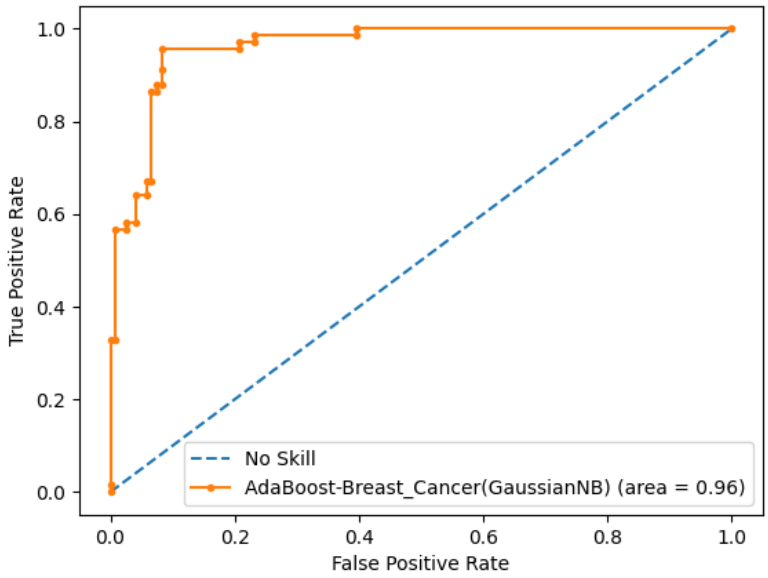


Figure 4.13 - Courbe ROC « AdaBoostClassifier » avec base estimator «GaussianNB» (sur le Dataset « Breast Cancer »).

4.4.5.4 Gradient Boosting

Maintenant, nous allons présenter les expérimentations faites avec l'algorithme Gradient Boosting pour les deux Dataset, on utilisant les paramètres suivant :

- N_estimator : de 50 jusqu'à 500 itération.
- Learning rate : entre 0.1 et 1.

Ci-dessous, les tableaux descriptifs des résultats obtenus.

Tableau 4.10 - Expérimentation Gradient Boosting sur le Dataset «Statlog»

	N_estimator	Accuracy (Acc)
Learning_rate=0.6, Max_depth=01	50	0.85
	100	0.80
	150	0.78
	200	0.78
N_estimators=50, Max_depth=01	Learning_rate	Accuracy (Acc)
	0.1	0.85
	0.2	0.85
	0.6	0.85
	1.0	0.85
N_estimators=50,Learning_rate=0.6	Max_depth	Accuracy (Acc)
	01	0.85
	02	0.81

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 50, Learning rate = 0.6, Max_depth=01 avec Accuracy = 0.85.

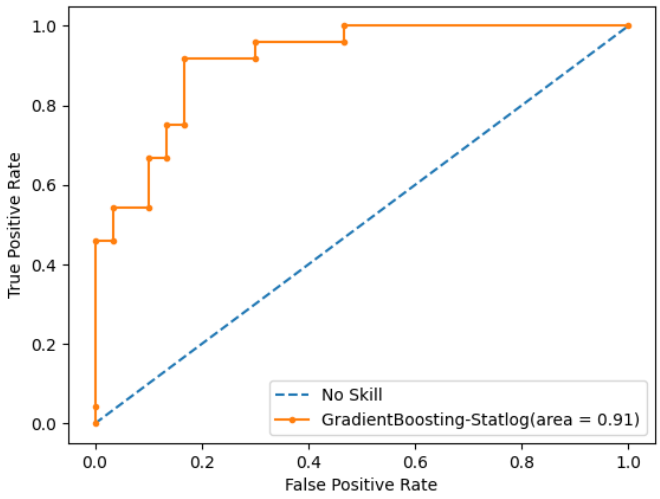


Figure 4.14 - Courbe ROC avec « GradientBoostingClassifier » sur le Dataset « Statlog ».

Tableau 4.11 Expérimentation Gradient Boosting sur le Dataset «Breast_Cancer»

	N_estimator	Accuracy (Acc)
Learning_rate=0.6, Max_depth=02	50	0.96
	100	0.95
	150	0.96
	200	0.97
N_estimators=50, Max_depth=01	Learning_rate	Accuracy (Acc)
	0.1	0.95
	0.2	0.96
	0.3	0.97
	0.4	0.96
N_estimators=50,Learning_rate=0.6	Max_depth	Accuracy (Acc)
	01	0.95
	02	0.97

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 200, Learning rate = 0.3, Max_depth=02 avec Accuracy = 0.97.

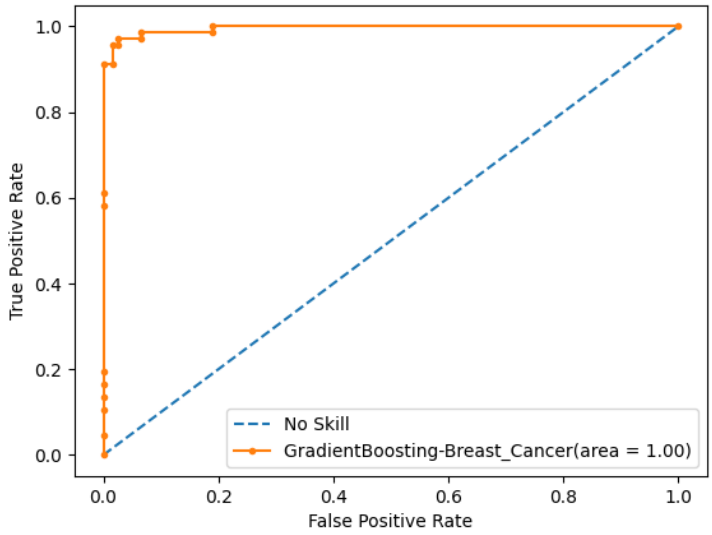


Figure 4.15 - Courbe ROC du « GradientBoostingClassifier » sur le Dataset «Breast_Cancer».

4.4.5.5 LogitBoost

Finalement, nous allons présenter les expérimentations faites avec l'algorithme LogitBoost pour les deux Dataset, on utilisant les paramètres suivant :

- N_estimator : de 50 jusqu'à 500 itération.
- Learning rate : entre 0.1 et 1.
- Boostrap : deux valeurs booléennes True et False.

Ci-dessous, les tableaux descriptifs des résultats obtenus.

Tableau 4.12 - Expérimentation LogitBoost sur le Dataset «Statlog»

	N_estimator	Accuracy (Acc)
Learning_rate=0.5, boostrap=True	50	0.91
	100	0.93
	150	0.94
	200	0.91
N_estimators=150, boostrap=True	Learning_rate	Accuracy (Acc)
	0.1	0.93
	0.2	0.89
	0.3	0.94
	0.5	0.94
N_estimators=150,Learning_rate=0.5	boostrap	Accuracy (Acc)
	True	0.93
	False	0.89

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 150, Learning rate = 0.5, boostrap=True avec Accuracy = 0.94.

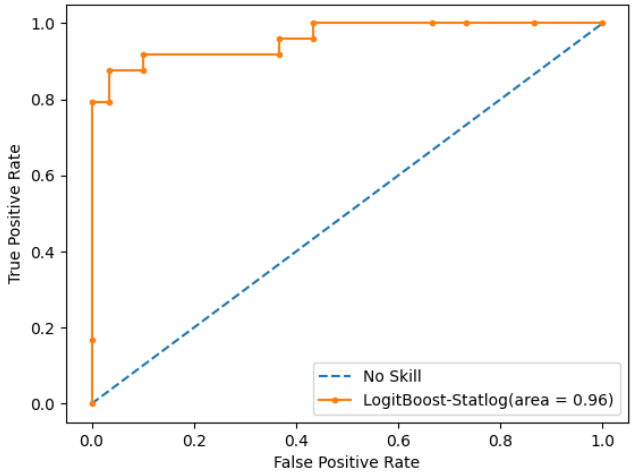


Figure 4.16 - Courbe ROC de « LogitBoostClassifier » sur le Dataset «Statlog».

Tableau 4.13 - Expérimentation LogitBoost sur le Dataset «Breast_Cancer»

	N_estimator	Accuracy (Acc)
Learning_rate=0.5, boostrap=True	50	0.97
	100	0.98
	150	0.98
	200	0.98
N_estimators=150, boostrap=True	Learning_rate	Accuracy (Acc)
	0.2	0.97
	0.5	0.98
	0.6	0.98
	0.9	0.98
N_estimators=150,Learning_rate=0.5	boostrap	Accuracy (Acc)
	True	0.98
	False	0.98

Les meilleurs résultats sont obtenus avec les paramètres suivant :

N_estimator = 100, Learning rate = 0.5, boostrap =True avec Accuracy = 0.98.

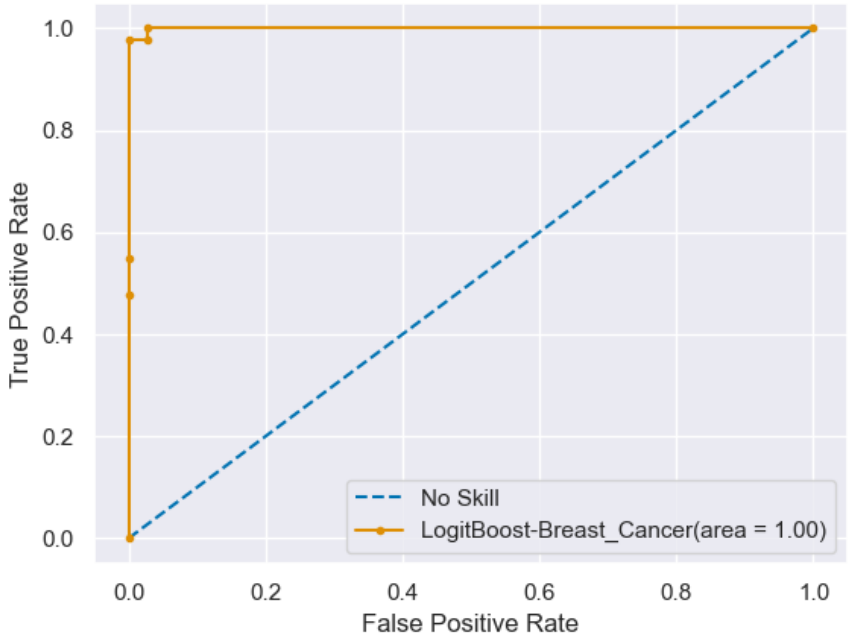


Figure 4.18 Courbe ROC de « LogitBoostClassifier » sur le Dataset «Breast_Cancer».

4.4.6 Interprétations

Nous abordons en premier lieu, la comparaison des performances réalisées par l'utilisation de l'algorithme AdaBoost basé sur les classifieurs SVM, DécisionTree et GaussianNB avec une Accuracy de 0.76, 0.90 et 0.88 respectivement pour Statlog Dataset et 0.98, 0.98 et 0.98 pour Breast Cancer. Ces trois algorithmes expérimentés individuellement ont donné une Accuracy de 0.71, 0.70 et 0.70 respectivement pour Statlog Dataset et 0.95, 0.91 et 0.93 pour Breast Cancer. On déduit d'après ses résultats que la combinaison de plusieurs classifieurs faibles améliore la précision.

En second lieu, une comparaison est faite entre les différents algorithmes d'ensemble (AdaBoost, GradientBoosting et LogitBoost) sur chaque Dataset.

D'après l'Accuracy obtenue, 0.90, 0.76, 0.88, 0.85, et 0.94 pour chaque algorithme AdaBoost/DécisionTree, AdaBoost/SVM, AdaBoost/GaussianNB, GradientBoosting et LogitBoost respectivement, sur Statlog Dataset, on peut remarquer que LogitBoost a été plus performant avec 94%. C'est une simple remarque car il ne s'agit que de quelques expérimentations sur un Dataset particulier. Et avec une Accuracy de 99%, obtenue avec

AdaBoost basé sur SVM sur le Dataset Breast Cancer, cet estimateur ensembliste a réalisé la meilleure performance.

Finalement, si nous comparons les résultats obtenus sur les deux Datasets, nous pouvons remarquer que les résultats obtenus sur Breast Cancer ont été meilleurs que ceux obtenus sur Statlog. Aucune déduction ne peut être émise à ce stade car il s'agit de deux Datasets complètement différents et il est bien connu que la qualité des données a une incidence certaine sur les performances d'un modèle entrainé sur ces données.

La seule déduction évidente que l'on peut émettre est que quel que soit l'algorithme de Boosting utilisé il est en général plus performant que les algorithmes individuels. Qui plus est, nous savons que le principe de Boosting permet d'éviter le biais dans le modèle généré. Les performances d'un algorithme de Boosting dépendent naturellement des données utilisées, du classifieur de base, et certainement de l'ajustement des hyperparamètres.

4.4.7 Sauvegarde Des Modèles

La sauvegarde du modèle finalisé en utilisant une fonction de sérialisation fournie par des modules tels que « joblib » permet de réutiliser un modèle bien paramétré sans avoir à le regénérer. Le gain en temps est considérable car il est connu que la génération d'un modèle basé sur le Boosting est très coûteuse en temps d'exécution car il s'agit d'un processus séquentiel qui comporte beaucoup d'itérations. La bibliothèque Joblib permet de sérialiser un modèle d'apprentissage automatique et l'enregistrer dans un un fichier binaire.

4.5 Conclusion

Dans ce chapitre, nous avons passé en revue l'expérimentation de trois algorithmes de Boosting et leur optimisation sur deux Datasets expérimentaux. Les résultats obtenus confirment les propriétés du Boosting.

Conclusion Générale

Le Machine Learning est une branche de l'intelligence artificielle englobant de nombreux algorithmes permettant de découvrir des patterns, d'effectuer des prédictions et de générer automatiquement des modèles à partir des données. Le sur-apprentissage, et le sous-apprentissage sont les causes principales des mauvaises performances de ces modèles prédictifs générés. En effet, un modèle prédictif dépend beaucoup plus des données que de la méthode de modélisation employée. Donc pour améliorer les performances il faut augmenter le nombre d'exemples de formation, ajouter d'autres variables et un meilleur traitement des entités et envisager d'autres valeurs pour les paramètres de formation utilisés par l'algorithme d'apprentissage.

Il existe deux grandes classes de méthodes ensemblistes, les méthodes d'apprentissage parallèles tels que le Bagging où les modèles sont indépendants et sont généralement entrainés sur des instances de données différentes. Le modèle final combine les prédictions de tous les modèles entrainés. Le Boosting, cependant, est un processus séquentiel dans lequel chaque nouveau modèle apprend des erreurs de ses précédents et améliore la prédiction. L'utilisation du Boosting dans des modèles de prédiction prend de plus en plus d'ampleur grâce aux résultats très précis des systèmes basés sur ce principe. Nous pouvons également citer le Stacking qui est plutôt une généralisation du Bagging.

Dans notre projet, nous avons d'abord orienté notre recherche documentaire pour mieux comprendre les fondements des méthodes ensemblistes de type « Boosting » qui est une approche largement utilisée pour résoudre les problèmes de classification et de régression. Sa force réside dans sa capacité à améliorer les performances de classificateurs individuels faibles pour en construire un puissant. Le succès obtenu par le premier algorithme de cette classe, AdaBoost, a donné lieu à la proposition de plusieurs algorithmes qui appliquent ce même principe mais avec des caractéristiques différentes tel que Gradient Boosting et LogitBoost.

Ce projet a été une occasion immense d'apprentissage tant sur le plan théorique que sur le plan pratique et technique. Nous avons pu découvrir toute la gamme des méthodes ensemblistes et leur rôle dans l'amélioration des performances des modèles de prédiction. Ensuite nous nous sommes concentrées sur le Boosting, et là aussi, nous avons découvert qu'il y a toute une gamme d'algorithmes qui appliquent ce principe mais avec des différences au niveau de quelques détails. Tous ces algorithmes ont vu le jour grâce au succès réalisé par le premier algorithme de Boosting : Adaboost.

Sur le plan pratique, nous avons pu découvrir un langage très performant tel que Python et les différents modules disponibles dont le rôle est de faciliter le travail du programmeur. En effet, avec des modules aussi bien faits, nous nous sommes plutôt concentrées sur la paramétrisation des algorithmes existants ce qui a permis d'obtenir rapidement des résultats.

Les résultats obtenus confirment les propriétés déjà annoncées des algorithmes de Boosting, à savoir que les performances sont meilleures lorsqu'on utilise du Boosting. De plus, le principe du Boosting permet de régler un des problèmes qu'on rencontre souvent avec les modèles de machine learning : l'undefitting. Les expérimentations faites n'ont pas pour objectif de comparer entre eux les trois algorithmes de Boosting traités dans le cadre de ce projet car une telle comparaison nécessiterait des expérimentations beaucoup plus larges.

Bibliographie

[1] Antoine Cornuéjols and Laurent Miclet. Apprentissage artificiel, concepts et algorithmes. Editions Eyrolles, August 2002.

[2] L. Breiman. Bagging Predictors. Mach Learn 24, 123-140 (1996), https://doi.org/10.1007/BF00058655

[3] L. Breiman, (1994). Heuristics of instability and stabilization in model selection. The Annals of Statistics, 24(6), 2350-2383, 1996.

[4] L.Breiman. (juin 1997). “Arcer le bord” (PDF) . Rapport technique 486 . Département de statistiques, Université de Californie, Berkeley.

[5] R. Clayton. Qu’est-ce que le machine learning ? (2019), Retrieved December 08, 2021, from Oracle Algeria: https://www.oracle.com/dz/artificial-intelligence/what-is-machine-learning.html

[6] Thomas G Dietterich,Ensemble Methods in Machine Learning Oregon State University Corvallis Oregon USA, tgdcs.orst.edu available at: https://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf

[7] D. Dua & Graff. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2019, http://archive.ics.uci.edu/ml

[8] B. Efron, & R. Tibshirani. An introduction to the bootstrap. Chapman & Hall, 1993.

[9] J. Friedman, T. Hastie and R. Tibshirani. Additive logistic regression: a statistical view of Boosting. Annals of Statistics, Volume 28, N° 2, pp. 337–407, 1998.

[10] JH.Friedman(février 1999). “Rapproximation de la fonction gourmande: une machine à booster de gradient” (PDF) .

[11] JH.Friedman(mars 1999). “Amélioration du gradient stochastique” (PDF) .

[12] A. Géron. Hands-On Machine_Learning_with_Scikit-learn, Keras and Tensorflow, 2nd edition. O’Reilly Media, Inc. 2019.

[13] I. Goodfellow, Y. Bengio and A. Courville. Deep Learning. MIT Press, 2016.

[14] Lubna A. Gabralla Conference Paper Prediction of Oil Prices Using Bagging and Random Subspace, June2014 : Advances in Intelligent Systems and Computing 303 DOI:10.1007/978-3-

319- 08156-4_34 ,Conference: IBICA2014 available at: https://www.researchgate.net/publication/267155389_Prediction_of_Oil_Prices_Using_Bagging_and_Random_Subspace

[15] Lubna A. Gabralla and A. Abraham. Prediction of Oil Prices Using Bagging and Random Subspace. Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014, Volume 303, Springer.

[16] https://oraprdnt.uqtr.uquebec.ca/pls/public/docs/FWG/GSC/Publication/1645/34/1918/1/328276/8/O0001163261_Salim_MOUDACHE_Depot_Final.pdf

[17] https://www.ottawaheart.ca/fr/maladie-du-c%C5%93ur/maladie-coronarienne-ath% C3%A9roscl%C3%A9rose.

[18] https://www.e-cancer.fr/Patients-et-proches/Les-cancers/Cancer-du-sein/Les-points-cles.

[19] https://archive.ics.uci.edu/ml/Datasets/Breast+Cancer+Wisconsin+%28Diagnostic% 29.

[20] http://e-biblio.univ-mosta.dz/bitstream/handle/123456789/20445/MINF273.pdf?sequence =1&isAllowed=y.

[21] https://www.google.com/search?q=naive+bayes+classifier&tbm=isch&hl=fr&sa=X&v e=2ahUKEwi2ktyByYX4AhXjRPEDHXvXB74QrNwCKAB6BQgBEOwB&biw=1349&bih=600#imgrc=ACitwWOUSPKHjM

[22] https://runebook.dev/fr/docs/scikit_learn/modules/generated / sklearn.ensemble.gradientBoostingclassifier

[23] https://logitBoost.readthedocs.io/logitBoost.html

[24] https://runebook.dev/fr/docs/scikit_learn/modules/generated/sklearn.ensemble.AdaBoostclassifier

[25] https://www.google.com/search?q=exemple+de+decision+tree+classification&sxsrf=ALiCzsabbPAeTZBLIlhKTKODhuzzdiGNoA:1652987785370&source=lnms&tbm=isch&sa=X&ved=2ahUKEwij_Y6uo-z3AhXRxIUKHXQuApcQ_AUoAXoECAEQAw&biw=1366&bih=600&dpr=1#imgrc=IoIiPlUF1b6BBM

[26] L.Mason; J.Baxter;PL. Bartlett; Frean, Marcus (1999). “Boosting Algorithms as Gradient Descent” (PDF) . Dans SA Solla et TK Leen et K. Müller (éd.). Progrès dans les systèmes de traitement de l’information neuronale 12 . Presse du MIT. p. 512–518.

[27] L.Mason; J.Baxter; PL.Bartlett; Frean, Marcus (mai 1999). “Boosting Algorithms as Gradient Descent in Function Space” (PDF) . Archivé de l’original (PDF) le 2018-12-22.

[28] A. Navalny. Adaboost Classifier in Python, available at: https://www.datacamp.com/tutorial/adaboost-classifier-python, accessed on: Mai 2022.

[29] H.-M. Suchier. Nouvelles Contributions du Boosting en Apprentissage Automatique. PhD thesis, université Jean Monnet de Saint-Etienne, France, 2006.

[30] S. Singh. Understanding the bias-variance tradeoff, available at: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229, accessed on: Mars 2022.

[31] A. Tripathy, A. Agrawal and S. Kumar Rath. Classification of Sentimental Reviews Using Machine Learning Techniques. 3rd International Conference on Recent Trends in Computing 2015, pp. 821-829, ISSN 1877-0509.

[33] P.Mehta,M. Bukov,C.Wang,A.G.R.Day,C. Richardson,C. K. Fisher,D. J. Schwab.// Rapports de physique,Volume 810 , 30 mai 2019 , Une introduction à l'apprentissage automatique à biais élevé et à faible variance .

[34] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel (...) Mathieu Blondel. «Scikit-learn: Machine Learning in python». Journal of Machine Learning Research 12, Octobre 2011.

[35] Scene classification using support vector machines with lda. N.Veeranjaneyulu, Akkineni Raghunath, B. Jyostna Devi , Venkata Naresh Mandhala. Journal of Theoretical and Applied Information Technology.2014.

[36] Understanding the Mathematics behind Support Vector Machines by Nikita Sharma | Heartbeat (comet.ml)

[37] D.H. Wolpert: Stacked generalization. Neural Networks, 5(2) :241-259 (1992).