



Département :

Génie Informatique

Optimisation des hyperparamètres d'un modèle d'apprentissage supervisé

Réalisé par :

Akrraye Chaymaa
Khattami Salwa
Mouftah Nouhaila

Encadré par :

Pr. Abdessamad KAMO USS

Id du groupe : G2

Id du projet : P169

Année Académique 2024/2025

Table des matières

Résumé	1
Abstract	2
Table des figures	3
Liste des tableaux	4
Introduction générale	5
1 Présentation du contexte du projet	6
1.1 Présentation de contexte	6
1.1.1 Processus de fonctionnement	6
1.1.1.1 Définition et principes de base	6
1.1.1.2 Importance des hyperparamètres dans les modèles	6
1.1.2 Enjeux de l'optimisation des hyperparamètres	7
1.1.2.1 Impact sur les performances des modèles	7
1.1.2.2 Défis associés à l'optimisation	7
1.1.3 Méthodes courantes d'optimisation	7
1.1.3.1 Méthodes Model-Free	7
1.1.4 Quelques chiffres clés	8
1.1.5 Intérêt de RO dans ce domaine	9
1.1.5.1 Définition de la Recherche Opérationnelle (RO)	9
1.1.5.2 L'essor du Machine Learning (ML) et l'intégration avec la RO	9
1.1.5.3 Le Paradigme : Prédire puis Optimiser	10
1.1.5.4 Optimisation pour Améliorer les Modèles ML	10
1.1.5.5 Les Algorithmes d'Optimisation au Service des Applications ML	10
1.1.5.6 Les Impacts de l'Intégration de la RO et du ML sur les Etudes et l'Industrie	10
1.2 Présentation de la problématique	11
1.2.1 Définition du problème à résoudre	11
1.2.1.1 Description du modèle d'apprentissage supervisé utilisé	11
1.2.1.2 Hyperparamètres ciblés pour l'optimisation	11
1.2.2 Objectifs du projet et résultats attendus	12
2 Modélisation	13
2.1 Choix de la modélisation appropriée	13
2.2 Variables de décisions	13
2.3 Contraintes	14
2.4 Fonction objectif	14
2.5 Description détaillée des méthodes	14
2.5.1 Processus Gaussien (Gaussian Process, GP)	14
2.5.2 Méthode d'optimisation L-BFGS-B	15
2.5.3 Avantages de l'approche hybride	16
3 Résolution	17
3.1 Outils utilisés	17
3.2 Résolution détaillée et résultats	18
3.2.1 Résolution détaillée	18
3.2.1.1 Utilisation d'un modèle surrogate (Gaussian Process Regressor)	18
3.2.1.2 Optimisation avec L-BFGS-B	18

3.2.1.3	Choix du nombre d'échantillons pour l'optimisation	19
3.2.2	Résultats obtenus	19
3.2.2.1	Interprétation des résultats obtenus	19
3.2.2.2	Comparaison avec des approches existantes	21
3.3	Interprétations et critiques	22
3.3.1	Analyse des performances du modèle optimisé	22
3.3.2	Discussion des avantages et limitations de l'approche adoptée	22

4	Conclusion Générale	23
----------	----------------------------	-----------

Résumé

Ce projet met l'accent sur l'optimisation des hyperparamètres dans les modèles d'apprentissage supervisé, un domaine clé pour accroître l'efficacité des systèmes d'intelligence artificielle. Les hyperparamètres, comme le taux d'apprentissage, la profondeur des arbres ou le nombre de neurones, sont essentiels pour la précision, la généralisation et la rapidité d'apprentissage des modèles. Cependant, leur optimisation à la main est compliquée et consomme beaucoup de temps, nécessitant des compétences poussées et des réglages précis.

Cette étude examine diverses techniques d'optimisation des hyperparamètres, incluant la recherche en grille (Grid Search), la recherche aléatoire (Random Search) et l'optimisation bayésienne. Chacune de ces méthodes a ses atouts et ses faiblesses, surtout quand il s'agit d'aborder d'importants domaines d'études ou des ensembles de paramètres liés. Les résultats issus de l'optimisation des hyperparamètres peuvent considérablement augmenter la précision des modèles, tout en diminuant les erreurs de prédiction, telles que l'erreur quadratique moyenne (MSE) et la racine carrée de l'erreur quadratique moyenne (RMSE). Ces optimisations peuvent être appliquées à plusieurs secteurs, notamment la santé, la finance et l'industrie, en permettant des modèles plus performants et plus efficaces.

Dans ce cadre, l'utilisation de la Recherche Opérationnelle (RO), particulièrement via des méthodes telles que la programmation linéaire et non linéaire, se révèle pertinente pour organiser et résoudre le défi d'optimisation des hyperparamètres. Grâce à un modèle d'apprentissage supervisé tel que XGBoost, ce projet a pour objectif de déterminer la meilleure combinaison d'hyperparamètres afin d'optimiser les performances du modèle. Les résultats escomptés comprennent une augmentation mesurable de la précision et de la solidité, en plus d'une diminution notable des erreurs.

Ce projet a pour objectif non seulement de comparer diverses techniques d'optimisation des hyperparamètres, mais aussi d'offrir une meilleure compréhension des effets de ces ajustements sur la performance globale des modèles d'apprentissage supervisé, aidant ainsi à développer des systèmes d'IA plus solides et plus efficaces.

Abstract

This project focuses on the optimization of hyperparameters in supervised learning models, a key area for increasing the efficiency of artificial intelligence systems. Hyperparameters, such as learning rate, tree depth or number of neurons, are essential for the accuracy, generalization and learning speed of models. However, optimizing them by hand is complicated and time-consuming, requiring advanced skills and fine-tuning.

This study examines various hyperparameter optimization techniques, including grid search, random search and Bayesian optimization. Each of these methods has its strengths and weaknesses, especially when dealing with large study domains or sets of related parameters. The results of hyperparameter optimization can significantly increase model accuracy, while reducing prediction errors such as mean square error (MSE) and root mean square error (RMSE). These optimizations can be applied to a number of sectors, including healthcare, finance and industry, resulting in better, more efficient models.

In this context, the use of Operations Research (OR), particularly via methods such as linear and nonlinear programming, is proving relevant to organizing and solving the hyperparameter optimization challenge. Using a supervised learning model such as XGBoost, this project aims to determine the best combination of hyperparameters to optimize model performance. Expected results include a measurable increase in accuracy and robustness, in addition to a significant reduction in errors.

The aim of this project is not only to compare various hyperparameter optimization techniques, but also to provide a better understanding of the effects of these adjustments on the overall performance of supervised learning models, thus helping to develop more robust and efficient AI systems.

Table des figures

3.1	Ajustement du modèle de substitution	18
3.2	Options de Kernel	18
3.3	Estimation initiale des hyperparamètres pour l'algorithme L-BFGS-B.	19
3.4	Ajustement du processus pour déterminer le nombre optimal d'échantillons requis (n=210) pour l'optimisation des hyperparamètres.	19
3.5	Évolution du MSE au fil des itérations	20
3.6	Distribution des hyperparamètres	20
3.7	Distribution des MSE	20
3.8	Corrélations entre les paramètres et le MSE	21

Liste des tableaux

- 1.1 Paramètres de XGBoost pour le réglage des arbres 12
- 3.1 Comparaison avec des approches existantes 21

Introduction générale

L'optimisation des hyperparamètres est un aspect essentiel dans l'amélioration des performances des modèles d'apprentissage automatique. Dans le cas de XGBoost, un modèle d'apprentissage supervisé largement utilisé pour des tâches de régression et de classification, cette optimisation devient un défi en raison de la nature complexe de ses hyperparamètres et de l'absence d'une relation explicite entre ces derniers et la fonction objectif, souvent mesurée par l'erreur quadratique moyenne (MSE). L'optimisation des hyperparamètres de XGBoost est donc classifiée comme un problème d'optimisation de type boîte noire.

Pour aborder ce défi, une approche hybride a été adoptée, combinant un modèle de substitution basé sur un Processus Gaussien (GPR) et une méthode d'optimisation non linéaire avec contraintes telle que la méthode L-BFGS-B. Le modèle de substitution, grâce au Processus Gaussien, permet de réduire le coût des évaluations de la MSE en approximant la relation entre les hyperparamètres et l'erreur, tout en guidant l'optimisation vers des solutions prometteuses. La méthode L-BFGS-B, quant à elle, est une technique d'optimisation itérative efficace qui permet de minimiser la fonction objectif en respectant les contraintes sur les hyperparamètres.

Ce processus d'optimisation hybride constitue une solution robuste pour maximiser les performances de XGBoost tout en tenant compte des contraintes imposées aux hyperparamètres du modèle. Cette approche permet ainsi de trouver les configurations d'hyperparamètres les plus adaptées tout en garantissant une convergence rapide et efficace vers le minimum de l'erreur quadratique moyenne.

Chapitre 1

Présentation du contexte du projet

1.1 Présentation de contexte

1.1.1 Processus de fonctionnement

1.1.1.1 Définition et principes de base

Les modèles d'apprentissage automatique (ML) peuvent être généralement classés en deux grandes catégories : les algorithmes d'apprentissage supervisé et non supervisé. Cette distinction dépend de la disponibilité ou non de données étiquetées pour l'entraînement.

Les algorithmes d'apprentissage supervisé consistent à associer des caractéristiques d'entrée à une cible en s'appuyant sur des données étiquetées. Parmi ces algorithmes, on retrouve les modèles linéaires, les k-plus proches voisins (k-NN), les machines à vecteurs de support (SVM), les algorithmes de Bayes naïf (NB), les modèles basés sur les arbres de décision, ainsi que les algorithmes d'apprentissage profond (DL).

Dans l'apprentissage supervisé, les données d'entrée x et les résultats y sont disponibles. L'objectif est de construire une fonction prédictive optimale f^* qui minimise une fonction de coût $\mathcal{L}(f(x), y)$, qui mesure l'erreur entre les prédictions et les résultats réels. La structure de cette fonction prédictive f varie selon le modèle choisi. De plus, avec des architectures de modèles limitées par différentes configurations d'hyperparamètres, le domaine de la fonction f est restreint à un ensemble de fonctions F . Ainsi, le modèle prédictif optimal f^* peut être obtenu via :

$$f^* = \arg \min_{f \in F} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x_i), y_i)$$

où n représente le nombre d'exemples dans l'ensemble d'entraînement, x_i est le vecteur des caractéristiques de l'exemple i , y_i est la sortie réelle correspondante, et \mathcal{L} est la valeur de la fonction de coût pour chaque échantillon.

Différents algorithmes d'apprentissage supervisé utilisent des fonctions de perte spécifiques, telles que la distance euclidienne au carré, l'entropie croisée ou le gain d'information. En parallèle, chaque algorithme génère des architectures de modèles prédictifs différentes, influencées par les configurations des hyperparamètres, qui seront examinées en détail dans cette sous-section.

1.1.1.2 Importance des hyperparamètres dans les modèles

Les algorithmes d'apprentissage automatique (ML) sont hautement configurables grâce à leurs hyperparamètres.

Les hyperparamètres sont des variables de configuration qui contrôlent le comportement des algorithmes d'apprentissage automatique. Ils jouent un rôle crucial dans la détermination de l'efficacité des systèmes basés sur ces technologies. La recherche manuelle des hyperparamètres est souvent insatisfaisante et devient impraticable lorsque leur nombre est élevé. Automatiser cette recherche est une étape importante vers l'automatisation de l'apprentissage automatique, libérant ainsi les chercheurs et les praticiens du fardeau de trouver un bon ensemble d'hyperparamètres par essais et erreurs.

Les hyperparamètres influencent plusieurs aspects des modèles :

- **La précision du modèle** : une mauvaise configuration peut entraîner un sous-apprentissage ou un sur-apprentissage.
- **La vitesse de convergence** : certains hyperparamètres, comme le taux d'apprentissage, déterminent à quelle vitesse un modèle atteint des performances optimales.
- **La capacité de généralisation** : il est essentiel de trouver un équilibre entre une bonne performance sur les

données d'entraînement et une robustesse sur des données non vues.

La sélection manuelle des hyperparamètres, souvent basée sur des essais et des erreurs, peut être longue, biaisée, sujette à des erreurs et difficilement reproductible d'un point de vue computationnel. Cela souligne la nécessité d'une optimisation efficace et systématique des hyperparamètres, un défi central dans ce domaine.

1.1.2 Enjeux de l'optimisation des hyperparamètres

L'optimisation des hyperparamètres est une étape cruciale dans le développement de modèles d'apprentissage automatique performants. Cette section examine les enjeux associés à cette optimisation, en mettant l'accent sur son impact sur les performances des modèles et les défis inhérents à sa mise en œuvre.

1.1.2.1 Impact sur les performances des modèles

Une optimisation efficace des hyperparamètres peut considérablement améliorer la précision et la robustesse des modèles d'apprentissage automatique. En ajustant correctement ces paramètres, on peut réduire les erreurs de prédiction, telles que l'erreur quadratique moyenne (MSE) ou l'erreur quadratique moyenne racine (RMSE). Par exemple, dans des domaines tels que la santé, la finance et l'industrie, des modèles optimisés offrent des prédictions plus fiables, ce qui conduit à de meilleures prises de décision et à des résultats opérationnels améliorés.

1.1.2.2 Défis associés à l'optimisation

L'optimisation des hyperparamètres présente plusieurs défis notables.

- **Complexité croissante avec la taille des espaces de recherche** : À mesure que le nombre d'hyperparamètres augmente, l'espace de recherche devient exponentiellement plus vaste, rendant l'exploration exhaustive impraticable.

- **Temps de calcul et ressources nécessaires pour les évaluations** : Chaque évaluation d'une configuration d'hyperparamètres requiert des ressources computationnelles importantes et peut être chronophage, surtout pour des modèles complexes.

- **Interdépendance complexe des hyperparamètres** : Les hyperparamètres sont souvent interdépendants de manière complexe, ce qui complique l'identification de la combinaison optimale.

Ces défis soulignent la nécessité de méthodes d'optimisation avancées pour naviguer efficacement dans l'espace des hyperparamètres et identifier les configurations les plus performantes.

1.1.3 Méthodes courantes d'optimisation

Plusieurs méthodes sont couramment utilisées pour l'optimisation des hyperparamètres, chacune ayant ses avantages et ses inconvénients. Ces méthodes peuvent être divisées en deux catégories principales : les méthodes *model-free* et les méthodes *model-based*.

1.1.3.1 Méthodes Model-Free

Les méthodes *model-free* (sans modèle) ne nécessitent pas de modèle explicite du processus d'optimisation, mais reposent sur des évaluations directes du modèle pour déterminer les meilleures configurations d'hyperparamètres. Ces techniques incluent des méthodes comme le "*babysitting*", la *recherche en grille* (*Grid Search*) et la *recherche aléatoire* (*Random Search*), qui, bien qu'efficaces pour des espaces de recherche réduits, présentent des limitations lorsque le nombre d'hyperparamètres ou la complexité des interactions entre ces derniers augmente.

— Babysitting

Le babysitting, également appelé *Trial and Error* ou *Grad Student Descent* (*descente de l'étudiant*), est une méthode d'optimisation des hyperparamètres très basique et largement utilisée, surtout dans le milieu académique. Cette méthode repose entièrement sur l'ajustement manuel des hyperparamètres : un étudiant teste différentes valeurs d'hyperparamètres en fonction de son expérience, de ses suppositions ou des résultats d'évaluations précédentes. Le processus est répété jusqu'à ce que l'étudiant manque de temps ou soit satisfait des résultats. Bien que cette méthode soit simple, elle nécessite un savoir-faire substantiel pour identifier des valeurs optimales avec un temps limité. Cependant, elle devient inefficace pour des problèmes plus complexes avec de nombreux hyperparamètres et des interactions non linéaires entre eux.

— Recherche par grille (Grid Search)

La recherche en grille est l'une des méthodes les plus courantes pour explorer l'espace de configuration des hyperparamètres. Elle peut être vue comme une recherche exhaustive ou une méthode de "force brute", où toutes les combinaisons d'hyperparamètres spécifiées dans la grille sont évaluées. La méthode consiste à évaluer le produit cartésien d'un ensemble de valeurs définies par l'utilisateur. Cependant, la recherche en grille a des limites car elle ne peut pas exploiter les régions bien performantes. Pour trouver l'optimum global, un processus manuel est souvent nécessaire, consistant à commencer avec un large espace de recherche et une grande taille de pas, puis à réduire cet espace et la taille du pas en fonction des résultats obtenus. L'inconvénient majeur de cette méthode est son inefficacité dans les espaces de recherche de grande dimension, car le nombre d'évaluations croît de manière exponentielle. Sa complexité computationnelle est de l'ordre de $O(n^k)$, où n est le nombre de valeurs distinctes pour chaque hyperparamètre et k le nombre d'hyperparamètres.

— Recherche aléatoire (Random Search)

La recherche aléatoire a été proposée pour surmonter certaines limites de la recherche en grille. Contrairement à la recherche en grille, qui teste toutes les valeurs dans l'espace de recherche, la recherche aléatoire sélectionne un nombre pré-défini d'échantillons à partir des limites des hyperparamètres et entraîne ces candidats jusqu'à ce que le budget soit épuisé. L'avantage de la recherche aléatoire est qu'elle permet d'explorer un espace de recherche plus vaste de manière plus efficace. Sa complexité est de $O(n)$, où n est le nombre total d'évaluations. Cependant, la recherche aléatoire souffre également de limitations, car elle ne tire pas parti des régions bien performantes, entraînant des évaluations inutiles. En raison de cette indépendance des évaluations, les méthodes comme l'optimisation bayésienne, qui utilisent les résultats précédents pour guider les évaluations suivantes, peuvent résoudre ce problème.

— Optimisation Basée sur le Gradient

L'optimisation basée sur le gradient est une technique traditionnelle qui calcule le gradient des variables pour identifier la direction prometteuse et se déplacer vers l'optimum. Après avoir sélectionné un point de données, l'algorithme se déplace dans la direction opposée au plus grand gradient. Ce processus continue jusqu'à ce qu'un optimum local soit atteint, qui est également l'optimum global pour les fonctions convexes. Cependant, cette méthode n'est efficace que pour optimiser les hyperparamètres continus et est limitée par la présence de minima locaux dans les fonctions non convexes. L'optimisation par gradient a une complexité de $O(n^k)$ pour k hyperparamètres.

— Optimisation bayésienne

L'optimisation bayésienne (Bayesian Optimization) est une technique probabiliste itérative qui se distingue des méthodes comme la recherche en grille et la recherche aléatoire en utilisant les résultats des évaluations précédentes pour guider les choix futurs des hyperparamètres. Elle repose sur deux composants principaux : un modèle de substitution et une fonction d'acquisition. Le modèle de substitution ajuste les points observés dans la fonction objective, tandis que la fonction d'acquisition détermine les points à évaluer en équilibrant exploration (échantillonnage des zones inexplorées) et exploitation (échantillonnage des régions prometteuses). Des exemples d'algorithmes d'optimisation bayésienne incluent les processus gaussiens et les estimateurs de Parzen structurés en arbre (TPE). Cette approche est particulièrement efficace pour des espaces de configurations complexes et de grande dimension, car elle réduit le nombre d'évaluations nécessaires pour identifier des configurations performantes.

1.1.4 Quelques chiffres clés

— Croissance du marché de l'intelligence artificielle (IA) :

Le marché de l'intelligence artificielle connaît une expansion remarquable. En 2023, sa valeur globale était estimée à environ **196,63 milliards de dollars**. Avec un taux de croissance annuel composé (CAGR) projeté à 36,6% entre 2024 et 2030, la valeur du marché devrait atteindre environ **1 811,75 milliards de dollars** d'ici 2030. Cette croissance rapide témoigne de l'intégration croissante de l'IA dans divers secteurs tels que la santé, la finance et l'industrie.

— Impact de l’optimisation des hyperparamètres sur les performances des modèles :

L’optimisation des hyperparamètres joue un rôle crucial dans l’amélioration des performances des modèles d’apprentissage automatique. Une étude intitulée **Improving Classification Accuracy of Fine-Tuned CNN Models** a démontré que l’optimisation des hyperparamètres peut augmenter la précision de classification des réseaux neuronaux convolutifs (CNN) de **6%**. Ce résultat illustre l’importance de cette étape pour maximiser la performance des systèmes d’apprentissage automatique, notamment dans des applications critiques où l’amélioration de la précision peut avoir un impact significatif.

1.1.5 Intérêt de RO dans ce domaine

1.1.5.1 Définition de la Recherche Opérationnelle (RO)

La Recherche Opérationnelle est une discipline à la frontière des mathématiques appliquées et de l’informatique. L’objectif de la Recherche Opérationnelle est de trouver des solutions (recherche) à des (vrais) problèmes (opérationnelle). Elle utilise des méthodes de résolution (algorithmes) permettant de construire des solutions sur des problèmes bien formulés (modèles), elle s’appuie sur une variété de méthodes analytiques avancées, notamment la modélisation mathématique, les statistiques et l’optimisation, pour élaborer des modèles qui décrivent et analysent des systèmes complexes. Ces modèles permettent d’identifier des solutions optimales ou quasi-optimales à des problèmes décisionnels, en tenant compte des contraintes et des objectifs spécifiques à chaque situation. Les étapes typiques d’une étude en Recherche Opérationnelle comprennent :

1. **Formulation du problème** : Définir clairement le problème à résoudre, les objectifs à atteindre, les variables de décision et les contraintes à respecter.
2. **Construction du modèle** : Élaborer un modèle mathématique ou informatique qui représente fidèlement le système étudié, en intégrant les relations entre les variables et les contraintes identifiées.
3. **Résolution** : Appliquer des méthodes et des algorithmes appropriés pour trouver des solutions optimales ou satisfaisantes au modèle formulé.
4. **Validation et analyse** : Vérifier la pertinence et la robustesse des solutions obtenues, en les confrontant aux données réelles et en évaluant leur sensibilité aux variations des paramètres du modèle.
5. **Mise en œuvre** : Proposer des recommandations concrètes pour la prise de décision, en tenant compte des résultats de l’analyse et des contraintes opérationnelles du contexte étudié.

La Recherche Opérationnelle trouve des applications dans de nombreux domaines, tels que la logistique, la production industrielle, la gestion des stocks, la planification des transports, la finance, la santé et les services publics. Par exemple, elle peut être utilisée pour optimiser les itinéraires de livraison, planifier la production dans une usine, gérer les files d’attente dans les services d’urgence ou encore élaborer des stratégies d’investissement financier.

1.1.5.2 L’essor du Machine Learning (ML) et l’intégration avec la RO

Au cours des dernières années, l’Apprentissage Automatique (ML), un sous-domaine de l’Intelligence Artificielle, a connu une croissance explosive. Le ML se concentre sur la création de modèles capables d’apprendre à partir de données et de faire des prédictions ou des classifications sans intervention humaine explicite. Cette évolution a transformé divers secteurs, de la santé aux systèmes financiers, en passant par la robotique et la gestion des opérations. En 2021, les investissements mondiaux dans l’IA ont dépassé les 90 milliards de dollars, soulignant l’importance croissante de ces technologies dans l’industrie.

Cependant, l’application des techniques de ML a parfois besoin de structures mathématiques solides pour garantir des décisions efficaces et optimales. C’est là que la Recherche Opérationnelle intervient. L’intégration de la RO et du ML ouvre la voie à des solutions plus robustes et performantes, combinant la puissance des algorithmes de prédiction du ML avec les capacités d’optimisation de la RO.

1.1.5.3 Le Paradigme : Prédire puis Optimiser

L'une des applications les plus prometteuses de la synergie entre RO et ML réside dans le paradigme "Prédire puis Optimiser".

Cette méthode se décompose en deux étapes principales :

Prédiction : Utilisation de modèles d'apprentissage automatique pour estimer les paramètres inconnus en se basant sur des données historiques ou actuelles.

Optimisation : Application de techniques d'optimisation, telles que celles issues de la recherche opérationnelle, pour déterminer la meilleure décision ou action possible en utilisant les paramètres estimés lors de la phase de prédiction.

Cela permet d'intégrer les modèles de prédiction dans le cadre plus large de l'optimisation des décisions

1.1.5.4 Optimisation pour Améliorer les Modèles ML

Inversement, l'optimisation peut également être utilisée pour améliorer les algorithmes de ML. Par exemple, des méthodes telles que **les forêts aléatoires** peuvent être rendues plus interprétables grâce à des techniques d'optimisation par exemple les méthodes d'élagage optimisés réduisant le nombre d'arbres ou la profondeur des arbres individuels, facilitant ainsi la compréhension du modèle sans compromettre significativement sa performance. De même, **les machines à vecteurs de support (SVM)** qui sont des algorithmes puissants pour la classification, mais leur performance peut être limitée lorsqu'ils traitent de grandes quantités de données ou des espaces de caractéristiques étendus. La génération de colonnes, une technique issue de la recherche opérationnelle, peut être utilisée pour améliorer l'efficacité des SVM. Cette méthode consiste à résoudre le problème d'optimisation en considérant uniquement un sous-ensemble des variables (ou colonnes) et en ajoutant progressivement de nouvelles variables pertinentes. Cela permet de gérer efficacement des ensembles de données de grande dimension et d'améliorer la précision du modèle. L'optimisation dans ce contexte permet de raffiner les modèles de ML et de les rendre plus efficaces dans la gestion de grandes quantités de données.

1.1.5.5 Les Algorithmes d'Optimisation au Service des Applications ML

L'intégration de l'optimisation et de l'apprentissage automatique (ML) est particulièrement bénéfique dans des domaines tels que le routage de véhicules et la gestion des chaînes d'approvisionnement. Dans ces contextes, les problèmes combinatoires complexes nécessitent des solutions efficaces pour améliorer la performance opérationnelle. Par exemple :

— Routage de Véhicules :

Les problèmes de routage de véhicules (VRP) impliquent la détermination des itinéraires optimaux pour une flotte de véhicules afin de desservir un ensemble de clients avec des contraintes spécifiques. L'apprentissage automatique peut être utilisé pour anticiper les demandes ou les conditions de trafic, tandis que les techniques d'optimisation, telles que les algorithmes de branchement et de bornes, résolvent le problème combinatoire sous-jacent. Des approches récentes intègrent l'apprentissage statistique dans les algorithmes de branchement et de bornes pour améliorer l'efficacité du processus de décision.

— Gestion des Chaînes d'Approvisionnement :

Dans la gestion des chaînes d'approvisionnement, l'optimisation est essentielle pour la planification des stocks, la sélection des fournisseurs et la distribution des produits. L'apprentissage automatique peut analyser des données historiques pour prévoir la demande, tandis que les modèles d'optimisation déterminent les stratégies les plus efficaces pour répondre à cette demande. Par exemple, l'IA est utilisée pour optimiser les itinéraires et la planification des chargements dans le domaine du transport et de la logistique, en analysant les conditions météorologiques en temps réel, les données de trafic et les modèles historiques pour trouver les itinéraires de livraison les plus efficaces.

1.1.5.6 Les Impacts de l'Intégration de la RO et du ML sur les Etudes et l'Industrie

L'intégration de la recherche opérationnelle (RO) et de l'apprentissage automatique (ML) offre des opportunités significatives pour optimiser la gestion des ressources dans divers secteurs. Cette synergie permet de combiner la capacité prédictive du ML avec les techniques d'optimisation de la RO, aboutissant à des solutions plus efficaces et adaptatives.

— Optimisation de l'Allocation des Ressources :

Dans des domaines tels que les centres de données et les réseaux de communication, l'apprentissage automatique peut analyser des données historiques pour anticiper les besoins en ressources. Ces prévisions, intégrées aux modèles d'optimisation de la RO, permettent une allocation dynamique et efficace des ressources, améliorant ainsi l'efficacité opérationnelle et réduisant les coûts.

— Applications Industrielles :

Des entreprises comme Amazon illustrent cette intégration en utilisant des algorithmes de ML pour optimiser leur chaîne d'approvisionnement. Cette approche a permis de réduire les délais de livraison de 20% et les coûts de stockage, démontrant l'impact tangible de la combinaison de la RO et du ML dans des environnements industriels complexes.

— Défis et Perspectives :

Bien que prometteuse, l'intégration de la RO et du ML pose des défis, notamment en termes de modélisation et de complexité computationnelle. Cependant, des initiatives académiques et industrielles s'efforcent de développer des méthodologies hybrides pour exploiter pleinement le potentiel de cette synergie, ouvrant la voie à des systèmes de gestion des ressources plus intelligents et réactifs.

1.2 Présentation de la problématique

1.2.1 Définition du problème à résoudre

1.2.1.1 Description du modèle d'apprentissage supervisé utilisé

XGBoost ou **Extreme Gradient Boosting**, un modèle d'apprentissage automatique conçu pour améliorer la vitesse et les performances, qui repose sur les arbres de décision. C'est une méthode de classification courante qui utilise une structure en arbre pour modéliser les décisions et leurs conséquences possibles en résumant un ensemble de règles de classification à partir des données. Un arbre de décision se compose de trois éléments principaux :

un nœud racine représentant l'ensemble des données, plusieurs **nœuds de décision** indiquant des tests décisionnels et des subdivisions des nœuds selon chaque caractéristique, et plusieurs **feuilles** représentant les classes de résultats.

Les nouveaux arbres utilisent des échantillons en fonction des résultats des arbres précédents. Les algorithmes de DT (*Decision Trees*) divisent récursivement l'ensemble d'entraînement en utilisant les valeurs des caractéristiques pour prendre de meilleures décisions sur chaque sous-ensemble. **XGBoost** est efficace avec les jeux de données volumineux et complexes en faisant appel à diverses méthodes d'optimisation. Le choix des bons paramètres et la détermination des valeurs idéales pour ces derniers sont cruciaux pour obtenir un résultat optimal. Ce processus devient complexe lorsqu'il s'agit de décider sur quels paramètres se concentrer et quelles valeurs leur attribuer, ce qui rend essentiel l'obtention de réponses pratiques pour garantir les meilleures performances possibles du modèle. L'objectif de XGBoost est de minimiser une fonction objective donnée :

$$\text{Obj} = -\frac{1}{2} \sum_{j=1}^t \frac{G_j^2}{H_j + \lambda} + \gamma t$$

où :

- t représente le nombre de feuilles dans un arbre de décision
- G et H sont les sommes des gradients de premier et de second ordre de la fonction de coût
- γ et λ sont des coefficients de pénalité.

1.2.1.2 Hyperparamètres ciblés pour l'optimisation

Avant de lancer XGBoost, nous devons configurer trois types de paramètres :

1. **Les paramètres généraux** : Ces paramètres définissent le fonctionnement global de XGBoost.

- **booster** [par défaut=gbtree] : Sélectionne le type de modèle à utiliser à chaque itération. Il existe deux options :
 - **gbtree** et **dart** : modèles basés sur des arbres
 - **gblinear** : modèles linéaires

- **silent** [par défaut=0] : Si la valeur est définie sur 1, les messages de fonctionnement ne seront pas imprimés.
- **nthread** [par défaut=le nombre maximum de threads disponibles si non défini] : Utilisé pour le traitement parallèle. Le nombre de cœurs du système doit être spécifié.

2. **Les paramètres de booster** : Nous nous concentrerons uniquement sur le booster basé sur des arbres, car il surpasse généralement le booster linéaire.

Paramètre	Description	Valeurs typiques
min_child_weight	Définir la somme minimale des poids des observations requise dans un fils.	[1 :5]
max_depth	Profondeur maximale de l'arbre. Utilisé pour contrôler le surapprentissage.	[3 :10]
gamma	Spécifie la réduction minimale de la perte requise pour effectuer une division.	Ajusté en fonction de la fonction de perte
subsample	Fraction d'observations échantillonnées aléatoirement pour chaque arbre.	[0.6 :1]
eta learning_rate	C'est la taille du pas utilisé pour réduire les poids des features lors de l'optimisation et pour prévenir le surapprentissage. Après chaque étape de renforcement, les poids des nouvelles features sont ajustés, ce qui rend le processus de boosting plus conservatif.	[0.01 :0.3]

TABLE 1.1 – Paramètres de XGBoost pour le réglage des arbres

1.2.2 Objectifs du projet et résultats attendus

- Identifier et ajuster les hyperparamètres du modèle XGBoost pour minimiser l'erreur de prédiction et améliorer la performance globale.
- Obtenir un modèle plus performant avec des prédictions plus précises et moins biaisées grâce à l'optimisation des hyperparamètres.
- Amélioration des performances du modèle.
- Réduire les erreurs de prédiction en optimisant les hyperparamètres du modèle et en contrôlant le surajustement.
- Une erreur quadratique moyenne (MSE) réduite, confirmant que le modèle prédit.

Chapitre 2

Modélisation

2.1 Choix de la modélisation appropriée

L'optimisation des hyperparamètres dans les modèles d'apprentissage supervisé, tels que **XGBoost**, constitue un problème classique d'optimisation de type boîte noire. Dans ce contexte, la relation entre les hyperparamètres et la fonction objectif à savoir l'erreur quadratique moyenne (Mean Squared Error, MSE) n'est pas explicitement définie. Cette absence de relation directe complique l'application des méthodes classiques d'optimisation

Pour relever ce défi, une approche hybride a été adoptée, combinant :

- Un **modèle de substitution** basé sur un **Processus Gaussien** (Gaussian Process Regression, GPR),
- Une **méthode d'optimisation non linéaire avec contraintes**, en l'occurrence la méthode **L-BFGS-B** (Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Box constraints).

Une fois le modèle de substitution établi, l'optimisation des hyperparamètres est réalisée à l'aide de la méthode L-BFGS-B. Cette méthode est particulièrement adaptée pour les problèmes d'optimisation non linéaire impliquant des variables continues ou discrètes, avec des contraintes de bornes.

2.2 Variables de décisions

les variables de décision représentent les hyperparamètres à ajuster pour optimiser la fonction objectif (minimiser l'erreur) :

$x_1 = \text{n_estimators}$:

- Nombre total d'arbres générés par le modèle.
- Variable discrète.

$x_2 = \text{max_depth}$:

- Définit la profondeur maximale des arbres.
- Variable discrète.

$x_3 = \text{min_child_weight}$:

- Spécifie le poids minimum requis pour une feuille d'un arbre. Si la somme des poids des observations dans une feuille est inférieure à cette valeur, l'arbre n'ajoutera pas de nouvelles divisions.
- Variable discrète.

$x_4 = \text{learning_rate}$:

- La vitesse à laquelle un algorithme met à jour ses estimations.
- Variable continue.

$x_5 = \text{subsample}$:

- Proportion des échantillons utilisés pour construire chaque arbre, pour réduire le surajustement.

— Variable continue.

2.3 Contraintes

les contraintes imposées à chaque x_i reflètent les plages valides des hyperparamètres :

$$\begin{aligned} 50 &\leq x_1 \leq 300 \\ 3 &\leq x_2 \leq 10 \\ 1 &\leq x_3 \leq 5 \\ 0.01 &\leq x_4 \leq 0.30 \\ 0.6 &\leq x_5 \leq 1.00 \\ x_1, x_2, x_3, x_4, x_5 &> 0 \end{aligned}$$

2.4 Fonction objectif

Un **modèle de substitution** a été construit pour approximer la fonction objectif réelle en fonction des hyperparamètres à optimiser. Le modèle choisi est un Processus Gaussien, qui offre une représentation probabiliste robuste de la relation entre les hyperparamètres et l'erreur du modèle.

L'objectif est de modéliser la fonction suivante :

$$Z = f(x_1, x_2, x_3, x_4, x_5) \quad (2.1)$$

où f est le modèle GPR, et x_1, x_2, x_3, x_4, x_5 sont les hyperparamètres du modèle XGBoost :

- x_1 : `n_estimators` (beta)
- x_2 : `max_depth`
- x_3 : `min_child_weight`
- x_4 : `learning_rate`(eta)
- x_5 : `subsample`

Ce modèle permet de réduire le nombre d'évaluations coûteuses nécessaires pour calculer directement l'erreur quadratique moyenne.

2.5 Description détaillée des méthodes

2.5.1 Processus Gaussien (Gaussian Process, GP)

Un Processus Gaussien est un modèle de substitution très utilisée pour modéliser des fonctions complexes. Il est particulièrement efficace pour les problèmes d'optimisation de *boîte noire*, où la relation exacte entre les variables d'entrée et la sortie n'est pas explicitement définie. (les hyperparamètres et l'erreur du modèle dans notre contexte)

1. Définition :

Un Processus Gaussien est défini comme un ensemble de variables aléatoires dans lequel tout sous-ensemble fini suit une distribution gaussienne. Pour un ensemble de données d'entrée $X = \{x_1, x_2, \dots, x_n\}$ et une sortie associée $Y = \{y_1, y_2, \dots, y_n\}$, on modélise la relation $y = f(x)$ par une distribution gaussienne conditionnelle :

$$f(x) \sim GP(m(x), k(x, x')) \quad (2.2)$$

où :

- $m(x)$ est la fonction moyenne, souvent définie comme nulle ($m(x) = 0$) pour simplifier.
- $k(x, x')$ est la fonction noyau (kernel) qui capture la similarité entre deux points x et x' .

2. Fonction noyau (Kernel Function)

La fonction noyau $k(x, x')$ joue un rôle central dans le GP en définissant la covariance entre deux points x et x' . Quelques noyaux courants utilisés sont :

- **Noyau exponentiel quadratique (RBF) :**

$$k(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \quad (2.3)$$

où σ est l'amplitude et ℓ est la longueur d'échelle (scale length). Ce noyau est utilisé dans notre projet pour capturer des relations lisses entre les hyperparamètres et la MSE.

- **Noyau Matern** :

$$k(x, x') = \sigma^2 \left(1 + \sqrt{3} \frac{\|x - x'\|}{\ell} \right) \exp \left(-\sqrt{3} \frac{\|x - x'\|}{\ell} \right) \quad (2.4)$$

où ν contrôle la rugosité du processus, ℓ est la longueur d'échelle, et K_ν est une fonction de Bessel.

3. Prédiction avec GP

Une fois le modèle GP entraîné avec les données observées (X, Y) , les prédictions pour un nouvel point x^* sont données par :

$$f(x^*) \sim \mathcal{N}(\mu(x^*), \sigma^2(x^*)) \quad (2.5)$$

où :

- $\mu(x^*)$ est la moyenne prédite.
- $\sigma^2(x^*)$ est l'incertitude associée à la prédiction.

Le GP est particulièrement utile pour guider l'optimisation en choisissant les points les plus prometteurs à évaluer, en tenant compte à la fois de l'incertitude et de la valeur attendue.

2.5.2 Méthode d'optimisation L-BFGS-B

La méthode L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Box constraints) est une extension de l'algorithme classique BFGS (Broyden-Fletcher-Goldfarb-Shanno), qui est une méthode de quasi-Newton pour l'optimisation non linéaire.

1. Principe de BFGS :

BFGS est une méthode d'optimisation itérative utilisée pour minimiser une fonction différentiable $f(x)$. Contrairement aux méthodes de descente de gradient, elle utilise une approximation de la matrice Hessienne (les dérivées secondes de $f(x)$) pour améliorer la vitesse de convergence.

L'idée principale de l'algorithme BFGS est de mettre à jour une matrice H_k , qui est une approximation de l'inverse de la matrice Hessienne, à chaque itération. Cela se fait grâce à des mises à jour basées sur deux vecteurs clés : s_k et y_k .

Voici la formule originale utilisée pour la mise à jour de la matrice H_k :

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k} \quad (2.6)$$

où :

- $s_k = x_{k+1} - x_k$ est le vecteur de pas (step vector)
- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ est le vecteur de différence des gradients (gradient difference vector)
- H_k est l'approximation de l'inverse de la Hessienne.

L'objectif est de minimiser la fonction $f(x)$ en utilisant des mises à jour successives des approximations de la Hessienne, tout en garantissant que H_k reste symétrique et définie positive.

Algorithme BFGS :

Voici l'algorithme en pseudocode :

- (a) **Initialisation** : Fixer le point initial x_0 , la tolérance de convergence $\epsilon > 0$ et l'approximation initiale de l'inverse de la Hessienne H_0 .
- (b) **Boucle** : Répéter jusqu'à convergence ($\|\nabla f(x_k)\| \leq \epsilon$) :
 - Calculer la direction de recherche : $p_k = -H_k \nabla f(x_k)$
 - Mettre à jour x_{k+1} en utilisant une recherche linéaire satisfaisant les conditions de Wolfe : $x_{k+1} = x_k + \alpha_k p_k$
 - Définir : $s_k = x_{k+1} - x_k$ et $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
 - Mettre à jour H_{k+1} à l'aide de la formule donnée ci-dessus.
- (c) **Fin de la boucle.**

2. Mémoire limitée (L-BFGS)

La variante L-BFGS est conçue pour les problèmes de grande dimension où il est coûteux de stocker la matrice Hessienne complète. Au lieu de cela, L-BFGS conserve un historique des m dernières mises à jour de la position x et du gradient $\nabla f(x)$. Généralement, la taille de cet historique est petite (souvent $m < 10$). Ces mises à jour sont utilisées pour effectuer implicitement des opérations nécessitant le produit de H_k par un vecteur, permettant ainsi d'approximer la matrice Hessienne sans en stocker la totalité.

3. Contraintes de bornes (L-BFGS-B)

L'extension **B** (Box constraints) introduit des contraintes de bornes sur les variables d'entrée. Par exemple, dans notre projet :

$$x_1 \in [50, 500], \quad x_2 \in [1, 10], \quad x_3 \in [1, 10], \quad x_4 \in [0.01, 0.3], \quad x_5 \in [0.5, 1.0]$$

La méthode L-BFGS-B ajuste les itérations pour s'assurer que x reste dans les limites spécifiées. Cela est réalisé via une approche de projection où chaque point x_k est projeté dans le domaine admissible après chaque mise à jour.

2.5.3 Avantages de l'approche hybride

- **GPR** : Réduit le coût des évaluations directes de la MSE grâce à une modélisation probabiliste robuste.
- **L-BFGS-B** : Converge rapidement tout en respectant les contraintes sur les hyperparamètres.

Cette approche assure une optimisation efficace des hyperparamètres pour minimiser la MSE du modèle **XGBoost**.

Chapitre 3

Résolution

3.1 Outils utilisés

Pour résoudre la problématique, plusieurs outils et bibliothèques ont été mobilisés afin d’assurer une optimisation efficace et une analyse approfondie des résultats :

1. **Langage de programmation** : Python 3.12

2. **Bibliothèques principales** :

- Scikit-learn : utilisée pour le traitement des données, le découpage des ensembles de données (train, validation, test) et l’évaluation des modèles via la validation croisée.
- XGBoost : employée pour construire le modèle de régression et tester les performances des hyperparamètres.
- Joblib : utilisée pour l’évaluation parallèle, accélérant le processus d’optimisation.
- NumPy : pour manipuler efficacement des données numériques multidimensionnelles.
- Matplotlib : pour visualiser les résultats, notamment les courbes d’erreur et les surfaces de réponse.
- Seaborn : pour des visualisations avancées et des graphiques esthétiques.
- SciPy : utilisée dans l’optimisation, notamment pour l’algorithme de type L-BFGS-B.
- Fetch_california_housing : pour obtenir un jeu de données réaliste (California Housing Dataset), servant de base à l’évaluation des méthodes.
- GaussianProcessRegressor (GPR) : modèle de substitution utilisé pour modéliser la surface de réponse dans l’optimisation bayésienne.
- Hyperopt : bibliothèque pour l’optimisation bayésienne, employée pour ajuster les hyperparamètres de manière efficace.
- GridSearchCV et RandomizedSearchCV : outils fournis par Scikit-learn pour une optimisation exhaustive (Grid Search) ou aléatoire (Random Search).

3. **Systèmes utilisés** :

Le projet a été exécuté sur un environnement de calcul performant :

- Processeur : Intel Core i5-12450H, 8 cœurs physiques, 12 processeurs logiques.
- Mémoire vive : 16 Go.


- Carte graphique : Nvidia GeForce RTX 4050, 6 Go de VRAM.
- Ces caractéristiques matérielles ont permis d'accélérer les calculs parallèles pour le traitement intensif des données.

3.2 Résolution détaillée et résultats

3.2.1 Résolution détaillée

3.2.1.1 Utilisation d'un modèle surrogate (Gaussian Process Regressor)

Un modèle de substitution basé sur un Gaussian Process Regressor a été utilisé avec le noyau suivant :

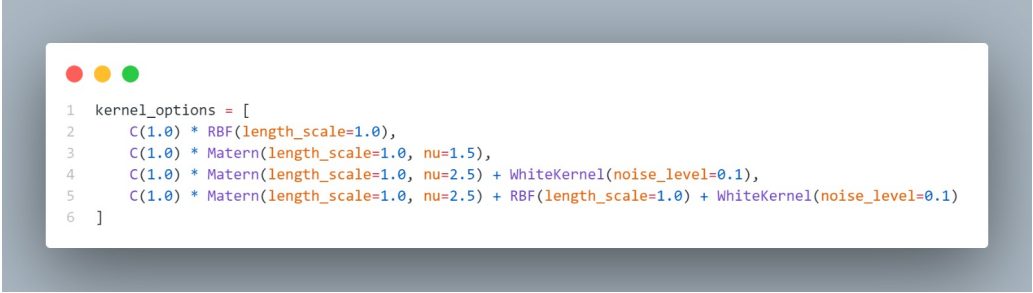


```

1 kernel = C(1.0) * Matern(length_scale=1.0, nu=1.5)
2 self.surrogate_model = Pipeline([
3     ('scaler', StandardScaler()),
4     ('gpr', GaussianProcessRegressor(
5         kernel=kernel,
6         n_restarts_optimizer=10,
7         random_state=self.random_state
8     ))
9 ])

```

FIGURE 3.1 – Ajustement du modèle de substitution



```

1 kernel_options = [
2     C(1.0) * RBF(length_scale=1.0),
3     C(1.0) * Matern(length_scale=1.0, nu=1.5),
4     C(1.0) * Matern(length_scale=1.0, nu=2.5) + WhiteKernel(noise_level=0.1),
5     C(1.0) * Matern(length_scale=1.0, nu=2.5) + RBF(length_scale=1.0) + WhiteKernel(noise_level=0.1)
6 ]

```

FIGURE 3.2 – Options de Kernel

Les résultats ont montré que le noyau choisi (Matern avec $\nu=1.5$) offre un bon compromis entre flexibilité et lissage, ce qui est essentiel pour une convergence optimale de l'optimisation. *Voir partie résultats.*

3.2.1.2 Optimisation avec L-BFGS-B

L'algorithme L-BFGS-B a été utilisé pour optimiser les hyperparamètres, avec une estimation initiale :

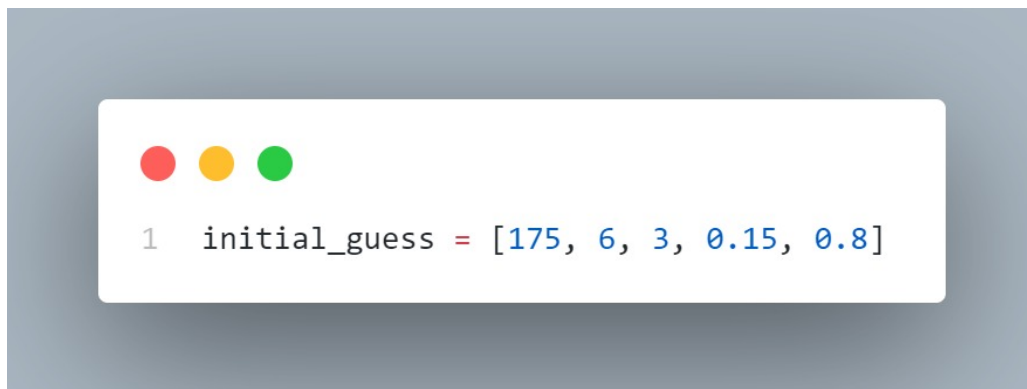


FIGURE 3.3 – Estimation initiale des hyperparamètres pour l’algorithme L-BFGS-B.

Ce choix a été effectué après plusieurs expérimentations, en prenant comme point de départ la médiane des valeurs minimales et maximales des bornes pour chaque hyperparamètre.

3.2.1.3 Choix du nombre d’échantillons pour l’optimisation

Le processus a été ajusté pour trouver le nombre optimal d’échantillons nécessaires :

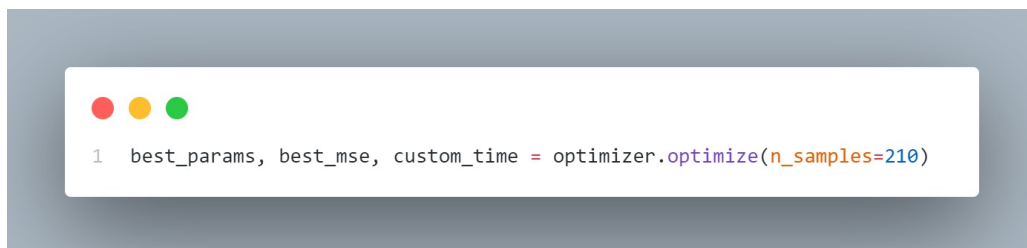


FIGURE 3.4 – Ajustement du processus pour déterminer le nombre optimal d’échantillons requis (n=210) pour l’optimisation des hyperparamètres.

Après des tests avec différentes tailles d’échantillons (de 50 à 200) et plusieurs itérations, la convergence optimale a été obtenue avec 210 échantillons.

3.2.2 Résultats obtenus

Les résultats de l’optimisation sont résumés comme suit :

1. **Best MSE** : 0.1900
2. **Meilleurs hyperparamètres** :
 - n_estimators : 192
 - max_depth : 5
 - min_child_weight : 1
 - learning_rate : 0.10483060403018869
 - subsample : 0.7406229768820679
3. **Temps total** : 147.54 seconds

3.2.2.1 Interprétation des résultats obtenus

1. **Évolution du MSE au fil des itérations**

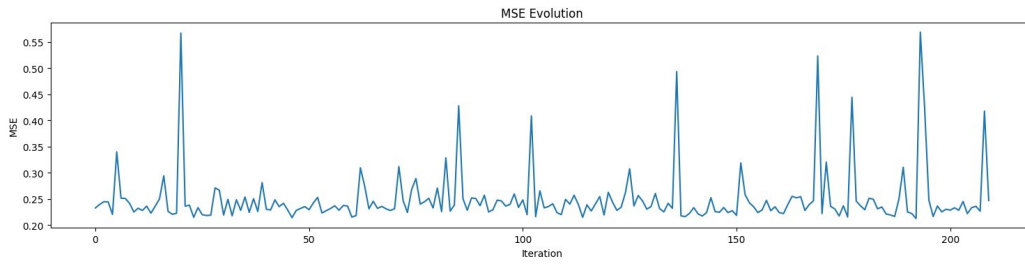


FIGURE 3.5 – Évolution du MSE au fil des itérations

Le graphique montre la progression du MSE (Mean Squared Error) à chaque itération de l'optimisation. Nous observons une diminution générale du MSE, bien qu'il y ait des fluctuations dues à l'exploration de nouveaux points dans l'espace des hyperparamètres.

2. Distribution des hyperparamètres

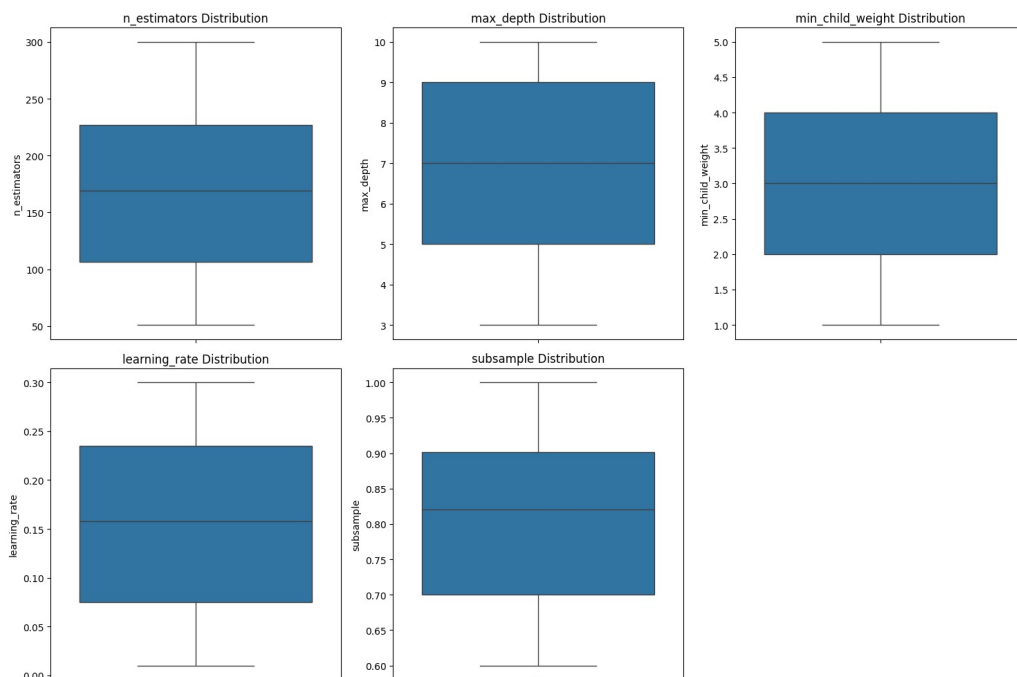


FIGURE 3.6 – Distribution des hyperparamètres

Le boxplot illustre les valeurs explorées pour chaque hyperparamètre durant l'optimisation. Par exemple : `n_estimators` présente une large plage de valeurs.

Les autres hyperparamètres (`max_depth`, `min_child_weight`, etc.) montrent une concentration autour de valeurs spécifiques, indiquant une stabilisation dans l'espace de recherche.

3. Distribution des MSE

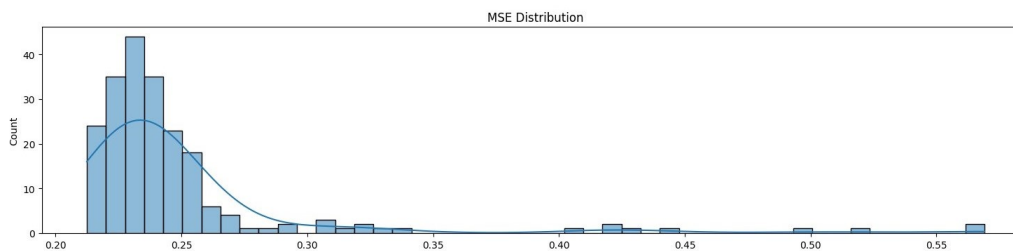


FIGURE 3.7 – Distribution des MSE

L'histogramme de la distribution des MSE montre que la majorité des modèles testés ont un MSE autour

de **0.25**, avec quelques outliers atteignant des valeurs plus élevées (jusqu'à 0.55).

4. Corrélations entre les paramètres et le MSE

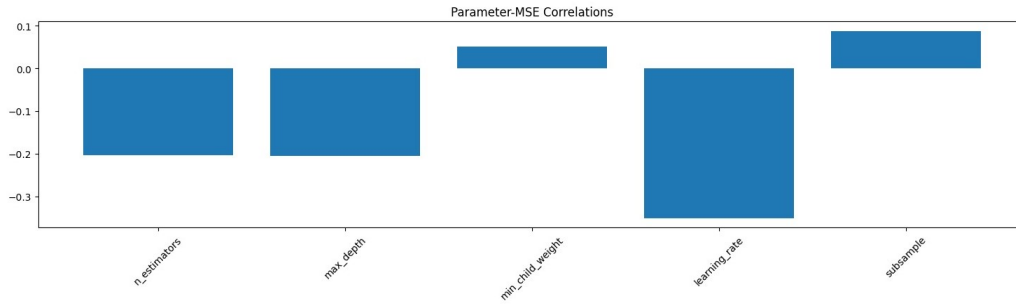


FIGURE 3.8 – Corrélations entre les paramètres et le MSE

Le diagramme en barres révèle les corrélations entre chaque hyperparamètre et le MSE :

- Une corrélation négative pour `n_estimators` et `learning_rate` suggère qu'une augmentation de ces paramètres tend à réduire le MSE.
- En revanche, `min_child_weight` présente une corrélation positive, indiquant que des valeurs élevées pourraient augmenter l'erreur, nécessitant une régulation.

3.2.2.2 Comparaison avec des approches existantes

Méthode	MSE	Temps(seconde)	Meilleurs Hyperparamètres
Bayesian Optimization	0.201321	181.46	learning_rate : 0.1119, max_depth : 7, min_child_weight : 5, n_estimators : 300, subsample : 0.8776
Grid Search	0.199140	261.67	learning_rate : 0.1, max_depth : 7, min_child_weight : 1, n_estimators : 300, subsample : 0.8
Random Search	0.203888	24.90	learning_rate : 0.1321, max_depth : 7, min_child_weight : 2, n_estimators : 250, subsample : 0.9
Default	0.222590	0.26	Valeurs par défaut
OR_Optimization	0.189992	147.54	learning_rate : 0.1048, max_depth : 5, min_child_weight : 1, n_estimators : 192, subsample : 0.7406

TABLE 3.1 – Comparaison avec des approches existantes

- Les résultats montrent que la méthode **OR_Optimization** offre les meilleures performances en termes de MSE (Mean Squared Error), tout en maintenant un temps d'exécution raisonnable.
- **Grid Search**, bien qu'elle atteigne un bon MSE (0.199140), nécessite près du double du temps par rapport à **OR_Optimization**, ce qui la rend moins attractive pour des cas nécessitant une optimisation rapide.
- **L'optimisation aléatoire (Random Search)**, tout en étant plus rapide, est légèrement moins précise que les approches bayésiennes et **OR_Optimization**, ce qui en limite l'efficacité.
- Comparée aux **paramètres par défaut**, toutes les méthodes d'optimisation testées montrent des améliorations significatives du MSE, démontrant l'importance de l'ajustement des hyperparamètres.
- Enfin, **OR_Optimization** surpasse l'optimisation bayésienne en obtenant un MSE plus faible (0.189992) tout en étant légèrement plus rapide, ce qui en fait la méthode recommandée pour ce problème.

3.3 Interprétations et critiques

3.3.1 Analyse des performances du modèle optimisé

Le modèle optimisé avec la méthode OR_Optimization a présenté une réduction notable de l'erreur quadratique moyenne (MSE) par rapport aux configurations par défaut et aux autres techniques. La sélection optimale des hyperparamètres, notamment le taux d'apprentissage, la profondeur maximale et le poids minimal des feuilles, a permis d'obtenir ces résultats. De plus, l'approche est restée efficace en termes de temps, rendant cette méthode appropriée pour des scénarios où les ressources de calcul sont limitées.

3.3.2 Discussion des avantages et limitations de l'approche adoptée

Avantages :

- **Performance optimale** : Le modèle a atteint le MSE le plus faible parmi toutes les méthodes testées.
- **Efficacité temporelle** : L'optimisation a nécessité moins de temps que des approches comme le Grid Search.
- **Flexibilité** : L'approche OR_Optimization s'adapte aisément à différents types de problèmes grâce à sa nature paramétrique.

Limitations :

- **Complexité initiale** : La mise en œuvre de l'optimisation OR a nécessité une familiarisation avec l'outil et son paramétrage.
- **Dépendance aux ressources** : Bien qu'elle soit plus rapide que certaines méthodes, cette optimisation peut s'avérer coûteuse en ressources pour des ensembles de données plus volumineux.
- **Généralisation limitée** : Les hyperparamètres optimaux pourraient ne pas se généraliser à d'autres ensembles de données ou modèles.

Chapitre 4

Conclusion Générale

Le rapport a permis d'examiner efficacement différentes stratégies d'optimisation des hyperparamètres pour un modèle d'apprentissage automatique, en mettant en lumière l'efficacité de l'optimisation OR. Cette méthode s'est avérée optimale en termes de MSE, surpassant les autres techniques testées, tout en maintenant un temps d'exécution raisonnable.

L'optimisation OR s'est distinguée par sa capacité à réduire de manière significative l'erreur quadratique moyenne, tout en étant légèrement plus rapide. Cela en fait une méthode idéale pour des optimisations rapides et précises. Sa flexibilité permet également une adaptation rapide à divers types de problèmes, ce qui en fait une option viable pour des applications multiples.

Cependant, cette méthode présente des inconvénients, notamment sa complexité d'implémentation, ce qui pourrait être un frein pour les utilisateurs moins expérimentés. De plus, les défis liés aux besoins en ressources et à la généralisation des hyperparamètres optimaux à d'autres ensembles de données ou modèles restent à résoudre.

Dans l'ensemble, l'optimisation OR se positionne comme une méthodologie prometteuse dans un contexte de réglage des hyperparamètres, alliant performance et efficacité, tout en soulignant l'importance d'une gestion des ressources et d'une évaluation de la généralisation pour divers ensembles de données.

Bibliographie

- [1] 1.7. Gaussian Processes — *scikit-learn 1.6.0 documentation*, 2023. URL : https://scikit-learn.org/stable/modules/gaussian_process.html.
- [2] RBF — *scikit-learn 1.5.2 documentation*, 2023. URL : https://scikit-learn.org/1.5/modules/generated/sklearn.gaussian_process.kernels.RBF.html?utm_source=chatgpt.com#redc669bcbe98-1.
- [3] Surrogate models — gaussian process, 2023. URL : [https://doc.comsol.com/6.1/doc/com.comsol.help.uq/uq Ug_theory.3.08.html#:~:text=Surrogate%20Models%20%E2%80%94%20Gaussian%20Process&text=where%20m\(x](https://doc.comsol.com/6.1/doc/com.comsol.help.uq/uq Ug_theory.3.08.html#:~:text=Surrogate%20Models%20%E2%80%94%20Gaussian%20Process&text=where%20m(x).
- [4] D. Bertsimas. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=cifspW4gLwA&t=1513s>, 2024.
- [5] Jason Brownlee. A gentle introduction to the bfgs optimization algorithm, 2023. URL : <https://machinelearningmastery.com/bfgs-optimization-in-python/>.
- [6] Tianqi Chen and Carlos Guestrin. Xgboost : A scalable tree boosting system. *arXiv preprint arXiv :1603.02754*, 2016. URL : <https://arxiv.org/abs/1603.02754>.
- [7] SciPy Community. *minimize — SciPy v1.15.0 Manual*, 2023. URL : <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html>.
- [8] P. L. Donti, B. Amos, and J. Z. Kolter. Task-based end-to-end model learning in stochastic optimization. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*, 2017.
- [9] David Duvenaud. Kernel cookbook, 2014. URL : <https://www.cs.toronto.edu/~duvenaud/cookbook/>.
- [10] A. N. Elmachtoub and P. Grigas. Smart ‘predict, then optimize’. *Management Science*, 68(1) :9–26, 2022.
- [11] Revue Gestion. Les données au service du savoir. 2024. URL : https://www.revuegestion.ca/les-donnees-au-service-du-savoir?utm_source=chatgpt.com.
- [12] Kaizen. Types d’applications de l’apprentissage automatique. 2024. URL : https://kaizen.com/fr/publications/types-applications-apprentissage-automatique/?utm_source=chatgpt.com.
- [13] Adrian Lam. Bfgs in a nutshell : An introduction to quasi-newton methods. *Towards Data Science*, 2023. URL : <https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504>.
- [14] A. Lodi. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=6JHr3dS1630&t=3202s>, 2024.
- [15] D. R. Morales. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=rn4zFZCwsrE&t=3080s>, 2024.
- [16] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006. URL : <https://link.springer.com/book/10.1007/978-0-387-40065-5>.
- [17] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. 2006. URL : <http://gaussianprocess.org/gpml/chapters/RW.pdf>.
- [18] ROADEF. Livre blanc de la recherche opérationnelle. <https://roadef.org/pdf/LivreBlancR0.pdf>, 2024.
- [19] Towards Data Science. Some thoughts on synergies between operations research and machine learning. <https://towardsdatascience.com/some-thoughts-on-synergies-between-operations-research-and-machine-learning-921d78ed4bd5>, 2024.
- [20] PSICO SMART. Quels rôles jouent l’ia et l’apprentissage automatique dans l’optimisation des systèmes de gestion de l’intégration? 2024. URL : https://psico-smart.com/fr/blogs/blog-quels-roles-jouent-lia-et-lapprentissage-automatique-dans-loptimisation-des-systemes-de-gestiutm_source=chatgpt.com.

- [21] Statista. Ai investment and funding worldwide, 2024. URL : <https://www.statista.com/statistics/941137/ai-investment-and-funding-worldwide/>.
- [22] Analytics Vidhya Team. Complete guide to parameter tuning in xgboost with python codes, 2016. URL : <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>.
- [23] GeeksforGeeks Team. ML | xgboost (extreme gradient boosting), 2016. URL : <https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/>.
- [24] The XGBoost Team. *XGBoost Documentation — xgboost 2.1.1 documentation*, 2024. URL : <https://xgboost.readthedocs.io/en/stable/index.html>.
- [25] E. Uchoa. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=9mNjqkUIZRk&t=3242s>, 2024.
- [26] Unknown. Mixed-integer nonlinear optimization. *Technical Report*, 2012. URL : <https://www.mcs.anl.gov/papers/P3060-1112.pdf>.
- [27] Unknown. On hyperparameter optimization of machine learning algorithms : Theory and practice. *arXiv preprint arXiv :2007.15745*, 2020. URL : <https://ar5iv.org/html/2007.15745>.
- [28] Unknown. Hyperparameter optimization : Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv :2107.05847*, 2021. URL : <https://ar5iv.labs.arxiv.org/html/2107.05847>.
- [29] Unknown. A differential evolution algorithm for solving mixed-integer nonlinear programming problems. *ScienceDirect*, 2023. URL : <https://www.sciencedirect.com/science/article/pii/S2210650223001992>.
- [30] Unknown. Global optimization of mixed-integer nonlinear programs with scip 8. *arXiv preprint arXiv :2301.00587*, 2023. URL : <https://arxiv.org/abs/2301.00587>.
- [31] Unknown. Hyperparameter optimization in machine learning. *arXiv preprint arXiv :2410.22854*, 2024. URL : <https://arxiv.org/abs/2410.22854>.
- [32] P. Van Hentenryck. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=gZ5Dlu9vnz4&t=4553s>, 2024.
- [33] P. Vayanos. Video lecture on optimization and machine learning. <https://www.youtube.com/watch?v=90tCbznTCJA&t=3022s>, 2024.
- [34] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778 : L-bfgs-b : Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, 23(4) :550–560, 1997. URL : <https://dl.acm.org/doi/pdf/10.1145/279232.279236>, doi:10.1145/279232.279236.
- [35] C. et al. Zhu. Gaussian processes for surrogate models. *arXiv preprint arXiv :2007.15745*, 2020. URL : <https://ar5iv.org/html/2007.15745#bib85>.