Challenge Auto-ML (G13)

Maelig Pesantez[1] Kilian Pousse[1] Arab Belamri[1]
(1) Le Mans Université, Avenue Olivier Messiaen, 72000 Le Mans, France
{maelig.pesantez.etu, kilian.pousse.etu,
arab.belamri.etu}@univ-lemans.fr

RÉSUMÉ

Ce travail présente le package AutoML, développé dans le cadre des modules Méthodologie pour l'Intelligence Artificielle, Méthodes Classiques pour l'Intelligence Artificielle et DevOps du Master 1 Informatique, spécialité Intelligence Artificielle à l'Université du Mans. L'objectif de ce projet est de concevoir un outil Python capable d'automatiser les étapes clés d'un processus d'apprentissage automatique : prétraitement des données, sélection et entraînement de modèle et évaluation des performances. Nous avons mis l'accent sur la reproductibilité expérimentale, la modularité du pipeline et la fiabilité des métriques produites. Cet article décrit l'architecture du système, les choix méthodologiques retenus ainsi que les résultats obtenus sur plusieurs jeux de données variés, afin d'évaluer la robustesse et la généricité de notre approche AutoML.

ABSTRACT

Auto-ML challenge

This work presents AutoML, a Python package developed as part of the Artificial Intelligence Methodology, Classical AI Methods, and DevOps courses of the Master's in Computer Science – Artificial Intelligence track at Le Mans University. The project aims to design a Python tool that automates key stages of a machine learning workflow, including data preprocessing, model selection and training, and performance evaluation. We emphasize experimental reproducibility, pipeline modularity, and the reliability of the generated metrics. This paper outlines the system architecture, discusses the methodological choices, and reports results obtained on multiple datasets to assess the robustness and generalization capacity of our AutoML approach.

MOTS-CLÉS : Auto-ML, Aprentissage automatique, flux de traitement, classification, regression .

KEYWORDS: Auto-ML, Machine Learning, pipeline, classification, regression.

Challenge Auto-ML 2025

Dans le cadre du challenge Auto-ML 2025 de l'Université Du Mans déstiné aux étudiants de première année du master informatique spécialité IA, nous avons eu l'occasion de concevoir un module python Auto-ML en groupe de trois étudiants. Nous détaillons ici les choix de conception que nous avons étés ammenés à faire concernant l'intégralité du pipeline de traitement.

Quelques informations sur les données que nous avons étés ammenés à traiter : les enseignant ont mis à notre disposition onze datasets de deux types, csv classique et dataset sparse. Pour chacun de ses datasets, deux types de modèles étaient applicables : les régressions et les classifications, pour certaines mutli-output.

Concernant les limites du package, celui-ci doit être éxécutable uniquement via CPU, fermant directement la porte à l'utilisation des GPUs et des bibliothèques correspondantes : pytorch, cuda... Nous avions en revanche l'autorisation d'utiliser des packages CPU du domaine ML comme sklearn.

Introduction

L'apprentissage automatique (*Machine Learning*) s'est imposé ces dernières années comme une technologie incontournable pour l'analyse de données complexes et la prise de décision automatisée. Cependant, la conception de modèles performants demeure une tâche experte, itérative et coûteuse en temps. Le processus traditionnel implique de nombreuses étapes manuelles, allant du pré-traitement des données à la sélection de l'algorithme adéquat, en passant par l'optimisation fine des hyperparamètres. Cette complexité constitue souvent un frein à l'adoption rapide et généralisée des solutions d'intelligence artificielle.

C'est dans ce cadre qu'émerge le concept d'apprentissage automatique automatisé, ou *AutoML* (*Automated Machine Learning*). L'AutoML vise à retirer cette complexité en automatisant l'intégralité du pipeline de modélisation, permettant ainsi de produire des modèles robustes avec une intervention humaine minimale.

Ce rapport présente les travaux réalisés dans le cadre du projet commun « Méthodologie IA et méthodes classiques ». L'objectif principal était de concevoir et de développer un pipeline AutoML complet, capable d'ingérer des jeux de données variés, d'entraîner et de sélectionner les meilleurs modèles, et de fournir des prédictions fiables de manière autonome. Nous détaillerons dans ce document l'architecture logicielle retenue, les choix méthodologiques effectués pour l'optimisation des hyperparamètres sur le cluster de calcul *Skinner*, ainsi que l'évaluation des performances de notre solution face à des capacités de généralisation exigées par le challenge.

1 Contexte du Projet

Le projet s'inscrit dans un challenge de *Machine Learning* visant à simuler un environnement de production où l'efficacité, la robustesse et l'automatisation sont primordiales. Cette section détaille les enjeux conceptuels de l'AutoML ainsi que les contraintes techniques et opérationnelles spécifiques à cette étude.

1.1 Le Paradigme AutoML

L'AutoML vise avant tout à automatiser les étapes traditionnellement réalisées par essais successifs, en les confiant à des procédures algorithmiques structurées., le système développé doit être capable de prendre en entrée des données brutes et de délivrer en sortie des prédictions, en automatisant les phases intermédiaires critiques :

— **Prétraitement des données** : nettoyage, normalisation et séparation des jeux de données (*train/validation/test*).
— **Sélection de modèle** : exploration de différentes familles d'algorithmes via la librairie *scikit-learn*.

— **Optimisation des hyperparamètres** : recherche de la configuration optimale pour chaque jeu de données spécifique.
— **Évaluation** : utilisation de métriques adaptées pour valider la pertinence des modèles retenus.

Cette automatisation vise à réduire les biais humains potentiels et à garantir une reproductibilité stricte des résultats.

1.2 Environnement Technique et Infrastructure

La dimension computationnelle du projet impose l'utilisation d'une infrastructure de calcul haute performance (*High Performance Computing*, HPC). Le développement et les entraînements sont réalisés sur le cluster étudiant *Skinner*. L'usage de ce cluster répond à la nécessité de traiter des volumes de données conséquents et d'exécuter des algorithmes d'optimisation coûteux en ressources.

Le pipeline doit respecter une interface utilisateur minimale et standardisée (par exemple : `automl.fit()`, `automl.eval()`), facilitant son utilisation par des tiers. De plus, une contrainte forte de généralisation est imposée : le système est développé sur un corpus de données initial (situé dans `/info/corpus/ChallengeMachineLearning`), mais doit être suffisamment robuste pour performer sur de nouvelles données ajoutées ultérieurement, inconnues lors de la phase de conception.

1.3 Cadre Organisationnel et Livrables

Ce projet ne se limite pas à la performance algorithmique ; il intègre également des exigences de génie logiciel strictes en prévision d'une réutilisation future dans le cadre du module *DevOps*. Le code doit être modulaire, lisible et packagé sous forme de librairie Python facilement installable.

Le travail est mené de manière collaborative via un gestionnaire de version (*GitHub*), impliquant l'usage de bonnes pratiques telles que la gestion de branches, la revue de code et la documentation technique.

2 Description et Formalisme des Données

Pour bien comprendre le fonctionnement de notre pipeline, il faut d'abord s'intéresser au format des données que nous avons reçues. Dans ce challenge, l'organisation des fichiers est toujours la même pour faciliter le traitement automatique. Chaque dataset est décomposé en trois fichiers distincts (.data, .solution, .type)

Le fichier `.data` contient la matrice des caractéristiques (*features*). Chaque ligne représente une instance unique et chaque colonne correspond à une variable descriptive. Ces données brutes présentent plusieurs caractéristiques clés que le pipeline doit gérer :

— **Hétérogénéité** : les variables peuvent être de nature numérique ou catégorielle.
— **Données manquantes** : le jeu de données n'est pas garanti complet ; des valeurs manquantes (*NaN*) peuvent être présentes, nécessitant une stratégie d'imputation automatique.

— **Absence d'étiquettes** : ce fichier ne contient strictement que les données d'entrée, sans la cible à prédire.

2.1 La Référence (`.solution`)

Le fichier `.solution` contient les cibles (*targets*) correspondant aux observations du fichier `.data`. La structure de ce fichier définit implicitement la typologie du problème d'apprentissage supervisé :

— **Régression** : une colonne de valeurs continues.
— **Classification binaire** : une colonne de valeurs discrètes (0 ou 1).
— **Classification multi-classes** : une colonne d'entiers ou une représentation encodée.
— **Classification multi-étiquettes (*multi-label*)** : une matrice binaire où chaque ligne comporte plusieurs colonnes (par exemple : 1 0 1), indiquant l'appartenance simultanée d'une instance à plusieurs classes indépendantes.

Contrairement à un déploiement réel où les cibles de test sont inconnues, nous disposons ici de l'ensemble des solutions, permettant une évaluation locale complète des performances (via un découpage *train/test*) avant la soumission finale.

2.2 Les Types (`.type`)

Le fichier `.type` agit comme un descripteur de métadonnées essentiel pour l'automatisation du prétraitement. Il qualifie explicitement chaque colonne du fichier `.data` :

— **Numerical** : désigne une variable quantitative (continue ou discrète) pouvant subir des opérations statistiques (moyenne, écart-type).
— **Categorical** : désigne une variable qualitative (codes ou chaînes de caractères) nécessitant une transformation (*encodage*) avant d'être interprétable par les algorithmes mathématiques.

Ce fichier permet au système AutoML d'appliquer automatiquement les stratégies de nettoyage et d'encodage adéquates sans avoir à inférer le type des données, réduisant ainsi le risque d'erreurs d'interprétation.

3 Chargement et Structuration des Jeux de Données

Le chargement des jeux de données constitue la première étape du pipeline AutoML. Cette phase vise à transformer des fichiers hétérogènes fournis par le challenge en une représentation interne unifiée, compatible avec l'ensemble des étapes ultérieures de nettoyage, de sélection de modèle et d'entraînement. Le module développé doit ainsi être capable de gérer à la fois des jeux de données tabulaires classiques et des jeux de données de type *sparse*.

3.1 Chargement des cibles et gestion du multi-output

Les variables cibles sont chargées à partir du fichier `.solution`, lu sous forme de table sans en-tête. Deux cas sont distingués automatiquement en fonction du nombre de colonnes détectées :

— **Sortie mono-dimensionnelle** : lorsque le fichier ne contient qu'une seule colonne, la tâche correspond à un problème supervisé standard (classification ou régression mono-sortie).
— **Sortie multi-dimensionnelle** : lorsque plusieurs colonnes sont présentes, celles-ci sont interprétées comme des cibles multiples (*multi-output*), couvrant à la fois les cas de régression multi-sortie et de classification multi-étiquettes.

Les colonnes cibles sont renommées de manière systématique (`target`, `target_i`) afin de garantir une manipulation homogène dans l'ensemble du pipeline. Le choix du type précis de tâche (classification, régression, multi-label) est volontairement différé et sera déterminé ultérieurement lors de la phase de sélection automatique du modèle.

3.2 Détection et chargement des données explicatives

Le fichier `.data` peut correspondre à deux formats distincts :

— un format tabulaire dense classique, où chaque ligne représente une instance et chaque colonne une variable explicative ;
— un format *sparse*, dans lequel chaque ligne est décrite par une liste de couples `indice:valeur`, représentant uniquement les variables non nulles.

La détection du format est réalisée automatiquement par l'analyse lexicale du fichier. La présence d'au moins un caractère `:` dans une ligne déclenche l'interprétation du fichier comme un jeu de données sparse.

Dans ce cas, les lignes sont parcourues afin d'extraire les indices et valeurs associés à chaque instance. Le nombre total de variables est estimé dynamiquement à partir de l'indice maximal observé. Les données sont ensuite reconstruites sous forme d'une matrice dense, initialisée à zéro, puis complétée avec les valeurs présentes dans la représentation sparse.

3.3 Choix de représentation et justification

Bien que le format sparse permette théoriquement des gains substantiels en mémoire et en temps de calcul, nous avons fait le choix assumé de convertir systématiquement les données en `DataFrame` *pandas* denses. Ce choix répond à plusieurs motivations :

— assurer une compatibilité totale avec les étapes de prétraitement, de nettoyage et de profilage automatique des données ;
— simplifier l'implémentation du pipeline en s'appuyant sur les abstractions vues en cours et largement utilisées dans l'écosystème *scikit-learn* ;
— garantir une interface uniforme entre les jeux de données, indépendamment de leur format d'origine.

Cette décision peut induire un surcoût computationnel pour certains jeux de données très creux, mais elle permet de maintenir une architecture logicielle cohérente, lisible et robuste dans le cadre pédagogique du projet.

4 Méthodologie de Nettoyage et Prétraitement des Données

Le développement du module de prétraitement a suivi une approche itérative, guidée par la confrontation entre les besoins théoriques de qualité de données et les contraintes réelles de performance sur le cluster.

4.1 De l'approche naïve à l'automatisation

Dans la phase initiale du projet, nous avons adopté une stratégie exploratoire *naïve*. L'objectif était d'obtenir rapidement un pipeline fonctionnel (*baseline*) en appliquant des règles de nettoyage simples, telles que la suppression systématique des lignes contenant des valeurs manquantes. Bien que rapide, cette méthode entraînait une perte d'information critique sur les jeux de données lacunaires.

Pour pallier ce défaut, nous nous sommes tournés vers *ydata-profiling*, un outil puissant d'analyse exploratoire automatisée. Cette librairie génère un profil complet du jeu de données, détectant automatiquement les types d'inférence, les valeurs manquantes, les distributions statistiques et les corrélations complexes.

Cependant, l'intégration systématique de cet outil a révélé un goulot d'étranglement majeur. Sur les jeux de données à haute dimensionnalité (plusieurs milliers de colonnes), le coût computationnel — spécifiquement le calcul de la matrice de corrélation en $O(n^2)$ — devenait prohibitif. Les temps d'exécution explosaient, ralentissant considérablement le pipeline global.

Face à ce constat, nous avons développé une stratégie optimisée et hybride, capable de tirer parti de la puissance de *ydata-profiling* tout en garantissant la scalabilité.

4.2 Stratégie Adaptative Implémentée

Notre solution finale repose sur une logique conditionnelle qui adapte la profondeur du nettoyage à la complexité du *dataset*.

4.2.1 Gestion de la Haute Dimensionnalité (*Safe Mode*)

Lorsque le nombre de colonnes dépasse un seuil critique (fixé à 2000), le pipeline bascule automatiquement en *Safe Mode*. Dans ce mode, le profilage coûteux est désactivé au profit d'opérations vectorisées natives via *pandas*, beaucoup plus rapides :

- **Nettoyage minimal** : suppression des colonnes constantes (variance nulle).
- **Imputation rapide** : les valeurs manquantes sont comblées par la médiane (numérique) ou le mode (catégoriel) sans analyse contextuelle approfondie.

Cette approche permet de traiter des jeux de données massifs sans provoquer de dépassement de mémoire (*Out Of Memory*) ou de temps limite.

4.2.2 Analyse Optimisée (*Standard Mode*)

Pour les jeux de données de taille raisonnable, nous utilisons *ydata-profiling* en mode optimisé (`minimal=True`). Ce paramétrage désactive les calculs graphiques superflus tout en conservant les métadonnées essentielles pour un nettoyage fin :

Traitement des Valeurs Manquantes

- **Suppression ($> 30\%$)** : les variables trop lacunaires sont éliminées.
- **Imputation ($\leq 30\%$)** : remplacement par la médiane (numérique) ou le mode (catégoriel).

Réduction de la Multicolinéarité Grâce au profilage, nous identifions les paires de variables présentant une corrélation de Pearson supérieure à 0.85. Pour éviter la redondance d'information, l'une des deux variables est supprimée, prioritairement celle ayant le plus de valeurs manquantes.

Cette méthodologie assure un compromis optimal entre la robustesse statistique nécessaire à l'AutoML et l'efficacité computationnelle requise par le challenge.

5 Détection automatique du type de tâche et sélection du modèle

Une fois les données chargées et nettoyées, le pipeline AutoML procède à une étape centrale : la détermination automatique du type de tâche d'apprentissage, suivie de la sélection du modèle le plus adapté parmi un ensemble de candidats. Cette phase vise à éliminer toute intervention manuelle tout en garantissant une adaptation raisonnable aux caractéristiques du jeu de données.

5.1 Détection du type de tâche

La détection du type de tâche repose exclusivement sur l'analyse de la structure et du type des variables cibles. Aucune information externe n'est requise, ce qui permet une automatisation complète du processus.

Trois cas sont distingués :

- **Classification multi-étiquettes (*multi-label*)** : lorsque la cible est constituée de plusieurs colonnes, la tâche est interprétée comme un problème multi-sortie. Ce cas correspond typiquement à des situations où chaque instance peut appartenir simultanément à plusieurs classes indépendantes.
- **Classification** : lorsque la cible est mono-dimensionnelle et de type catégoriel (chaîne de caractères, catégorie explicite) ou présente un nombre limité de valeurs distinctes (seuil fixé empiriquement à 20), le problème est considéré comme une classification.
- **Régression** : lorsque la cible est numérique continue et présente une forte cardinalité, la tâche est traitée comme un problème de régression.

Cette approche heuristique, bien que simple, s'est révélée suffisante dans le cadre du challenge, où les jeux de données respectent des conventions relativement homogènes.

5.2 Gestion du cas multi-output

Dans le cas des tâches multi-sorties, le pipeline s'appuie sur une stratégie générique fondée sur le méta-modèle `MultiOutputClassifier`. Celui-ci encapsule un classificateur de base — en l'occurrence un arbre de décision — afin d'apprendre indépendamment chaque sortie cible.

Ce choix présente plusieurs avantages :
- compatibilité avec un large éventail de jeux de données multi-labels ;
- simplicité d'implémentation et robustesse ;
- intégration transparente dans le reste du pipeline AutoML.

Dans ce cas précis, aucune comparaison de modèles n'est effectuée, le modèle étant directement sélectionné afin de garantir la stabilité du processus.

5.3 Espace de modèles candidats

Pour les tâches de classification et de régression mono-sortie, un ensemble de modèles candidats est défini à l'avance, en s'appuyant sur des algorithmes classiques et éprouvés de l'écosystème *scikit-learn*, ainsi que sur des bibliothèques spécialisées telles que *XGBoost* et *LightGBM*.

Classification Les modèles de classification incluent notamment :
- k plus proches voisins (KNN),
- régression logistique,
- forêts aléatoires,
- AdaBoost,
- Naive Bayes gaussien,
- classifieur Ridge,
- modèles à base de gradient boosting (*XGBoost*, *LightGBM*).

Régression Pour la régression, l'espace de recherche comprend :
- régression linéaire et Ridge,
- forêts aléatoires,
- AdaBoost,
- modèles de gradient boosting (*XGBoost*, *LightGBM*).

Ces modèles pour la plupart vus en cours ont été choisis pour leur complémentarité, leur robustesse et leur capacité à traiter des données tabulaires hétérogènes.

5.4 Procédure de sélection du modèle

La sélection du modèle repose sur une évaluation comparative via validation croisée. Chaque modèle candidat est évalué sur un sous-ensemble des données à l'aide d'une validation croisée à trois plis.

Afin de maîtriser le coût computationnel sur les grands jeux de données, une stratégie de sous-échantillonnage est appliquée lorsque le nombre d'instances dépasse un seuil fixé à 5000. Dans ce cas, un échantillon aléatoire est utilisé pour la phase de sélection, sans impact sur l'entraînement final du modèle retenu.

Les métriques utilisées dépendent du type de tâche :

- **Accuracy** pour les problèmes de classification,
- **coefficient de détermination** R^2 pour les problèmes de régression.

Le modèle obtenant le meilleur score moyen est sélectionné pour la suite du pipeline.

6 Choix des hyper-paramètres

L'optimisation des hyper-paramètres constitue une étape clé du pipeline *AutoML*, permettant d'améliorer significativement les performances des modèles. Afin de s'adapter à différents contextes d'utilisation et de complexité, plusieurs méthodes d'optimisation sont proposées.

La coexistence de ces trois stratégies d'optimisation permet au package AutoML de s'adapter à différents scénarios d'utilisation, allant d'une optimisation rapide et simple à une recherche approfondie de performances maximales.

6.1 Stratégies d'optimisation

Le choix de la méthode d'optimisation est contrôlé par le paramètre `train_method`, qui associe chaque stratégie à une fonction dédiée :

- **`"h"` :** optimisation heuristique par recherche sur grille (*Grid Search*)
- **`"g"` :** optimisation par descente de gradient
- **`"l"` :** optimisation via une bibliothèque spécialisée (*Optuna*)

Cette approche permet de proposer un compromis entre simplicité, temps de calcul et qualité des modèles obtenus.

6.2 Optimisation heuristique (Grid Search)

La méthode heuristique repose sur une recherche exhaustive sur une grille prédéfinie d'hyper-paramètres, à l'aide de la validation croisée. Pour chaque type de modèle, un espace de recherche spécifique est défini (par exemple pour Random Forest, XGBoost ou LightGBM) La métrique d'évaluation est choisie dynamiquement selon le type de tâche :

- F1-score macro pour les problèmes de classification
- Coefficient de détermination R^2 pour les problèmes de régression

Cette méthode est robuste et simple à mettre en oeuvre, mais peut devenir coûteuse en temps de calcul lorsque l'espace de recherche est large.

6.3 Optimisation par descente de gradient

La méthode par descente de gradient est spécifiquement conçue pour les modèles compatibles avec l'apprentissage incrémental, tels que `SGDClassifier` et `SGDRegressor`. Elle repose sur l'utilisation de la méthode `partial_fit` et permet :

— un apprentissage par mini-lots (*mini-batch*)
— un contrôle explicite du taux d'apprentissage
— un nombre configurable d'époques

Lorsque le modèle sélectionné n'est pas compatible avec ce type d'optimisation, la méthode est ignorée et le modèle est conservé sans modification. Cette stratégie offre une optimisation efficace pour les grands jeux de données tout en limitant la consommation mémoire.

6.4 Optimisation via une bibliothèque spécialisée

La troisième approche repose sur l'utilisation de la bibliothèque *Optuna*, qui implémente des algorithmes d'optimisation bayésienne pour l'exploration intelligente de l'espace des hyper-paramètres.

Pour chaque type de modèle (Random Forest, XGBoost, LightGBM, régression logistique), un espace de recherche spécifique est défini. Chaque configuration candidate est évaluée à l'aide d'une validation croisée, et l'objectif est de maximiser une métrique adaptée au type de tâche. Cette méthode permet :

— une exploration plus efficace de grands espaces de recherche
— une meilleure convergence vers des configurations performantes
— une réduction du nombre d'essais nécessaires par rapport à une recherche exhaustive

En contrepartie, elle introduit une dépendance externe et un coût computationnel potentiellement plus élevé.

7 Évaluation des performances et comparaison aux baselines

L'évaluation des performances constitue l'étape finale du pipeline AutoML. Elle vise à mesurer objectivement la qualité prédictive du modèle sélectionné, tout en fournissant des points de comparaison simples permettant de contextualiser les résultats obtenus. Cette évaluation est réalisée de manière entièrement automatique et s'adapte au type de tâche détecté (classification, régression ou classification multi-étiquettes).

7.1 Métriques d'évaluation

Le choix des métriques dépend de la nature du problème d'apprentissage supervisé.

Classification et classification multi-étiquettes Pour les tâches de classification, les métriques suivantes sont calculées :

— **Accuracy**, mesurant la proportion globale de prédictions correctes ;
— **F1-score** décliné en moyennes micro, macro et pondérée, afin de tenir compte des déséquilibres de classes ;
— **Précision** et **rappel**, également évalués selon des moyennes micro et macro.

Ces métriques permettent d'obtenir une vision à la fois globale et détaillée du comportement du modèle, en particulier dans les contextes multi-classes ou multi-labels.

Régression Pour les tâches de régression, l'évaluation repose sur des métriques continues standard :

- **Erreur quadratique moyenne (MSE)** ;
- **Racine de l'erreur quadratique moyenne (RMSE)** ;
- **Erreur absolue moyenne (MAE)** ;
- **Coefficient de détermination** R^2.

Ces indicateurs permettent d'évaluer à la fois l'erreur moyenne de prédiction et la capacité du modèle à expliquer la variance des données.

7.2 Génération de baselines

Afin de contextualiser les performances du modèle sélectionné, deux baselines simples sont systématiquement générées :

- **Baseline aléatoire** : les prédictions sont générées de manière aléatoire à partir de la distribution observée des cibles ;
- **Baseline constante** : prédiction d'une valeur fixe (classe majoritaire ou moyenne des cibles selon le type de tâche).

Ces baselines constituent des points de référence faibles mais essentiels, permettant de vérifier que le modèle appris capture effectivement des régularités non triviales dans les données.

7.3 Comparaison et agrégation des résultats

Les performances du modèle AutoML sont comparées directement à celles des baselines à l'aide des mêmes métriques. Pour les tâches de classification, l'accent est mis sur l'accuracy et le F1-score macro, tandis que pour la régression, le coefficient R^2 et les erreurs moyennes sont privilégiés.

Dans le cas de la régression, les scores R^2 négatifs obtenus par certaines baselines sont tronqués à zéro afin de faciliter l'interprétation et la comparaison.

L'ensemble des métriques est regroupé dans une structure unifiée, puis affiché sous forme tabulaire afin de faciliter l'analyse et la comparaison entre modèles et jeux de données.

7.4 Discussion

Cette stratégie d'évaluation permet d'obtenir une mesure robuste et interprétable des performances du pipeline AutoML, tout en restant compatible avec des tâches variées. Bien que reposant sur des métriques classiques, elle fournit un cadre cohérent pour analyser la capacité de généralisation des modèles et justifier leur sélection dans le cadre du challenge.

8 Interface publique

L'interface publique du package *AutoML* regroupe l'ensemble des fonctions et méthodes accessibles à l'utilisateur final. Elle définit le contrat d'utilisation du package et masque les détails d'implémenta-

tion internes.

Elle propose deux modes d'utilisation : une utilisation *haut niveau*, entièrement automatisée, et une utilisation *bas niveau*, offrant un contrôle plus fin sur les étapes d'apprentissage et d'évaluation.

8.1 Fonctions de haut niveau

Ce mode d'utilisation permet d'exécuter automatiquement l'ensemble du pipeline *AutoML* via une unique méthode.

- **`load_data(data_path, solution_path, dataset_name) :`** charge le jeu de données d'apprentissage à partir du chemin `data_path` ainsi que les solutions associées depuis `solution_path`. Le paramètre `dataset_name` permet d'identifier le jeu de données utilisé. Cette méthode effectue également les étapes initiales de validation et de préparation des données.
- **`run(data_path, solution_path, dataset_name, train_method) :`** exécute l'ensemble du pipeline AutoML. Cette méthode charge les données, entraîne automatiquement les modèles selon la stratégie définie par `train_method`, évalue leurs performances et sélectionne le meilleur modèle obtenu. Elle constitue la méthode principale d'utilisation du package.

8.2 Utilisation bas niveau

Ce mode d'utilisation offre un contrôle explicite sur les différentes étapes du processus d'apprentissage automatique.

- **`fit(data_path, solution_path, train_method) :`** entraîne le modèle AutoML sur les données d'apprentissage fournies. Cette méthode déclenche la recherche automatique du meilleur modèle et ajuste ses paramètres.
- **`predict(test_path) :`** génère des prédictions à partir de nouvelles données en utilisant le modèle entraîné.
- **`eval() :`** évalue les performances du modèle sur un jeu de données de test à l'aide de métriques adaptées au problème (classification ou régression).