Prédiction du niveau d’obésité

APPLICATION DE TECHNIQUES D’APPRENTISSAGE MACHINE SUR LE JEU DE DONNEES UCI

Lyes Babou – A00235273

INFO4301 – APPRENTISSAGE MACHINE | AUT2025

Table des matières

1. Introduction

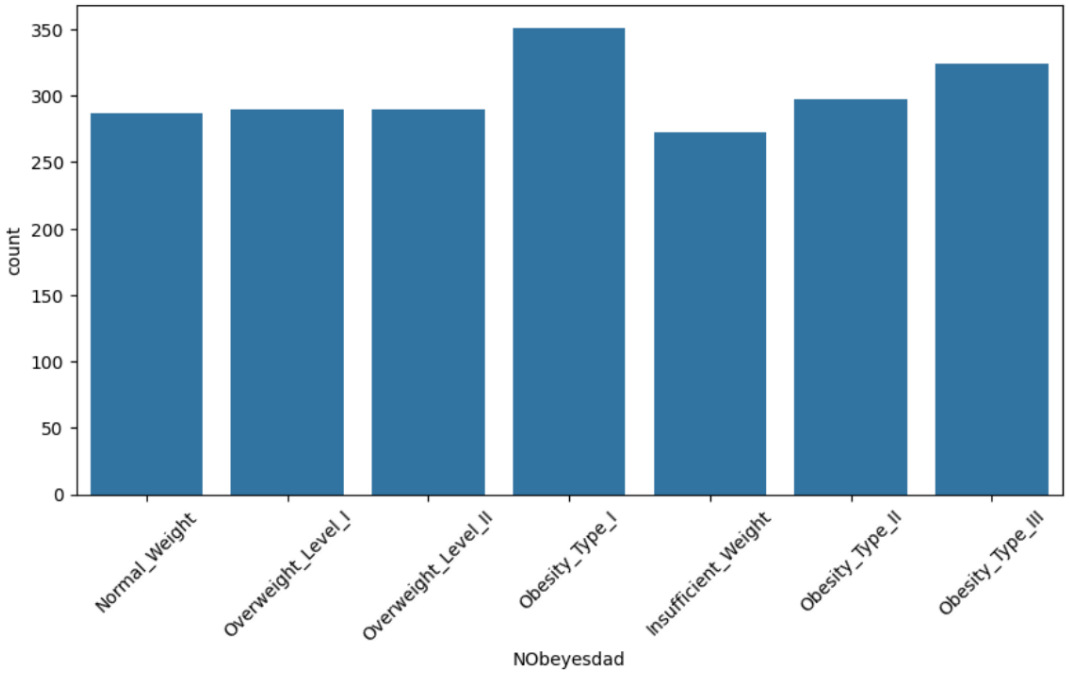
- Le but de ce projet est d'appliquer différentes techniques d'apprentissage machine afin de prédire le niveau d'obésité d'un individu à partir de ses habitudes alimentaires, de son mode de vie et de certaines caractéristiques physiques. L'obésité représente aujourd'hui un enjeu de santé important, et ce type de modèle peut aider à mieux comprendre les facteurs qui influencent les différents niveaux de poids.
- Pour réaliser ce travail, j'ai utilisé le jeu de données "Estimation of Obesity Levels Based on Eating Habits and Physical Condition" provenant de l'UCI Machine Learning Repository. Il contient plusieurs variables liées à l'âge, au poids, aux habitudes alimentaires, à l'activité physique et au style de vie. L'objectif était de construire un modèle capable de classer chaque individu dans l'une des catégories d'obésité définies dans le dataset.
- Le projet consiste à explorer les données, préparer le jeu de données pour l'entraînement, tester plusieurs algorithmes de classification, optimiser leurs hyperparamètres et analyser leurs performances. À la fin, le but est d'identifier le modèle le plus performant et d'évaluer sa capacité de généralisation sur un ensemble de test séparé.

2. Exploration des données (EDA)

- L'exploration des données a commencé par une analyse générale de la structure du dataset. Celui-ci contient plusieurs variables numériques (comme l'âge, le poids, la taille, la fréquence des repas, etc.) ainsi que plusieurs variables catégorielles (comme le genre, le type de transport, la consommation d'alcool, les antécédents familiaux, etc.). La variable cible NObeyesdad comporte sept classes représentant différents niveaux d'obésité.
- Les statistiques descriptives montrent que les valeurs numériques sont cohérentes et qu'il n'y a pas d'anomalies évidentes. Par exemple, les valeurs d'âge, de poids et de taille sont dans des plages réalistes, et les moyennes/écarts-types ne montrent pas de comportements extrêmes. Les visualisations (histogrammes et countplots) confirment que les distributions semblent normales et ne présentent pas de valeurs aberrantes qui nécessiteraient d'être modifiées ou supprimées.
- Un point essentiel de l'EDA était la vérification des valeurs manquantes. Le résultat montre qu'il n'y a aucune valeur manquante dans le dataset, ce qui est assez rare pour un jeu de données réel. Cela signifie que je n'ai pas eu besoin

d’imputer ou de nettoyer des colonnes, ce qui simplifie considérablement la préparation des données.

- J’ai également examiné la distribution des classes de la variable cible pour vérifier s’il existait un déséquilibre important.



Les proportions des classes sont relativement comparables et aucune classe n’est sous-représentée de manière critique. Par conséquent, il n’a pas été nécessaire d’utiliser des techniques comme l’oversampling, l’undersampling ou SMOTE. Le dataset est naturellement équilibré, ce qui permet d’entraîner les modèles sans risque majeur de biais vers une classe dominante.

- En résumé, l’EDA montre que le dataset est propre, complet, bien structuré et prêt à être utilisé pour la modélisation sans nécessiter de modifications particulières. Les seules transformations nécessaires concerneront la préparation normale pour l’apprentissage machine, comme l’encodage des variables catégorielles et la normalisation des variables numériques.

3. Préparation des données

Après l’exploration du dataset, la prochaine étape consistait à préparer les données pour l’entraînement des modèles. Le jeu de données contient à la fois des variables numériques (par exemple : âge, taille, poids, consommation d’eau) et des variables catégorielles (par exemple : genre, type de transport, antécédents familiaux). Comme les modèles d’apprentissage supervisé ne peuvent pas traiter

des catégories sous forme de texte, il était nécessaire de transformer ces informations avant l'entraînement.

La première étape a été de séparer les données en deux ensembles :

- X : qui contient toutes les variables explicatives,
- Y : qui contient uniquement la variable cible NObeyesdad.

Ensuite, j'ai séparé les colonnes en deux groupes :

- Colonnes numériques,
- Colonnes catégorielles,

afin d'appliquer un prétraitement approprié à chacune.

Pour les variables numériques, j'ai utilisé un StandardScaler, ce qui permet de normaliser les valeurs (moyenne 0 et variance 1). Cette étape est particulièrement importante pour les modèles sensibles aux variations d'échelle, comme les réseaux de neurones, SVM ou KNN.

Pour les variables catégorielles, j'ai utilisé un OneHotEncoder, qui transforme chaque catégorie en plusieurs colonnes binaires. Cette transformation permet aux modèles d'interpréter correctement les valeurs qualitatives.

Comme la variable cible est textuelle, j'ai utilisé un LabelEncoder pour convertir chaque classe d'obésité en un nombre entier. Cela permet aux modèles de travailler efficacement avec une cible multiclasses.

Enfin, j'ai divisé les données en trois ensembles distincts :

- 60 % pour l'entraînement (train),
- 20 % pour la validation (validation),
- 20 % pour le test final (test).

Cette division a été réalisée en deux étapes avec la fonction train_test_split, en utilisant l'option stratify pour conserver la distribution des classes. L'ensemble d'entraînement a servi pour ajuster les modèles, l'ensemble de validation pour comparer les performances et optimiser les hyperparamètres, et l'ensemble de test uniquement pour l'évaluation finale.

L'ensemble du prétraitement (scaling et encodage) a été intégré dans un ColumnTransformer, puis associé à chaque modèle à l'aide d'un Pipeline. Cela garantit que toutes les transformations sont appliquées automatiquement et de manière cohérente à chaque étape du processus.

4. Développement des modèles

Après la préparation des données, j'ai testé plusieurs modèles d'apprentissage supervisé afin de comparer leurs performances sur ce problème de classification multiclasses. L'objectif était d'évaluer plusieurs approches, allant des modèles simples aux modèles plus avancés, pour déterminer lesquels s'adaptent le mieux au dataset.

Les modèles testés sont les suivants :

- Régression logistique, qui sert souvent de modèle de base pour la classification.
- K-Nearest Neighbors (KNN), un modèle basé sur la distance.
- Arbre de décision, un modèle simple mais interprétable.
- Random Forest, un ensemble d'arbres permettant de réduire la variance.
- Gradient Boosting, un modèle d'ensemble séquentiel très performant.
- SVM (Support Vector Machine), adapté aux problèmes non linéaires.
- MLPClassifier (réseau de neurones), capable de capturer des relations complexes.

Tous les modèles ont été intégrés dans un Pipeline, incluant automatiquement le préprocesseur (StandardScaler + OneHotEncoder). Cela garantit une chaîne de traitement cohérente et évite les erreurs lors de la transformation des données.

Chaque modèle a été entraîné sur l'ensemble d'entraînement (60 %) et évalué sur l'ensemble de validation (20 %). Les métriques utilisées pour la comparaison étaient :

- accuracy,
- macro F1-score,
- ainsi que le classification report pour analyser les performances sur chaque classe.

Les résultats montrent que les modèles d'ensemble (Random Forest et Gradient Boosting) et le réseau de neurones MLP étaient clairement supérieurs aux autres, avec des scores de validation nettement plus élevés.

[25]:

	Model	Validation_Accuracy	Validation_F1
4	GradientBoosting	0.947867	0.946704
3	RandomForest	0.936019	0.934819
6	MLP	0.936019	0.933772
2	DecisionTree	0.926540	0.924487
5	SVC	0.921801	0.919243
0	LogisticRegression	0.848341	0.839799
1	KNN	0.836493	0.817919

Les modèles plus simples comme la régression logistique et KNN ont obtenu des performances correctes, mais moins adaptées à la complexité du problème.

Cette première comparaison a permis d'identifier trois modèles particulièrement prometteurs :

- Gradient Boosting,
- Random Forest,
- MLPClassifier.

Ces trois modèles ont ensuite été sélectionnés pour l'étape d'optimisation approfondie des hyperparamètres.

5. Optimisation des hyperparamètres

Après avoir identifié les trois modèles les plus prometteurs lors de la phase de comparaison initiale (Gradient Boosting, Random Forest et MLPClassifier), j'ai procédé à une optimisation des hyperparamètres afin d'améliorer leurs performances. Cette étape est essentielle, car les modèles d'ensemble et les réseaux de neurones sont très sensibles aux valeurs de leurs paramètres, et de bons réglages peuvent faire une différence importante sur les résultats.

Pour réaliser cette optimisation, j'ai utilisé la méthode GridSearchCV, qui teste automatiquement plusieurs combinaisons d'hyperparamètres en utilisant une validation croisée. Cette approche permet de sélectionner les paramètres donnant les meilleures performances sur l'ensemble de validation, tout en évitant de surajuster le modèle aux données d'entraînement.

Les grilles d'hyperparamètres testées étaient adaptées aux modèles :

- Gradient Boosting : nombre d'estimateurs, taux d'apprentissage, profondeur maximale des arbres.
- Random Forest : nombre d'arbres, profondeur maximale, min_samples_split.
- MLPClassifier : taille des couches cachées, fonction d'activation, taux d'apprentissage initial et coefficient de régularisation.

Chaque combinaison a été évaluée avec une validation croisée à trois plis (cv=3) afin de garantir une estimation stable de la performance. Une fois les recherches terminées, les meilleurs modèles ont été récupérés à l'aide de best_estimator_.

Les résultats ont montré une amélioration notable pour les trois modèles, avec des gains de performance sur l'ensemble de validation.

[31]:

	Model	Val_Accuracy	Val_F1
2	MLP_opt	0.966825	0.965399
0	GB_opt	0.950237	0.949260
1	RF_opt	0.931280	0.930220

Le modèle Gradient Boosting a obtenu de très bons résultats, mais c'est finalement le **réseau de neurones MLP** optimisé qui a atteint les scores les plus élevés. Cette étape a donc permis de sélectionner un modèle final performant tout en réduisant les risques de surapprentissage.

6. Évaluation finale

Après l'optimisation des hyperparamètres, les trois meilleurs modèles ont été évalués sur l'ensemble de test, qui représente 20 % du dataset et n'a jamais été utilisé pendant l'entraînement ni pendant la validation. Cette étape permet d'estimer la capacité réelle de généralisation de chaque modèle et de vérifier qu'il ne s'est pas surajusté aux données de validation.

Les performances ont été mesurées à l'aide de deux métriques principales :

- l'accuracy, qui indique la proportion de prédictions correctes ;
- le macro F1-score, qui évalue la performance moyenne sur toutes les classes, même en cas de léger déséquilibre.

Les résultats obtenus sur l'ensemble de test montrent que les trois modèles optimisés ont conservé de bonnes performances. Le modèle MLPClassifier optimisé (MLP_opt) s'est clairement démarqué comme le meilleur, avec une accuracy d'environ 97 % et un macro F1-score d'environ 97 %. Ces performances dépassent légèrement celles obtenues sur l'ensemble de validation, ce qui confirme que le modèle généralise très bien aux nouvelles données.

Le modèle Gradient Boosting optimisé (GB_opt) arrive en deuxième position, avec une accuracy d'environ 93,6 % et un F1-score similaire. Enfin, le Random Forest optimisé (RF_opt) obtient également de bons résultats, avec une accuracy proche de 93 %. Bien que ces deux modèles soient performants, leurs résultats sont légèrement inférieurs à ceux du réseau de neurones.

L'écart faible entre les scores de validation et de test pour les trois modèles indique qu'il n'y a pas eu de surapprentissage important. Cela confirme que les transformations appliquées, la division du dataset et l'optimisation des hyperparamètres ont permis d'obtenir des modèles stables et fiables.

Au final, le MLP optimisé est retenu comme modèle final du projet, car il offre les meilleures performances globales et la meilleure capacité de généralisation.

7. Analyse critique des résultats

L'analyse des résultats après l'optimisation montre que certains modèles se sont nettement démarqués, tandis que d'autres ont atteint leurs limites face à la complexité du problème. Dans l'ensemble, les performances sont élevées, mais chaque modèle n'a pas appris de la même façon, ce qui permet de mieux comprendre leurs forces et leurs faiblesses.

Tout d'abord, avant l'optimisation, les modèles les plus performants étaient déjà le Gradient Boosting, le Random Forest et le réseau de neurones (MLPClassifier). Après l'optimisation des hyperparamètres, ces trois modèles ont encore amélioré leurs résultats, ce qui montre que le choix des paramètres a un impact réel sur leur performance.

C'est finalement le MLP optimisé qui a obtenu les meilleurs résultats, autant sur l'ensemble de validation que sur l'ensemble de test. Ses scores élevés (environ 97 % d'accuracy sur le test) indiquent que le modèle arrive très bien à capturer les relations entre les différentes variables du dataset. Le réseau de neurones est particulièrement efficace lorsqu'il y a un mélange de variables numériques et catégorielles, et cela semble être le cas ici. De plus, l'utilisation du scaling a certainement aidé le MLP à converger correctement.

Le Gradient Boosting arrive en deuxième position. Il reste un modèle très performant, mais légèrement en dessous du MLP. Il s'adapte bien aux interactions complexes, mais il peut être un peu plus sensible aux hyperparamètres comme le learning rate ou la profondeur des arbres.

Quant au Random Forest, il présente de bons résultats, mais il semble légèrement moins capable de capturer certains détails par rapport aux deux autres modèles. C'est un modèle robuste, mais il n'offre pas toujours le même niveau de finesse que le Gradient Boosting ou un réseau de neurones.

Un point important est que les trois modèles ont montré une bonne capacité de généralisation : les scores sur l'ensemble de test sont proches de ceux obtenus sur l'ensemble de validation. Cela signifie qu'il n'y a pas de surapprentissage important et que les données ont été suffisamment bien préparées pour éviter ce problème.

Enfin, le dataset étant propre, complet, sans valeurs manquantes et avec une variable cible bien équilibrée, les modèles ont pu apprendre dans de bonnes conditions. L'une des limites du projet est que certaines variables peuvent être difficiles à interpréter dans le contexte d'un réseau de neurones, ou qu'il aurait été intéressant d'essayer d'autres modèles d'ensemble comme XGBoost ou LightGBM pour comparer davantage. Néanmoins, les résultats obtenus sont solides et cohérents.

8. Conclusion

Ce projet m'a demandé beaucoup de temps et d'efforts, mais il m'a aussi permis de comprendre concrètement comment mettre en place un système complet d'apprentissage machine. Le travail était assez long parce qu'il fallait passer par plusieurs étapes : exploration du dataset, préparation, choix des modèles, optimisation et finalement l'évaluation. Même si chacune des étapes était claire en théorie, les appliquer dans un vrai projet m'a demandé plus de pratique que ce à quoi je m'attendais au début.

L'exploration des données a été assez simple puisque le dataset était propre, mais la suite était plus difficile. Par exemple, comprendre quels prétraitements appliquer, comment structurer le pipeline, ou encore comment encoder correctement les variables m'a demandé plusieurs essais. L'optimisation des hyperparamètres a aussi été une étape plus compliquée que prévu, surtout pour le réseau de neurones, parce que certains paramètres avaient un impact énorme sur les résultats. Il fallait trouver un bon compromis entre performance, temps d'entraînement et stabilité.

Au final, le modèle MLP optimisé s'est révélé être le plus performant, avec une excellente précision sur les données de test. Malgré les difficultés, cela confirme que les efforts investis dans l'optimisation ont eu un impact réel. Le Gradient Boosting et le Random

Forest ont aussi bien fonctionné, ce qui montre que plusieurs modèles étaient adaptés au problème.

Comme limite principale, je dirais que certains modèles, notamment les réseaux de neurones, sont plus difficiles à comprendre et à interpréter. Une autre limite est que le projet demande beaucoup de pratique pour être vraiment à l'aise avec tous les outils utilisés. Dans le futur, j'aimerais explorer des modèles plus avancés comme XGBoost et aussi mieux comprendre l'importance des variables pour interpréter les résultats.

Globalement, ce projet m'a vraiment aidé à comprendre comment construire un pipeline complet et à appliquer les concepts vus en cours dans un cas concret. Même si c'était parfois difficile et que certaines étapes demandaient beaucoup de patience, j'ai pu apprendre beaucoup sur la démarche et sur la pratique de l'apprentissage machine.

9. Références

Machine Learnia (Youtube) : (888) Machine Learnia - YouTube

Scikit_Learn:

train_test_split — scikit-learn 1.7.2 documentation

7.3. Preprocessing data — scikit-learn 1.7.2 documentation

10. Annexes (facultatif)

S’il y a lieu :

- Graphiques supplémentaires
- Autres tableaux de résultats