PEUT-ON PRÉDIRE SI LES REVENUS D'UN INDIVIDU EXCÈDENT 50K DOLLARS PAR AN ÉTANT DONNÉ SON PROFIL?

EL RAFEI Jana - HASSAN Yousra - MA Haiyang
Rapport - Printemps 2024

RÉSUMÉ

L'objectif du projet de cette UV AI28 - Machine Learning, enseignée à l'Université de Technologie de Compiègne (UTC), est de mettre en pratique les méthodes d'apprentissage supervisé abordées en cours, en les appliquant à un jeu de données réel. Le jeu de données sélectionné, intitulé Adult, accessible via ce lien, est un ensemble de données tiré par Barry Becker en 1994 de la base de données Census. L'étude vise à déterminer les indicateurs permettant de prédire les revenus d'un individu en fonction de son profil.

1 INTRODUCTION

Ce projet de prédiction est destiné à analyser et interpréter le profil d'une multitude d'individus. L'objectif est de mettre en évidence les facteurs permettant de prédire si les revenus d'un individu sont supérieurs à 50K/an. Pour ce faire, nous entreprendrons dans un premier temps une analyse exploratoire des données (EDA) pour pouvoir les comprendre et avoir une idée globale de l'échantillon étudié. Nous utiliserons notamment différentes solutions de visualisations. Nous continuerons en présentant les différentes méthodes de traitement de données réalisées qui nous donne accès à l'information utile en se débarrassant du bruit. Finalement, en tirant parti des algorithmes d'apprentissage automatique (ML), nous construirons des modèles prédictifs de classification binaire permettant d'estimer les revenus d'un individu en fonction de son profil.

2 EXPLORATION DES DONNÉES

Avant de s'atteler à la prédiction des revenus pour un individu X, il nous a semblé fondamental de mieux comprendre le jeu de données. L'analyse exploratoire est disponible dans le fichier **notebooks/data_exploration.ipynb**. Celle-ci est très complète et seuls les éléments essentiels à retenir pour la suite du rapport sont expliqués ci-dessous.

Ce dataset est une matrice (48 842 x 15). Les colonnes, exceptées la dernière, représentent les variables et composent le profil d'un individu; elles seront appelées par leur dénomination anglaise, 'features'. La dernière est la variable cible, appelée 'target'. Les noms des colonnes et leur signification sont consignées ci-dessous :

Variable	Description
age	Age
workclass	Catégorie socio-professionnelle
fnlgwt	Poids d'instance, caractéristiques démographiques similaires - correspond à l'appartenance à un groupe ethnique précis
education	Type d'études réalisées
education-num	Nombre d'années d'études réalisées
marital-status	Statut marital
occupation	Emploi actuel
relationship	Relations
race	Groupe ethnique (différent de fnlgwt qui entre dans le détail de la région. Ici, on ne souhaite savoir que si l'individu est un 'blanc', 'noir, 'amérindien', .. de manière très générale).
sex	Genre
capital-gain	Gains en capital
capital-loss	Pertes en capital
hours-per-week	Nombre d'heures travaillées par semaine
native-country	Pays d'origine
income	Revenus. 'Income' est la variable cible, elle est catégorielle et est composée de deux valeurs '>50K' ou '<=50K'.

2.1 Présentation des données

2.1.1 Target

Nous commençons notre analyse par l'encodage de la variable cible. En effet, prédire qu'un individu perçoit plus ou moins que 50K par an revient à un problème de classification binaire. On choisit alors d'encoder cette colonne qui prendra une valeur booléenne. Ainsi, si un individu perçoit plus de 50K, la valeur codée est 1. La colonne, initialement de type object et appelée 'income', devient alors de type int64 et intitulée '>50K'.
Nous observons immédiatement que les classes sont déséquilibrées: seules 25% des observations concernent les revenus '>50K'.

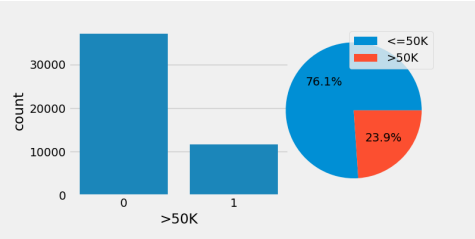

Figure 1: Distributions des classes dans le dataset

2.1.2 Features

Notre dataset est composé principalement de variables catégorielles, mais aussi de variables continues :

- Les variables catégorielles sont de type 'object', et composent 57% des variables (8/14).
- Les variables continues sont de type 'int64', et composent 43% des variables (6/14)

A cette étape, nous avions pensé à faire un encodage one hot pour les variables catégorielles mais dans ce cas, le nombre de colonnes passent de 14 à 108 à cause de la multitude de catégories pour chaque variable.

Les graphiques ci-dessous représentent la distribution des variables catégorielles puis numériques, par classes pour avoir une idée plus concrète des données disponibles.

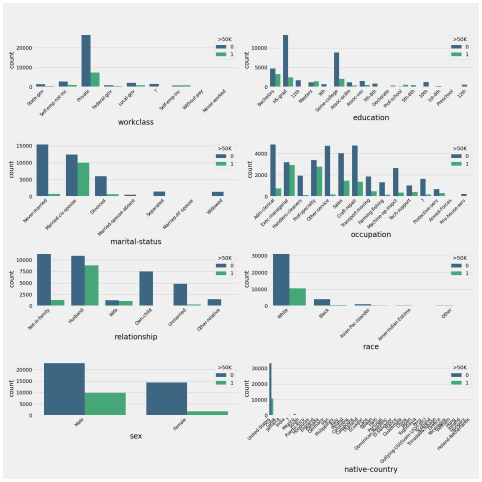

Figure 2: Distributions des variables catégorielles par classe

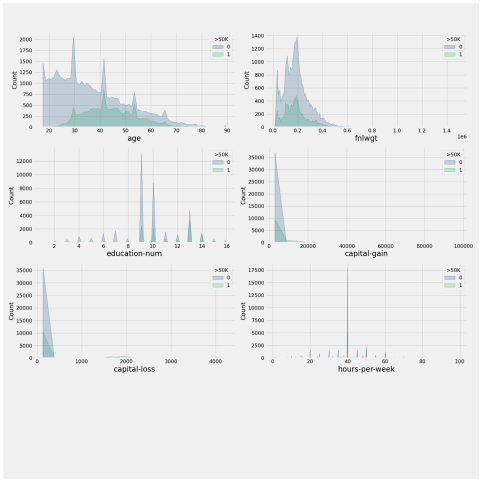

Figure 3: Distributions des variables numériques par classe

Nous observons que les données catégorielles présentent des pics significatifs qui témoignent de l'homogénéité du profil des individus. En effet, 90% des individus sont américains d'origine, 85% sont blancs, 67% sont des hommes, 70% d'une catégorie socio-professionnelle privée, .. Ces chiffres mettent en évidence le caractère biaisé des données issues d'une certaine catégorie de profil, ce qui pourra largement biaiser le modèle (erreurs de prédictions sur d'autres profils).

Les données numériques présentent également des données exploitables. Les données *Age* sont très irrégulières et s'étalent sur un spectre large même si la plupart des individus gagnant plus de 50K/an sont dans la tranche d'âge de 30 à 60 ans. Les données *education_num* fournissent des informations pertinentes sur le profil des individus et plusieurs pics sont mis en évidence à 9, 10 et 13 années d'études. Le reste des graphiques suivent une loi plutôt normale. Enfin, la feature 'fnlwgt' semble avoir une très faible influence sur la cible, car les distributions des deux groupes sont très similaires.

2.2 Pré-traitement avant visualisation

A la réception du dataset, 48 lignes sont des duplicatas. On les supprimera lors du prétraitement mais également dans ce notebook pour mieux comprendre les données. De plus, trois variables (catégorielles) ont des valeurs manquantes avec des taux inférieurs à 2% . On décide alors de les garder pour conserver l'intégrité des données.

2.3 Valeurs aberrantes

Les valeurs aberrantes des variables numériques sont représentées sur le graphique ci-dessous :

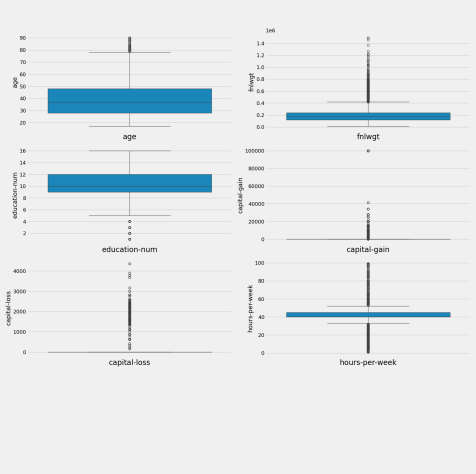

Figure 4: Distributions des variables catégorielles par classe

Nous avons choisi d'être prudents avec les valeurs aberrantes de ce dataset car tout profil différent du profil majoritaire évoqué plus tôt peut être considéré comme atypique.
Nous avons donc choisi de ne supprimer que les valeurs excédant 40000 pour la variable *capital-gain* et 4000 *capital_loss*.

Les outliers des variables catégorielles seront évoquées dans la partie prétraitement-regroupement des variables.

2.4 Corrélations

Les données sont très peu corrélées entre elles, comme en témoignent les deux graphiques:

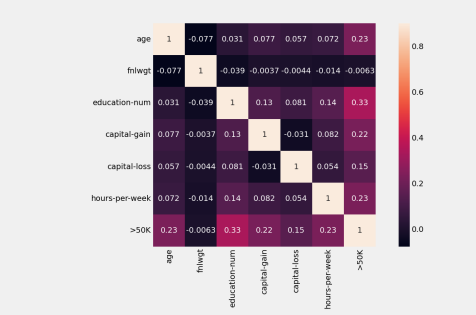

Figure 5: Matrice de corrélation entre les variables catégorielles

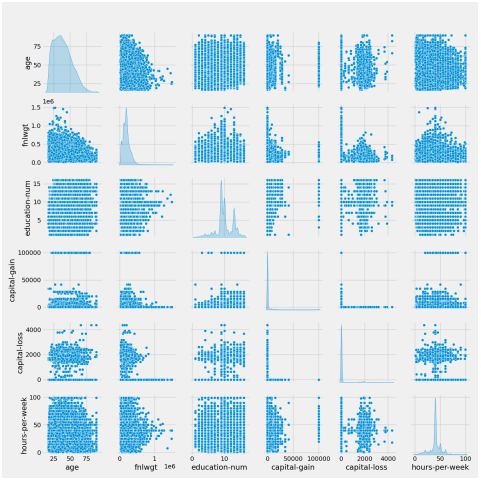

Figure 6: Distributions de la corrélation entre les variables numériques

Finalement, la visualisation des différents aspects de ces données nous a permis d'établir une stratégie de pré-traitement développée dans la partie suivante.

3 Traitement des données

Après avoir visualiser les données, nous connaissons à présent l'information utile dans ce dataset. Nous procédons donc séquentiellement pour éliminer le bruit avant de passer à la prédiction. Tous les modèles développés possèdent cette étape dans leur notebook. Une fonction *preprocess* issue du fichier **src/data_preprocessing.py** est appelée, et permet le nettoyage des données. Le préprocessing continue ensuite avec l'imputation des valeurs manquantes et l'encodage des variables.

3.1 Nettoyage des données

La première phase de prétraitement est composée de plusieurs étapes distinctes :

- **Encodage de la variable cible:** Premièrement, on s'attelle à encoder la target pour qu'elles puissent prendre des valeurs binaires. Cela simplifiera grandement la prédiction et l'analyse des résultats par la suite. De plus, la variable était initialement codée avec des chaines de caractères et était inscrite dans le dataset sous plusieurs formes pour une même classe. Il était donc impératif de normaliser la colonne.

- **Suppression de variables inutiles:** Nous avons fait le choix de supprimer la colonne *education* car elle était équivalente à la colonne *education-num*. Ainsi, *education-num* représentait la colonne supprimée encodée en ordinal. Elle n'était donc pas nécessaire. Nous avons également souhaités supprimer la variable *fnlwgt* car elle n'apportait pas d'informations pertinentes. Cependant, après sa suppression, nous nous sommes rendus compte qu'elle engendrait +6000 duplicatas. Nous en avons conclu que cette variable apportait une information importante qui permet de distinguer des profils entre eux. Nous allons donc la garder même si l'information encodée est un nombre qui n'est pas exploitable (en effet, il représente l'encodage d'un groupe ethnique, sans qu'on puisse savoir lequel).

- **Suppression des valeurs aberrantes:** Comme nous l'avons vu dans les diagrammes en boîtes des variables numériques, les variables *capital gain* et *capital-loss* ont des seuils (respectivement 40000 et 4000) à partir desquels les valeurs sont considérées comme aberrantes. Ces observations ont été supprimées.

- **Regroupement des catégories:** La plupart des variables sont catégorielles et proposent une multitude de catégorie. Cependant, comme expliqué dans l'analyse exploratoire, beaucoup de catégories sont présentes mais sont peu utilisées dans le dataset. Il était donc nécessaire pour mieux comprendre le jeu de données de regrouper des catégories entre elles. Ainsi, la variable *relationship* présente plusieurs catégories dont 'Husband' et 'Wife'. Il n'est pas imprudent de regrouper ces catégories dans une nouvelle appelée 'Married'. Plusieurs regroupements de ce style ont été réalisés. De la même manière manière, nous avons regroupé dans une catégorie 'Other', tous les pays d'origine qui sont inscrits dans moins de 200 observations. En effet, l'AED a mis en évidence que 89,7% des individus ont comme pays d'origine les Etats-Unis mais qu'il y a un total de 42 pays répertoriés. Il est donc nécessaire de minimiser le nombre de valeurs catégorielles inutiles/ peu représentatives pour simplifier l'encodage par la suite. Ainsi, sans ce regroupement, l'encodage one hot nous aurait fait passer de 14 à 108 features, mais avec, on obtient 47 features. Ce regroupement nous a permis d'améliorer significativement les performances. Dernier point, les valeurs '?' dans plusieurs colonnes du dataset ont été renommées en 'Not referenced'.

- **Suppression des duplicatas:** L'étape d'après a été d'éliminer les duplications, qu'elles soient des erreurs de saisies où les conséquence des regroupements. Environ 300 duplicatas ont été supprimés.

Cette phase de nettoyage de données s'est avérée nécessaire pour pouvoir avoir des résultats exploitables. Les nouvelles données sont alors présentées dans la partie (1) de l'Appendix.

3.2 Imputation des valeurs manquantes

La suite du pré-traitement consiste en l'imputation des valeurs manquantes. Seules deux variables catégorielles voient certaines de leur valeurs manquer: *workclass* et *occupation*. Nous remplaçons alors les valeurs manquantes par la valeur la plus fréquente dans le dataset.

3.3 Normalisation des données - encodage des variables

La dernière phase de ce pré-traitement est l'encodage des variables. Les features sont regroupées par variables numériques et catégorielles. Les variables numériques sont encodées avec un *StandardScaler* tandis que les variables catégorielles sont encodées avec *One Hot Encoder*. Ce choix a été réalisé pour lever le problème de la priorisation de certaines variables lors d'un encodage ordinal pour certains modèles.

4 MODÈLES DE PRÉDICTION - APPRENTISSAGE SUPERVISÉ

Dans cette partie, nous allons explorer différents modèles que nous avons choisi d'implémenter dans dans notre projet.

Nous nous intéressons principalement à la métrique 'f1_score'. Ce choix repose sur le fait que nous ne privilégions pas une classe par rapport à une autre. En effet, prédire faussement qu'une classe est négative à le même impact que prédire qu'une classe est positive. Le but est donc de maximiser la prédiction et le rappel. Le problème revient donc à maximiser le score F1.

4.1 RÉGRESSION LOGISTIQUE

Le notebook `models_RegLog.ipynb` traite les différents types de régression logistique. Nous nous intéressons dans u premier temps à la regression logistique parcequ'elle est particulièrement adaptée à ce type de problème de classification binaire. Elle permet de modéliser la relation entre les variables indépendantes (telles que l'âge, le niveau d'éducation, l'état matrimonial, etc.) et la variable dépendante (niveau de revenu) en estimant les coefficients qui maximisent la vraisemblance des observations. De plus, elle fournit une interprétation claire des coefficients, facilitant la compréhension de l'impact de chaque variable sur la probabilité de l'événement étudié.

Nous testons dans un premier temps la régression logistique simple afin d'identifier la performance du modèle et sa capacité de généralisation.

Nous obtiendrons les résultats suivants :

	precision	recall	f1-score	support
0	0.88	0.93	0.90	7422
1	0.72	0.57	0.64	2287

Nous remarquons que le f1-score de la classe 1 est relativement bas par rapport à celui de la classe 0. Cela peut être dû à un jeu de données déséquilibré. Nous effectuons alors un sur-échantillonnage de la classe minoritaire (classe 1) en utilisant la technique SMOTE.

Nous obtiendrons donc les résultats suivants :

	precision	recall	f1-score	support
0	0.93	0.80	0.86	7422
1	0.56	0.82	0.66	2287

Nous remarquons une amélioration relative de f1-score pour la classe 1. Cependant, une recule relative pour le f1-score de la classe 0.

Nous affichons ensuite la courbe d'apprentissage.

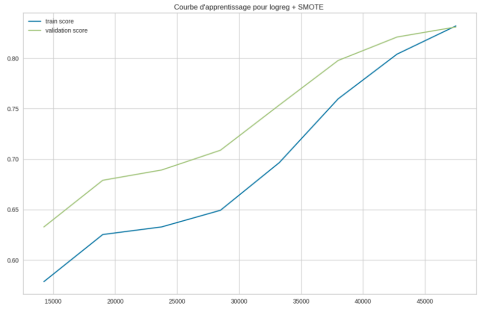

Figure 7: Courbe d'apprentissage du modèle de régression logistique

A partir du graphe, nous pouvons déduire que le modèle est en sur-apprentissage pour les différentes tailles de jeux de données. Cependant, l'écart entre les deux courbes diminue avec l'augmentation de la taille de l'échantillon et elles se chevauchent à la fin. Cela nous permet d'espérer un ajustement du modèle si nous avions plus de données.
Afin de régler ce sur-apprentissage, nous ajoutons une régularisation de modèle en utilisant les régressions logistiques Lasso, Ridge et ElasticNet.

Termes de pénalisation :

Pour améliorer la capacité de généralisation de notre modèle, nous utilisons différents termes de pénalisation dans les régressions logistiques. Voici les termes de pénalisation pour chaque type de régression :

Ridge (L2) Régularisation :

$$J(\beta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\beta(x_i)) + (1-y_i)\log(1-h_\beta(x_i))] + \lambda\sum_{j=1}^{p}\beta_j^2$$

A revoir (ajout courbe de validation..) !!

Lasso (L1) Régularisation :

$$J(\beta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\beta(x_i)) + (1-y_i)\log(1-h_\beta(x_i))] + \lambda\sum_{j=1}^{p}|\beta_j|$$

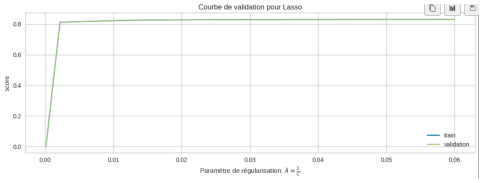

Figure 8: Courbe d'apprentissage du modèle de régression logistique

D'après la courbe de validation pour la regression Lasso, Le score augmente très lègerement pour les deux dataset à partir de ≈ 0.0025 de lambda . Le modèle Lasso est ajusté (pas de sur-apprentissage ni de sous-apprentissage) à partir de cette valeur.
Le meilleur paramètre trouvé par la technique Grid Search est : C = 1.0 pour 100 itérations maximale.

ElasticNet Régularisation :

$$J(\beta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\beta(x_i)) + (1-y_i)\log(1-h_\beta(x_i))] + \lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}\beta_j^2$$

A revoir (ajout courbe de validation..) !!

Résultats : Nous obtiendrons les résultats suivants pour les trois modèles de régressions régularisées :

	precision	recall	f1-score	support
0	0.88	0.93	0.90	7422
1	0.72	0.57	0.64	2287

Aucune amélioration de performance n'est obtenue malgré la régularisation du modèle. Nous essayerons alors par la suite les autres modèles d'apprentissage.

4.2 Machines à vecteurs de support - SVM

4.3 Algorithme des K- plus proche voisins - KNN

4.4 Arbres de décision et forêts aléatoires

Le notebook `models_random_forests.ipynb` traite les algorithmes d'arbres de décision et des forêts aléatoires.

4.4.1 Arbres de décision

Les arbres de décision offrent un bon équilibre entre simplicité, interprétabilité et performance, ce qui en fait un choix judicieux pour une tâche de classification binaire comme prédire si un individu gagne plus ou moins de 50K. Ils sont particulièrement utiles lorsque l'objectif est de comprendre les

facteurs influençant les décisions et d'obtenir des résultats rapidement sans un prétraitement lourd des données. Ayant émis des réserves concernant les valeurs atypiques dans le dataset, il nous a semblé important d'entraîner un modèle peut sujet à l'impact du prétraitement.

Pour entraîner ce modèle, nous avons commencé par entraîner un modèle simple d'arbre de décision. Les résultats sont consignés ci-dessous :

	precision	recall	f1-score	support
0	0.88	0.88	0.88	7422
1	0.60	0.60	0.60	2287

Comme pour les modèles précédents, nous tentons alors d'augmenter les performances en suréchantillonant la classe minoritaire.

Les résultats sont les suivants :

	precision	recall	f1-score	support
0	0.88	0.86	0.87	7422
1	0.57	0.63	0.60	2287

A première vue, les résultats semblent très légèrement se dégrader mais la courbe d'apprentissage nous montre que le modèle suréchantilloné offre de meilleurs perspectives car il permet de diminuer la variance (malgré un biais plus faible).

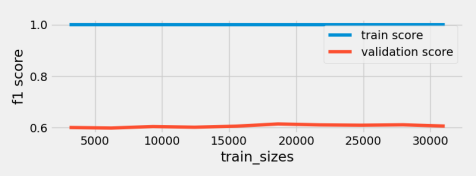

Figure 9: Courbe d'apprentissage de l'arbre de décision entraîné

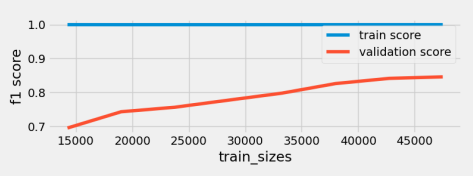

Figure 10: Courbe d'apprentissage de l'arbre de décision entraîné après suréchantillonage de la classe minoritaire

Nous décidons alors d'entraîner un modèle optimisé sur les données suréchantillonnées. Pour l'optimisation des hyper-paramètres, nous utilisons également GridSearchCV. Pour fixer l'intervalle des hyper-paramètres, nous procédons de manière itérative. Nous initialisons l'intervalle avec un spectre large de valeurs et en fonction du résultat, nous recentrons cet intervalle. Ainsi par exemple, nous avons commencé avec param_grid = 'max_depth': [None, 10, 20, 30, 40], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'max_features': [None, 'sqrt', 'log2'] . Les meilleurs paramètres déterminés ont été : param_grid = 'max_depth': [16], 'min_samples_split': [2], 'min_samples_leaf': [6], 'max_features': [None] .

Ces paramètres ont fourni les résultats suivants, qui sont bien meilleurs que ceux d'avant mais toujours pas exceptionnels.

	precision	recall	f1-score	support
0	0.91	0.84	0.88	7422
1	0.59	0.74	0.66	2287

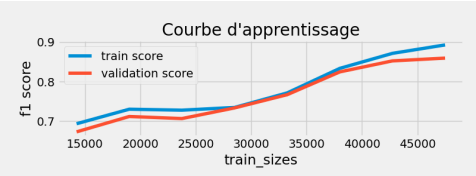

Figure 11: Courbe d'apprentissage de l'arbre de décision optimisé

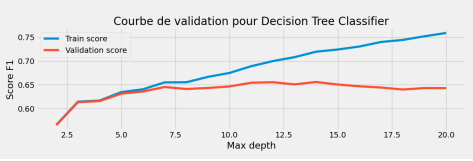

Figure 12: Courbe de validation de l'arbre de décision optimisé

La courbe d'apprentissage de l'ensemble test se mêle à celle de la validation et s'arrête à 37000 observations environ. Cela est cohérent avec notre hypothèse de départ. Les données sont très homogènes et lorsqu'il s'agit de prendre en compte d'autres données (d'augmenter le nombre), l'écart se creuse. Nous somme sur un cas typique de sur-apprentissage.

De la même manière, la courbe de validation stagne aux alentours de 0.6 comme les autres modèles. Nous allons donc tester un autre modèle, les forêts aléatoires pour tenter d'améliorer les performances.

Remarque, nous aurions pu proposer directement à l'algorithme GridSearchCV un intervalle plus fourni en valeurs mais les ordinateurs des membres du groupe peinaient à aboutir à un résultat avec la méthode précédente. Il n'était donc pas envisageable d'augmenter le nombre de valeurs de cet intervalle et donc le nombre de combinaisons.

4.4.2 Forêts aléatoires

Les arbres de décision ont tendance à sur-ajuster les données d'entraînement, surtout s'ils sont profonds (dans notre cas, max_depth = 16). Cela signifie qu'ils peuvent capturer le bruit dans les données et ne pas bien généraliser sur les données de test. En combinant plusieurs arbres de décision (souvent des centaines), chacun formé sur un sous-échantillon des données avec des caractéristiques sélectionnées aléatoirement, les forêts aléatoires réduisent le risque de sur-apprentissage. Les résultats sont moyennés sur tous les arbres, ce qui tend à lisser les prédictions et à améliorer la généralisation.

A l'instar de la méthodologie utilisée pour les arbres de décisions, nous avons entraîné un modèle sur les données sur-échantillonées puis nous avons optimisé les hyper-paramètres. Les résultats sont consignés ci-dessous:

Random Forest -données suréchantillonnées :

	precision	recall	f1-score	support
0	0.89	0.89	0.89	7422
1	0.65	0.66	0.66	2287

Random Forest -données suréchantillonnées et hyperparamètres optimisés:

	precision	recall	f1-score	support
0	0.90	0.89	0.89	7422
1	0.65	0.67	0.66	2287

Les hyper-paramètres choisis par l'algorithme GridSearchCV sont : 'max_depth': 50, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100.

La précision et le rappel ont augmenté mais légèrement malgré un entraînement sur plusieurs heures (plus de 5 heures). Nous continuons donc à chercher un modèle performant pouvant entraîner de meilleurs résultats.

4.5 Algorithmes d'ensemble

Dans cette partie, nous nous intéressons aux algorithmes d'ensemble. Nous souhaitons combiner les prédictions de plusieurs modèles de base afin d'obtenir une performance supérieure par rapport à l'utilisation d'un seul modèle. En combinant plusieurs modèles de base, les modèles peuvent souvent améliorer la précision des prédictions. Les erreurs de certains modèles peuvent être compensées par les prédictions correctes d'autres modèles, ce qui conduit à une meilleure performance globale.

Dans le domaine du Machine Learning, plusieurs modèles d'algorithmes d'ensemble sont utilisés, notamment : Adaboost, XGBoost et Stacking. Afin de déterminer lequel de ces trois techniques est le plus adapté à notre sujet, nous avons entrepris une démarche de comparaison de leurs performances.

Dans le notebook `models_comparaison_ensembles.ipynb`, nous faisons la comparaison entre les modèles précités.

Nous avons initialisé les modèles de manière à garantir une égalité de conditions entre eux, notamment en fixant `n_estimators` à 100 pour Adaboost et XGBoost, et en utilisant une validation croisée à 5 plis pour le modèle Stacking.

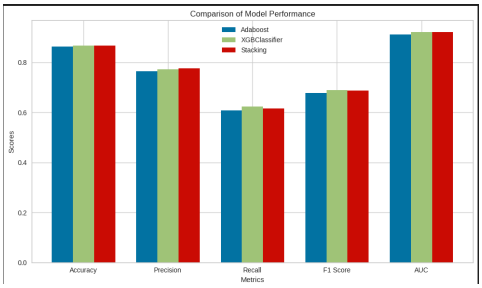

Figure 13: Courbe d'apprentissage du modèle de régression logistique

Les résultats de notre comparaison, illustrés par l'histogramme ci-dessus, montrent les performances des trois techniques selon différentes métriques : Accuracy, Precision, Recall, F1 Score et AUC. Nous observons que les trois modèles présentent des performances comparables en termes de précision et de rappel, avec des scores légèrement supérieurs pour XGBoost et Stacking en AUC, ce qui indique une meilleure capacité à distinguer entre les classes.

Après une analyse comparative rigoureuse des performances des modèles Adaboost, XGBoost et Stacking, notre choix s'est porté sur XGBoost pour la recherche d'hyperparamètres. Cette décision est motivée par plusieurs facteurs clés :

- **Performances supérieures :** XGBoost a démontré des scores élevés sur toutes les métriques de performance, en particulier en termes de Recall.

- **Efficacité et flexibilité :** XGBoost est reconnu pour sa capacité à gérer efficacement les données déséquilibrées et à s'adapter à différents types de problèmes grâce à ses nombreux paramètres réglables.

Cependant, il est important de noter que cette démarche n'est pas exhaustive. Nous pourrions également tirer parti des modèles Adaboost et Stacking pour obtenir un modèle plus performant en optimisant leurs hyperparamètres. Cette approche permettrait de choisir le modèle le plus adapté en fonction des meilleures performances obtenues.

Table 1: Résultats des modèles Random Forest

Modèle	Precision	Recall	F1-score
Régression Logistique			
Classe 0	0.93	0.80	0.86
Classe 1	0.56	0.82	0.66
Régression Logistique avec pénalisation			
Classe 0	0.88	0.93	0.90
Classe 1	0.72	0.57	0.64
SVM			
Classe 0	0.	0.	0.
Classe 1	0.	0.	0.
KNN			
Classe 0	0.	0.	0.
Classe 1	0.	0.	0.
Arbres de décision			
Classe 0	0.88	0.88	0.88
Classe 1	0.60	0.60	0.60
Forêts aléatoires			
Classe 0	0.88	0.86	0.87
Classe 1	0.57	0.63	0.60

4.6 Application des modèles : résultats, visualisations et performances

5 Interprétations et discussions

6 Conclusion

References

A Appendix

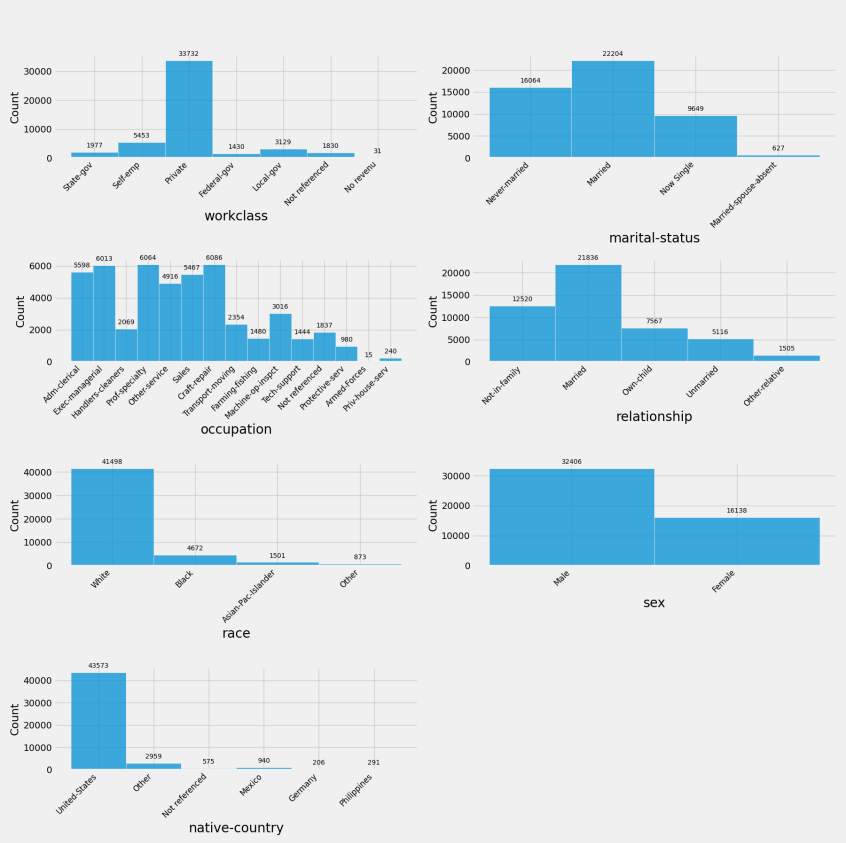

Figure 14: Distribution des variables catégorielles après nettoyage des données