PROJET DE MACHINE LEARNING
Partie 2 - Modélisation

Accidents routiers en France
- de 2005 à 2022 -

Emmanuel GAUTIER
Erika MÉRONVILLE
Rémi THINEY

Table des matières

1 Modélisation 1

1.1 Implémentation d'un premier modèle

Sur la base du fichier obtenu lors du premier préprocessing, il est procédé à une 1ère modélisation de type classique, à savoir une régression logistique. Le choix de ce modèle se justifie entre autres par les raisons suivantes :

— il est adapté à la nature binaire de la variable cible (accident grave ou non grave),

— il est capable de gérer les nombreuses variables catégorielles présentes dans le jeu de données,

— il offre une bonne base pour servir de référence à des modèles plus complexes, etc.

Le code créé met en place un pipeline complet pour l'entraînement et l'optimisation d'un modèle de régression logistique.

- Importation des librairies

Le code commence par importer les bibliothèques essentielles pour la manipulation des données (Numpy et Pandas), la visualisation (Matplotlib), le prétraitement (StandardScaler), la division des données (train_test_split), la modélisation (LogisticRegression), l'optimisation des hyperparamètres (GridSearchCV), et l'évaluation des performances (diverses métriques et validation croisée).

- Chargement et prétraitement des données

Par la suite, le code charge et prépare les données pour une analyse de machine learning. Il commence par importer un fichier CSV prétraité ('data.csv'), utilisant des tabulations comme séparateurs. Ensuite, il convertit toutes les colonnes en nombres entiers pour assurer une cohérence des types de données. Finalement, il sépare les données en deux parties : X, qui contient toutes les variables explicatives, et y, qui isole la variable cible 'grav_grave'. Cette dernière sera l'objet de la prédiction dans le modèle à venir pour indiquer la gravité des accidents.

- Standardisation des variables numériques

Egalement, le code effectue la standardisation des variables numériques du jeu de données, en suivant cette démarche :

— Une liste 'dummy_columns' est définie, contenant les noms des colonnes catégorielles qui ne doivent pas être standardisées.

— Une nouvelle liste 'columns_to_scale' est créée pour y inclure toutes les colonnes de X qui ne sont pas dans 'dummy_columns', identifiant ainsi les variables numériques à standardiser.

— Un objet StandardScaler est instancié pour normaliser les variables numériques en les centrant et réduisant (moyenne de 0 et écart-type de 1).

— Une copie de X est créée et nommée 'X_scaled' pour préserver les données originales.

— La méthode fit_transform() du StandardScaler est appliquée uniquement aux colonnes identifiées dans 'columns_to_scale'. Cette opération ajuste le scaler aux données puis transforme ces colonnes.

Cette étape de prétraitement permet de mettre toutes les variables numériques à la même échelle, pour améliorer les performances de nos futurs modèles.

- Création d'un ensemble d'entraînement et d'un ensemble test

Au moyen de la fonction 'train_test_split' de scikit-learn, les données sont divisées en ensembles d'entraînement (80%) et de test (20%). Une graine aléatoire 'random_state' est fixée à 12 pour permettre d'assurer la reproductibilité de la division.

- Configuration du modèle et préparation des hyperparamètres

Le code commence par initialiser un classificateur de régression logistique avec des paramètres de base et configure une stratégie de validation croisée à 3 plis. Une grille de recherche d'hyperparamètres est définit pour explorer méticuleusement différentes combinaisons de force de régularisation, types de pénalité et, dans le cas de la pénalité elasticnet, divers ratios entre les normes L1 et L2. Cette approche exhaustive vise à identifier la configuration optimale du modèle.

- Création d'un GridSearchCV personnalisé pour sauvegarder les résultats partiels

Le code définit ensuite une classe personnalisée 'GridSearchWithProgress' qui hérite de GridSearchCV. Cette classe modifie la méthode fit() pour avoir plus de contrôle sur le processus de recherche par grille. Voici ses principales caractéristiques :

— Elle calcule le nombre total de combinaisons de paramètres à tester.

— Pour chaque combinaison de paramètres, il enregistre le temps de début, configure l'estimateur avec ces paramètres, effectue une validation croisée et enregistre le temps de fin.

— Pour chaque itération, il stocke les résultats (paramètres et score moyen de test), incrémente le compteur de combinaisons complétées, sauvegarde les résultats partiels dans un fichier joblib et affiche les résultats partiels, y compris les paramètres, le score moyen et l'écart-type, et le temps d'exécution.

— Après avoir testé toutes les combinaisons, il identifie les meilleurs paramètres et le meilleur score, configure le meilleur estimateur avec les meilleurs paramètres et entraîne le meilleur estimateur sur l'ensemble des données.

La recherche d'hyperparamètres est lancée avec cette classe personnalisée.

Voici les 3 premiers résultats obtenus :

Résultat partiel 1/24 :
Paramètres : 'C' : 0.1, 'max_iter' : 1000, 'penalty' : 'l1'
Score moyen : 0.6736 (+/- 0.0028)
Temps de fit : 498.71 secondes

Résultat partiel 2/24 :
Paramètres : 'C' : 1, 'max_iter' : 1000, 'penalty' : 'l1'
Score moyen : 0.6737 (+/- 0.0025)
Temps de fit : 573.75 secondes

Résultat partiel 3/24 :
Paramètres : 'C' : 10, 'max_iter' : 1000, 'penalty' : 'l1'
Score moyen : 0.6736 (+/- 0.0025)
Temps de fit : 632.74 secondes

1.2 Évaluation des performances du modèle

A la suite des résultats obtenus, le code créé effectue l'évaluation finale et l'utilisation du meilleur modèle identifié par la recherche d'hyperparamètres :

— Il affiche les meilleurs hyperparamètres trouvés lors de la recherche par grille.

— Le meilleur score obtenu durant cette recherche est également affiché.

— Le meilleur modèle (celui avec les hyperparamètres optimaux) est extrait de l'objet GridSearchCV.

— Ce modèle optimal est ensuite utilisé pour faire des prédictions sur l'ensemble de test.

— Enfin, la précision (accuracy) du meilleur modèle sur l'ensemble de test est calculée et affichée.

Cette séquence permet de voir rapidement les résultats de l'optimisation des hyperparamètres et d'évaluer la performance du modèle optimisé sur des données non vues durant l'entraînement et l'optimisation.

Voici l'affichage correspondant :

Meilleurs paramètres : 'C' : 0.1, 'l1_ratio' : 0.2, 'max_iter' : 1000, 'penalty' : 'elasticnet'
Meilleur score : 0.6738685114087616
Accuracy du meilleur modèle : 0.6719810449379543

Le rapport de classification est édité pour permettre de noter les différentes métriques du meilleur modèle de prédiction retenu.

Voici l'affichage correspondant :

Rapport de classification du meilleur modèle :

	precision	recall	f1-score	support
1	0.70	0.82	0.76	10221
2	0.63	0.06	0.12	572
3	0.53	0.39	0.45	3631
4	0.67	0.66	0.66	10477
accuracy			0.67	24901
macro avg	0.63	0.48	0.50	24901
weighted avg	0.66	0.67	0.66	24901

1.3 Analyse des résultats

Du point de vue de la performance globale, l'accuracy de 67% indique que le modèle est plutôt satisfaisant, mais il y a encore une marge d'amélioration significative. Le modèle performe de manière cohérente entre la validation croisée (67.39%) et l'ensemble de test (67.20%), ce qui suggère une bonne généralisation.

Par contre, la performance par classe du modèle est clairement insatisfaisante : d'abord parce que les classes sont mal réparties entre elles (sur-représentation des classes 1 et 4), ensuite parce que la classe 2 relative aux accidents graves est très mal prédite (recall : 0.06 et F1-score : 0.12).

Au-delà du rééquilibrage nécessaire des classes, il est important de noter que la régression logistique est un modèle linéaire. De ce fait, elle peut ne pas capturer des relations complexes ou non linéaires de nos données.

Il apparaît intéressant d'essayer des algorithmes non linéaires comme les forêts aléatoires ou les gradient boosting machines pour aller plus loin. Aussi, essayer plusieurs modèles doit permettre d'améliorer les prédictions.

2 Modélisation 2

Après les résultats insuffisants du premier préprocessing, de nouvelles modélisations sont effectuées sur la base du deuxième préprocessing.

2.1 Essais de nombreux modèles de classification

- Codage pour réaliser des essais

Le code de ce notebook a été conçu pour permettre d'essayer facilement les modèles en faisant varier des paramètres, en affichant des métriques de performances des temps d'entraînement, des graphiques. Son codage a été fait pour permettre de réaliser facilement des changement et évolutions. La liste des modèles avec quelques informations est ainsi dans un dictionnaire très facile à compléter.

- Importation des librairies

D'abord, le code importe une vaste gamme de bibliothèques et modules Python essentiels pour l'analyse de données, l'apprentissage automatique et la visualisation. Il inclut des outils fondamentaux comme NumPy et Pandas pour la manipulation de données, des modules pour la gestion du temps et des fichiers, ainsi que des bibliothèques de visualisation comme Matplotlib et Seaborn, et Tabulate pour le formatage. Le code importe également de nombreux modules de scikit-learn, couvrant le prétraitement des données, la décomposition, la sélection de modèles, divers algorithmes de classification, des méthodes d'ensemble, et des métriques d'évaluation. Enfin, il intègre d'autres bibliothèques spécialisées comme LightGBM pour l'apprentissage automatique .

- Fonction d'affichage des messages

La fonction 'printlog()' est créée pour afficher à l'écran et enregistrer dans une chaîne de caractères 'logres' divers messages relatifs au processus d'entraînement et d'évaluation du modèle. Ces messages incluent les paramètres testés, les résultats de la recherche des meilleurs hyperparamètres, les scores de performance, la matrice de confusion, et d'autres informations pertinentes. Cette fonction permet ainsi de suivre en détail le déroulement de l'entraînement et de sauvegarder ces résultats pour une analyse ultérieure.

- Fonction de formatage de la durée

La fonction 'format_duree' sera utilisée spécifiquement pour formater le temps d'entraînement des modèles de manière plus lisible. Elle est appelée pour afficher la durée d'entraînement dans les résultats. Cette fonction permet de convertir le temps d'entraînement (initialement en secondes) en un format plus facile à lire, utilisant des heures, minutes et secondes selon la durée totale de l'entraînement.

- Chargement et préparation des données

Le code créé charge les données depuis un fichier CSV, puis sépare les variables explicatives (X) de la variable cible (y) nommée 'grav_grave'. Les noms des colonnes de (X) sont également enregistrés dans la variable 'feature_names' qui sera utile pour l'analyse de 'feature_importances'.

Cette étape prépare les données pour l'analyse et la modélisation en machine learning, en distinguant les informations utilisées pour faire des prédictions (X) de ce qui doit être prédit (y).

- Réduction de dimension et standardisation des données

Le jeu de données contient plus de 250 variables explicatives. Ce nombre étant très important, nous décidons d'appliquer une Analyse en Composantes Principales (PCA) pour réduire la dimensionnalité des données à 100 composantes. Après avoir appliqué la PCA, nous normalisons le jeu de données de dimension réduite.

Cette procédure vise à simplifier le jeu de données tout en préservant l'information essentielle pour la modélisation.

Deux graphiques nous ont aidé à choisir le nombre de composantes en affichant la variance expliquée par composante :

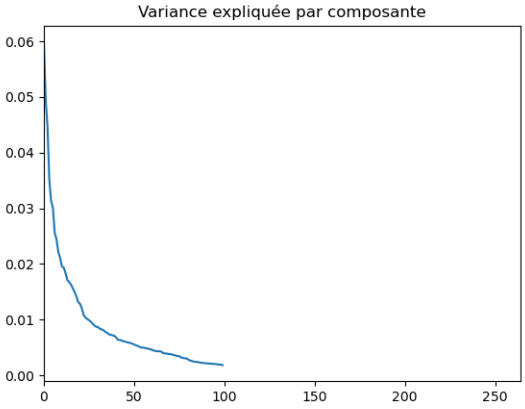


FIGURE 1 – Variance Expliquée

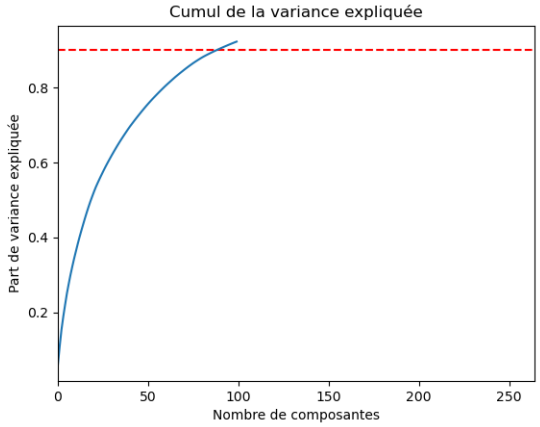


FIGURE 2 – Cumul de Variance Expliquée

Nous observons ici que 100 composantes principales suffisent à capturer plus de 90% de la variance totale des données originales. A notre avis, cela offre un bon compromis entre la réduction de dimensionnalité et la préservation de l'information.

- Séparation du jeu de données

Les données sont divisées en ensembles d'entraînement (80%) et de test (20%), et le random_state est fixé à 1234 pour assurer la reproductibilité de cette division.

- Définition des modèles

Le dictionnaire 'modeles' fixe la liste des modèles à implémenter sur la base de quelques éléments mentionnés ci-après :

— "prmsgscv" : paramètres à optimiser via une grille de recherche (GridSearchCV).
— "prmfixes" : paramètres fixes du modèle.
— "perf" : destiné à stocker les performances du modèle.
— "nom" : nom court du modèle.
— "libelle" : intitulé long du modèle.
— "classe" : chemin d'importation de la classe du modèle dans scikit-learn.
— "instance" : instance du modèle avec des paramètres par défaut ou spécifiques.

Extrait du code Python

```
modeles = {

"SVC" : {
    "prmsgscv" : {"gamma" : ["scale", "auto"]
                  },
    "prmfixes" : {},
    "pred"     : None,
    "perf"     : [],
    "nom"      : "SVC",
    "libelle"  : "Classification à support de vecteurs",
    "classe"   : "sklearn.svm.SVC",
    "instance" : SVC(),
    "grid"     : None
},

"LR" : {
    "prmsgscv" : {'solver' : ['liblinear', 'lbfgs'],
                  'C'      : [0.003, 0.005, 0.01, 0.02, 0.04]},
    "prmfixes" : {},
    "perf"     : {},
    "nom"      : "LogisticRegression",
    "libelle"  : "Régression logistique",
    "classe"   : "sklearn.xxx",
    "instance" : LogisticRegression(max_iter = 10000),
    "grid"     : None
},

(...)

}
```

Voici un extrait du texte produit :

Tailles	Total	Graves (True)		Non graves (False)	
		Nombre	Prop	Nombre	Prop
Jeu d'entraînement	142113	71101	50.03%	71012	49.97%
Jeu de test	35529	17720	49.87%	17809	50.13%
Total	177642	88821	50.00%	88821	50.00%

Nombre de variables explicatives (après PCA) : 100

- - - Classification à support de vecteurs [1/11] - - -

Paramètres essayés :
gamma : ['scale', 'auto']

SVC()
Entraînement avec GridSearchCV : Recherche des meilleurs paramètres

Paramètres retenus :
gamma : scale

Prédiction :

Durée	:	1 h 43min
Score sur le jeu d'entraînement	:	0.8491%
Score sur le jeu de test	:	0.7828
Scores F1	:	0.7717, 0.7928

	precision	recall	f1-score	support
False	0.82	0.73	0.77	17809
True	0.76	0.83	0.79	17720
accuracy			0.78	35529
macro avg	0.79	0.78	0.78	35529
weighted avg	0.79	0.78	0.78	35529

Pas de feature_importances

Matrice de confusion :

```
[[ 13047   4762 ]
 [ 2956   14764 ]]
```

```
[[ 0.36722114   0.13403135 ]
 [ 0.08319964   0.41554786 ]]
```

Enregistrement du modèle entraîné : SVC.mdl, 65930443octets, 62Mo

(...)

Ce code est, d'abord, destiné à la mise au point du notebook, puis à l'affichage de résultats, son affichage n'est pas optimisé pour la lisibilité mais pour le suivi des entraînement.

2.2 Évaluation des performances des modèles

- Initialisation de comptes-rendus d'analyse

Le code initialise des variables et structures de données pour l'analyse et l'évaluation des modèles de machine learning :

— Création d'un DataFrame vide (tbl_perf) pour stocker les performances des modèles, avec des colonnes pour différentes métriques (score, durée, précision, rappel, etc.).

— Initialisation de 'logres' pour stocker les messages de log.

— Calcul et stockage de statistiques sur les ensembles de données : nombre total d'observations, nombre d'observations pour l'entraînement et le test, nombre d'observations 'graves' et 'non graves' pour chaque ensemble, etc.

— Affichage d'un tableau récapitulatif des statistiques calculées, montrant la répartition des classes dans les ensembles d'entraînement et de test.

— Affichage du nombre de variables explicatives après PCA.

— Initialisation d'un compteur de temps (ttot1) et d'un index de modèle (idx_modele) pour le suivi chronométré et numéroté des modèles.

Ce code prépare l'environnement pour l'évaluation systématique des modèles, en organisant la structure pour stocker et afficher les résultats de manière cohérente.

Voici un extrait des résultats obtenus :

	Libellé	Entraînement	Test	f1 0	f1 1	AUC
0	Classification à support de vecteurs	0.849148	0.782769	0.771738	0.792783	0.782895
1	Régression logistique	0.755582	0.752568	0.749108	0.755934	0.752608
2	Hist Gradient Boosting Classifier	0.974534	0.774128	0.766314	0.781436	0.774217

	Libellé	Précision	Recall 0	Recall 1	f béta=2
0	Classification à support de vecteurs	0.785702	0.732607	0.833183	0.816539
1	Régression logistique	0.752835	0.736931	0.768284	0.763296
2	Hist Gradient Boosting Classifier	0.775541	0.738840	0.809594	0.798091

	Libellé	Vrais nég.	Faux pos.	Faux nég.	Vrais pos.
0	Classification à support de vecteurs	13047	4762	2956	14764
1	Régression logistique	13124	4685	4106	13614
2	Hist Gradient Boosting Classifier	13158	4651	3374	14346

	Libellé	% Vrais nég.	% Faux pos.	% Faux nég.	% Vrais pos.
0	Classification à support de vecteurs	0.367221	0.134031	0.0831996	0.415548
1	Régression logistique	0.369388	0.131864	0.115568	0.38318
2	Hist Gradient Boosting Classifier	0.370345	0.130907	0.0949647	0.403783

	Libellé	Durée entraînement	Taille modèle
0	Classification à support de vecteurs	6229.81	6.59304e+07
1	Régression logistique	0.501175	1647
2	Hist Gradient Boosting Classifier	90.3109	2.41198e+07

- Optimisation des modèles et évaluation des performances

Un code est créé pour effectuer une boucle d'entraînement et d'évaluation pour plusieurs modèles de classification figurant dans le dictionnaire préalablement défini. Voici les principales étapes :

- Pour chaque modèle dans le dictionnaire 'modeles' :
  - Affiche les informations sur le modèle et ses paramètres.
  - Utilise GridSearchCV pour optimiser les hyperparamètres du modèle.
  - Entraîne le modèle avec les meilleurs paramètres trouvés.
  - Fait des prédictions sur l'ensemble de test.
- Calcule diverses métriques de performance :
  - Scores d'entraînement et de test.
  - Scores F1, AUC, précision, rappel, et F-beta.
  - Matrice de confusion.
- Tente d'afficher l'importance des caractéristiques si disponible.
- Enregistre le modèle entraîné dans un fichier.
- Stocke toutes les métriques de performance dans le DataFrame 'tbl_perf'.
- Après avoir traité tous les modèles :
  - Affiche le tableau récapitulatif des performances (tbl_perf).
  - Calcule et affiche le temps total d'exécution.
- Génère un rapport complet dans un fichier texte "entrainement.txt".

Ce code permet une évaluation systématique et comparative de plusieurs modèles de classification, en enregistrant leurs performances et caractéristiques.

Ensuite, un autre code vient le compléter pour utiliser la fonction 'tabulate' en vue d'afficher de manière formatée et lisible les performances des différents modèles de classification stockées dans le DataFrame 'tbl_perf'. Il crée cinq tableaux distincts :

- Le premier tableau montre les scores d'entraînement, de test, les scores F1 pour les classes 0 et 1, et l'AUC.
- Le deuxième tableau présente la précision, le recall pour les classes 0 et 1, et le score F-beta avec beta=2.
- Le troisième tableau affiche les valeurs brutes de la matrice de confusion (vrais négatifs, faux positifs, faux négatifs, vrais positifs).
- Le quatrième tableau montre les mêmes informations que le précédent, mais en pourcentages.

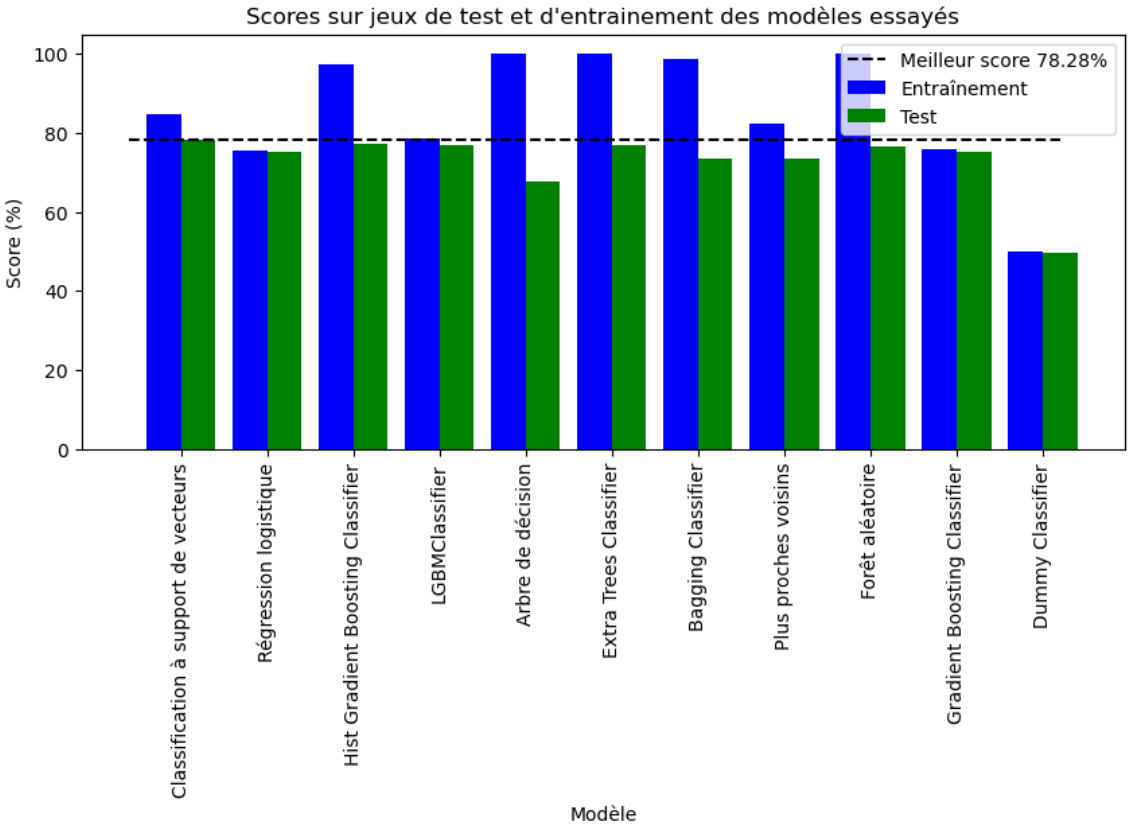
— Le dernier tableau indique la durée d'entraînement (en secondes et en format lisible) ainsi que la taille du modèle enregistré.

- Graphique de comparaison des scores d'entraînement et de test par modèle

Un code est défini pour créer un graphique à barres comparant les scores d'entraînement (en bleu) et de test (en vert) pour les différents modèles de classification :

— Il crée une figure de taille 10x4.
— Définit la largeur des barres et calcule les positions sur l'axe x pour chaque modèle.
— Trace deux séries de barres :
  — Barres bleues pour les scores d'entraînement
  — Barres vertes pour les scores de test, légèrement décalées
— Ajoute une ligne horizontale en pointillés noirs indiquant le meilleur score de test obtenu.
— Configure le titre, les étiquettes des axes et la légende.
— Affiche la légende et le graphique.

Nous obtenons le résultat suivant :



Ce graphique permet de visualiser rapidement les performances de chaque modèle et de comparer leurs scores d'entraînement et de test, facilitant l'identification du meilleur modèle et la détection d'éventuels problèmes de surapprentissage.

- Graphique d'évaluation des performances par modèle

Dans la continuité, un code est défini pour créer un graphique à barres empilées comparant les performances des différents modèles de classification :

— Il crée une figure de taille 10x4.
— Définit la largeur des barres et calcule quatre séries de positions sur l'axe x, une pour chaque catégorie de prédiction.
— Génère quatre séries de barres côte à côte pour chaque modèle réprésentant :
  — en vert, les vrais positifs (TP)

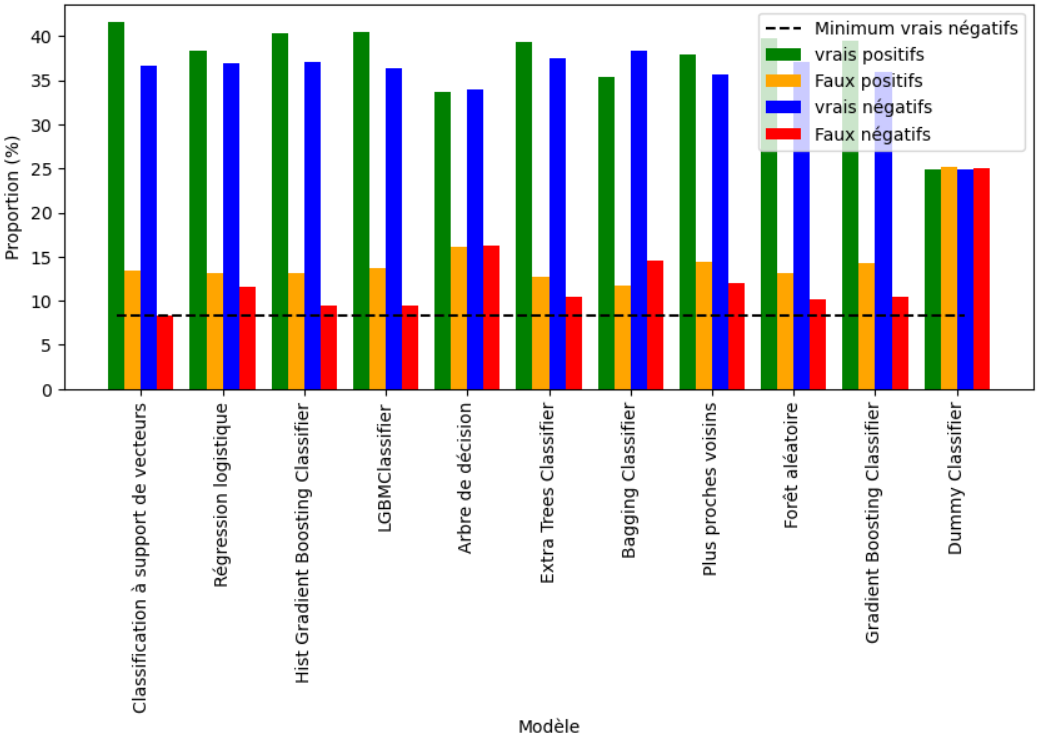
— en orange, les faux positifs (FP)
— en bleu, les vrais négatifs (TN)
— en rouge, les faux négatifs (FN)

— Calcule et ajoute une ligne horizontale en pointillés noirs indiquant le minimum de faux négatifs parmi les modèles (excluant le "Dummy Classifier").

— Configure les étiquettes des axes et la légende.

— Affiche la légende et le graphique.

Nous obtenons le résultat suivant :



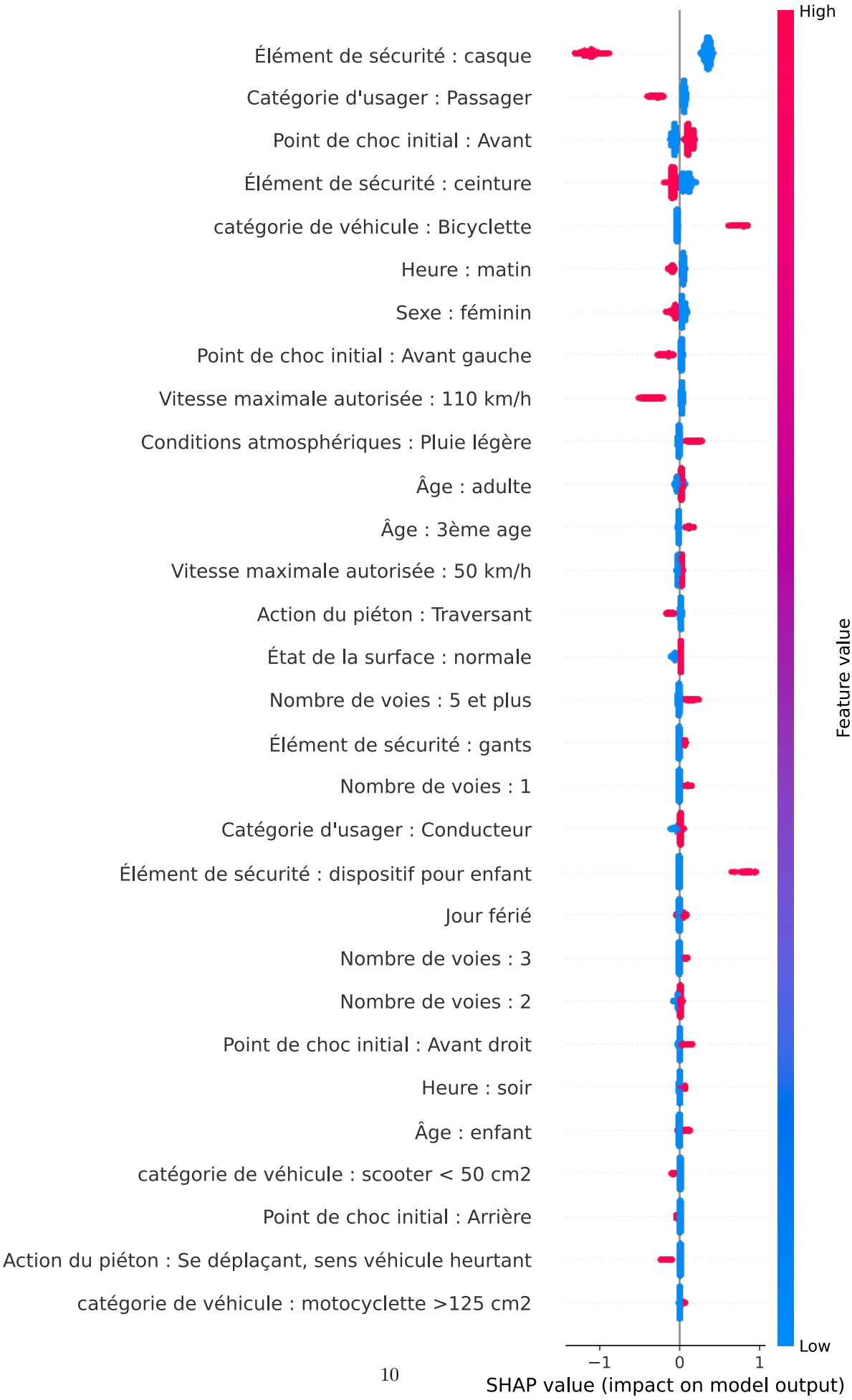
Ce graphique permet de visualiser la répartition des prédictions (TP, FP, TN, FN) pour chaque modèle, offrant une comparaison détaillée de leurs performances en termes de classification. Il aide à identifier les modèles les plus équilibrés et performants en montrant clairement leurs forces et faiblesses dans chaque catégorie de prédiction.

- Analyse SHAP

Les dernières cellules contiennent le code d'une analyse SHAP pour interpréter les prédictions du modèle LGBM. LGBM est un des meilleurs modèles et il a une taille réduite sur disque. La première cellule du notebook importe la librairie SHAP. L'évaluation avec SHAP est réalisée avec deux cellules à la fin du notebook ; cette décomposition permet de réduire les temps de mise au point. La première cellule créée un TreeExplainer utilisant le classificateur LGBM entraîné et les données du jeu de test, puis lui fait farie un évaluation avec les données de tests. La deuxième cellule fait l'affichage des variables explicatives les plus pertinentes et leurs impact sur la cible.

Nous pouvons voir des effets que nous connaissons tous et que l'on pourrait qualifier d'"évidences" : Le port du casque réduit nettement le risque de blessures graves et décés, Les cycliste sont très exposés. La ceinture protège...Cependant le dispositif enfant semble avoir un impact négatif. C'est très vraisemblablement parque les enfants victimes d'accident en voiture sont transportés avec les sièges destinés à les protéger. Ce graphique ne doit évidement pas être utilisé pour une campagne de prévention.

Le graphique montre que notre modèle voit les impacts des circonstances des accidents. Il devra être utilisé avec une combinaison de circonstances.


Élément de sécurité : casque
Catégorie d'usager : Passager
Point de choc initial : Avant
Élément de sécurité : ceinture
catégorie de véhicule : Bicyclette
Heure : matin
Sexe : féminin
Point de choc initial : Avant gauche
Vitesse maximale autorisée : 110 km/h
Conditions atmosphériques : Pluie légère
Âge : adulte
Âge : 3ème age
Vitesse maximale autorisée : 50 km/h
Action du piéton : Traversant
État de la surface : normale
Nombre de voies : 5 et plus
Élément de sécurité : gants
Nombre de voies : 1
Catégorie d'usager : Conducteur
Élément de sécurité : dispositif pour enfant
Jour férié
Nombre de voies : 3
Nombre de voies : 2
Point de choc initial : Avant droit
Heure : soir
Âge : enfant
catégorie de véhicule : scooter < 50 cm2
Point de choc initial : Arrière
Action du piéton : Se déplaçant, sens véhicule heurtant
catégorie de véhicule : motocyclette >125 cm2
High
Feature value
Low
−1
0
1
SHAP value (impact on model output)

3 Modélisation 3

3.1 Implémentation d'un modèle de deep learning

- Importation des librairies

Pour cette modélisation, plusieurs bibliothèques essentielles sont importées : pandas et numpy pour la manipulation des données, seaborn et matplotlib.pyplot pour la visualisation, scikit-learn pour la préparation et l'évaluation des données, ainsi que GridSearchCV pour l'optimisation des hyperparamètres. Pour les réseaux de neurones, le code utilise Keras via TensorFlow, incluant KerasClassifier, Sequential, Dense, Dropout, ainsi que les optimiseurs Adam et RMSprop, tout en intégrant des fonctions pour l'analyse des courbes ROC et le module warnings pour gérer les avertissements.

- Chargement et préparation des données

Le code créé définit d'abord le chemin d'accès au répertoire contenant les données prétraitées. Il procède ensuite au chargement de l'ensemble de données à partir d'un fichier CSV, puis prépare ces données. Il convertit toutes les colonnes en type entier, sépare donc les variables explicatives (features) de la variable cible 'grav_grave', divise les données en ensembles d'entraînement (80%) et de test(20%) avec une stratification pour maintenir la proportion de classes dans les ensembles, et normalise les features à l'aide de StandardScaler.

Cette préparation assure que les données sont dans un format approprié pour l'entraînement du modèle de réseau de neurones.

- Préparation du modèle

Le code créé définit une fonction 'create_model' qui construit un réseau de neurones séquentiel pour la classification binaire, avec :

— Trois couches denses (64, 32, et 16 neurones) utilisant l'activation ReLU.

— Des couches de dropout (taux de 0.4) entre chaque couche dense pour réduire le surapprentissage.

— Une couche de sortie avec un seul neurone et une activation sigmoid pour la classification binaire.

Le modèle est compilé avec la fonction de perte 'binary_crossentropy', adaptée à la classification binaire, et utilise l'accuracy comme métrique.

Un wrapper KerasClassifier est créé autour de ce modèle, permettant son utilisation avec des outils de scikit-learn comme GridSearchCV. Il est configuré avec des paramètres par défaut pour l'optimiseur, le nombre d'époques, la taille de batch, et l'initialisation des poids.

Cette approche permet une flexibilité dans l'optimisation des hyperparamètres et facilite l'intégration du modèle de deep learning dans un pipeline de machine learning.

- Configuration et exécution de la recherche d'hyperparamètres

Le code créé configure et exécute une recherche d'hyperparamètres pour optimiser le modèle de réseau de neurones :

— Il définit une grille de paramètres à tester, incluant différentes tailles de batch, nombres d'époques, optimiseurs, et initialisations de poids.

— GridSearchCV est configuré avec le modèle KerasClassifier et la grille de paramètres définie. Une validation croisée à 3 plis est spécifiée.

— La recherche d'hyperparamètres est lancée en ajustant le modèle sur les données d'entraînement normalisées.

— Enfin, le code affiche le meilleur score obtenu et les hyperparamètres correspondants.

Cette approche permet d'explorer systématiquement différentes configurations du modèle pour trouver celle qui offre les meilleures performances, en utilisant la validation croisée pour une évaluation robuste.

Nous obtenons le résultat suivant :
Best score : 0.7827, using Best parameters : 'batch_size' : 32, 'epochs' : 50, 'model___init' : 'he_normal', 'optimizer' : 'adam'

- Entraînement du meilleur modèle

Le code créé finalise la création et l'entraînement du modèle optimal :

— Il extrait les meilleurs hyperparamètres identifiés par la recherche GridSearchCV.
— Un nouveau modèle Keras est créé en utilisant ces hyperparamètres optimaux, notamment l'optimiseur et l'initialisation des poids.
— Ce modèle optimisé est ensuite entraîné sur les données d'entraînement normalisées.
— L'entraînement utilise une division de validation (20% des données d'entraînement), le nombre d'époques et la taille de batch optimaux.
— L'historique de l'entraînement est conservé, permettant de suivre l'évolution des métriques au fil des époques.

Cette étape permet d'obtenir un modèle final qui intègre les meilleures configurations identifiées lors de la recherche d'hyperparamètres, tout en fournissant des informations sur le processus d'entraînement.

3.2 Evaluation des performances

- Evaluation et prédictions

Le code créé évalue les performances du modèle optimisé sur l'ensemble de test :

— Il utilise la méthode evaluate() pour calculer la perte (loss) et la précision (accuracy) du modèle sur les données de test normalisées.
— Ces métriques de performance sont affichées avec une précision de 4 décimales.
— Ensuite, le modèle est utilisé pour faire des prédictions sur l'ensemble des données de test.
— Les prédictions, qui sont initialement des probabilités, sont converties en classes binaires (0 ou 1) en utilisant un seuil de 0.5 (point d'équilibre par défaut).

Cette évaluation permet de mesurer la performance du modèle, donnant une estimation plus réaliste de sa capacité de généralisation.

Nous obtenons le résultat suivant :
Test loss : 0.4532
Test accuracy : 0.7851

- Métriques de performance

Le code suivant évalue les performances du modèle en affichant un rapport de classification et une matrice de confusion. Également, il calcule et affiche le score AUC-ROC, qui correspond à la mesure de la capacité du modèle à distinguer entre les classes.

Le code créé effectue une évaluation des performances du modèle :

— Il génère et affiche un rapport de classification, qui inclut la précision, le rappel, le score F1 et le support pour chaque classe, ainsi que les moyennes.
— Une matrice de confusion est calculée et affichée, permettant de visualiser les vrais positifs, faux positifs, vrais négatifs et faux négatifs.
— Enfin, le score AUC-ROC est calculé et affiché. Ce score mesure la capacité du modèle à distinguer entre les classes.

Ces métriques offrent une vue complète des performances du modèle pour pouvoir appréhender son efficacité dans différents aspects de la classification.

Nous obtenons les résultat suivants :

Classification Report :

	precision	recall	f1-score	support
0	0.81	0.75	0.78	17765
1	0.76	0.82	0.79	17764
accuracy			0.79	35529
macro avg	0.79	0.79	0.78	35529
weighted avg	0.79	0.79	0.78	35529

Confusion Matrix :
AUC-ROC Score : 0.8667

[[13243 4522]
 [3114 14650]]

3.3 Analyse des résultats

- Tracé de l'historique d'entraînement

Le code créé génère une visualisation de l'historique d'entraînement du modèle :

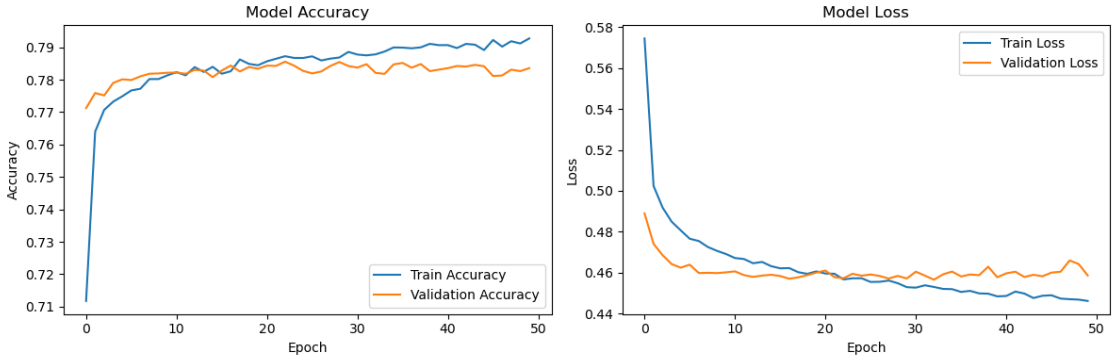
— Il génère une figure avec deux sous-graphiques côte à côte.

— Le premier graphique montre l'évolution de la précision (accuracy) au fil des époques, à la fois pour l'ensemble d'entraînement et l'ensemble de validation.

— Le second graphique illustre l'évolution de la perte (loss) au cours de l'entraînement, également pour les ensembles d'entraînement et de validation.

Cette visualisation permet d'observer rapidement comment le modèle a progressé durant l'entraînement, d'identifier d'éventuels problèmes de surapprentissage ou de sous-apprentissage, et de voir à quel moment les performances ont commencé à se stabiliser.

Nous obtenons les résultats suivants :



Interprétation globale :

— Apprentissage efficace : Le modèle apprend efficacement, atteignant taux de précision de 0.79 sur les données d'entraînement et de validation.

— Excellente généralisation : Le modèle généralise très bien aux données de validation, avec des performances presque identiques à celles sur l'ensemble d'entraînement.

— Surapprentissage minimal : L'écart très faible entre les performances d'entraînement et de validation indique un surapprentissage minimal, ce qui est acceptable.

— Stabilisation : Les courbes se stabilisent après environ 30 époques, suggérant que le modèle a atteint un plateau de performance.

— Optimisation réussie : Le modèle semble bien équilibré entre apprentissage et généralisation, suggérant que les techniques de régularisation (comme le dropout) ont été efficaces.

En conclusion, ces graphiques montrent un modèle très bien optimisé, avec une bonne capacité d'apprentissage et de généralisation. Le risque de surapprentissage est minimal, ce qui est idéal pour la tâche de prédiction de la gravité des accidents de la route. Les performances sont stables et cohérentes entre l'entraînement et la validation, indiquant que le modèle devrait bien se comporter sur de nouvelles données non vues.

- Fonctions d'analyse et de visualisation

Le code créé définit trois fonctions importantes pour l'analyse et la visualisation des résultats du modèle :

— La fonction 'get_feature_importance' calcule l'importance des caractéristiques du modèle de réseau de neurones en utilisant les poids de la première couche avec 'weights', puis en calculant la moyenne des valeurs absolues de ces poids avec 'importance'. Elle retourne un dictionnaire associant les noms des caractéristiques à leur importance.

— La fonction 'plot_feature_importance' trace un graphique à barres montrant les 20 caractéristiques les plus importantes du modèle (triées par ordre décroissant).

— La fonction 'plot_roc_curve' trace la courbe ROC pour évaluer la performance du classificateur binaire. Elle utilise 'roc_curve' pour calculer le taux de faux positifs (fpr) et le taux de vrais positifs (tpr) en fonction des vraies étiquettes (y_true) et des probabilités prédites (y_pred_proba). L'AUC est calculée en utilisant auc(fpr, tpr).

Ces fonctions permettent une analyse visuelle approfondie des résultats du modèle, en mettant en évidence les caractéristiques les plus influentes et en évaluant la capacité du modèle à distinguer entre les classes à différents seuils de classification.

- Utilisation des fonctions d'analyse et de visualisation

Le code créé applique les fonctions d'analyse et de visualisation précédemment définies :

— Il calcule d'abord l'importance des caractéristiques du meilleur modèle en utilisant la fonction 'get_feature_importance'. Le résultat est stocké dans un dictionnaire.

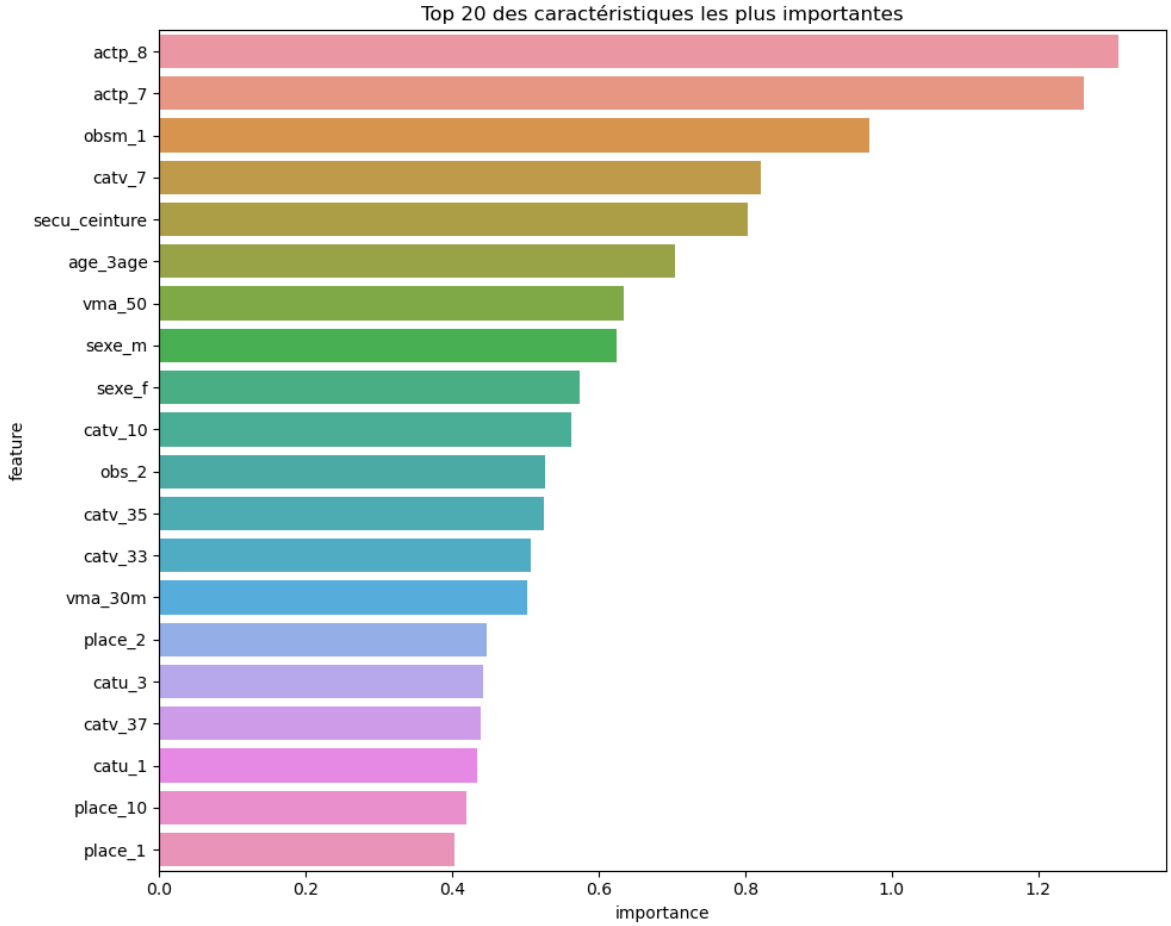
— Ce dictionnaire est ensuite utilisé pour générer un graphique à barres des caractéristiques les plus importantes via la fonction 'plot_feature_importance'.

— Ensuite, le code génère des prédictions probabilistes sur l'ensemble de test normalisé.

— Ces prédictions sont utilisées avec les vraies étiquettes pour tracer la courbe ROC en appelant la fonction 'plot_roc_curve'.

Cette séquence permet de visualiser rapidement quelles caractéristiques ont le plus d'impact sur les prédictions du modèle et d'évaluer graphiquement sa performance en termes de compromis entre le taux de vrais positifs et le taux de faux positifs.

Nous obtenons un premier résultat :



Interprétation du graphique :

— Les deux caractéristiques les plus importantes sont 'actp_8' et 'actp_7', qui sont liées à l'action du piéton au moment de l'accident.

— 'obsm_1' est la troisième caractéristique la plus importante, liée à un type d'obstacle mobile impliqué dans l'accident.

— 'secu_ceinture' souligne l'importance du port de la ceinture de sécurité dans la détermination de la gravité des blessures.

— 'age_3age' indique que l'âge des personnes impliquées, particulièrement les personnes âgées, est un facteur significatif.

— 'vma_50' et 'vma_30m' indiquent une vitesse maximale autorisée, montrant que la limite de vitesse dans la zone de l'accident est un facteur important.

— 'sexe_m' et 'sexe_f' apparaissent tous deux, suggérant que le genre des personnes impliquées joue un rôle dans la gravité de l'accident.

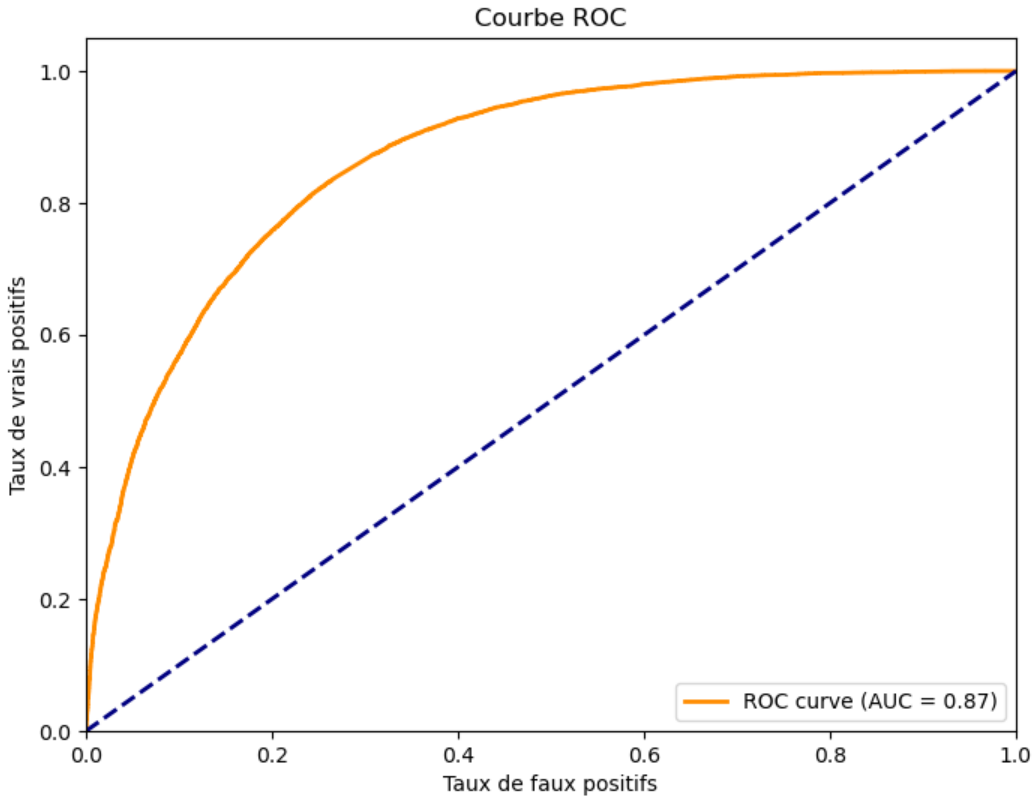
— Plusieurs catégories de véhicules (catv_7, catv_10, catv_35, catv_33, catv_37) sont présentes, indiquant que différents types de véhicules ont des impacts variés sur la gravité des accidents.

— Différentes valeurs de 'place' (place_2, place_10, place_1) suggèrent que la position des occupants dans le véhicule est un facteur important.

— 'catu_3' et 'catu_1' représentent différentes catégories d'usagers de la route, montrant que le type d'usager influence la gravité de l'accident.

Cette analyse met en évidence l'importance des facteurs liés au comportement des piétons, au type de véhicule, à l'âge et au sexe des personnes impliquées, ainsi qu'aux conditions de circulation (comme la vitesse autorisée) dans la détermination de la gravité des accidents de la route. Ces informations peuvent être cruciales pour cibler les efforts de prévention et améliorer la sécurité routière.

Nous obtenons ensuite un deuxième résultat :



Interprétation du graphique :

— Courbe ROC : La courbe orange représente la performance du modèle à différents seuils de classification. Elle montre le compromis entre le taux de vrais positifs (sensibilité) et le taux de faux positifs (1 - spécificité) à mesure que le seuil de décision varie.

— Forme de la courbe : La courbe s'élève rapidement vers le coin supérieur gauche, ce qui est souhaitable. Cela indique que le modèle atteint un bon taux de vrais positifs tout en maintenant un faible taux de faux positifs.

— Comparaison avec la ligne de base : La courbe est nettement au-dessus de la ligne diagonale en pointillés (qui représente une classification aléatoire), ce qui confirme que le modèle est bien meilleur qu'une prédiction basée sur le hasard.

— Robustesse : La courbe lisse et régulière suggère que le modèle est robuste et performant sur une large gamme de seuils de décision.

Cette courbe ROC indique que le modèle a une excellente capacité à distinguer entre les accidents graves et non graves. Avec un AUC de 0.87, il démontre une forte performance de classification, ce qui est particulièrement important dans le contexte de la prédiction de la gravité des accidents de la route.

- Analyse SHAP

Le code créé met en œuvre une analyse SHAP pour interpréter les prédictions du modèle :

— Il commence par importer la bibliothèque SHAP.

— Une fonction de prédiction est définie pour être utilisée avec l'explainer SHAP.

— Un KernelExplainer SHAP est créé, utilisant un échantillon de 100 instances des données d'entraînement comme données de fond.

— Les valeurs SHAP sont calculées pour un sous-ensemble (max 100 échantillons) de l'ensemble de test.

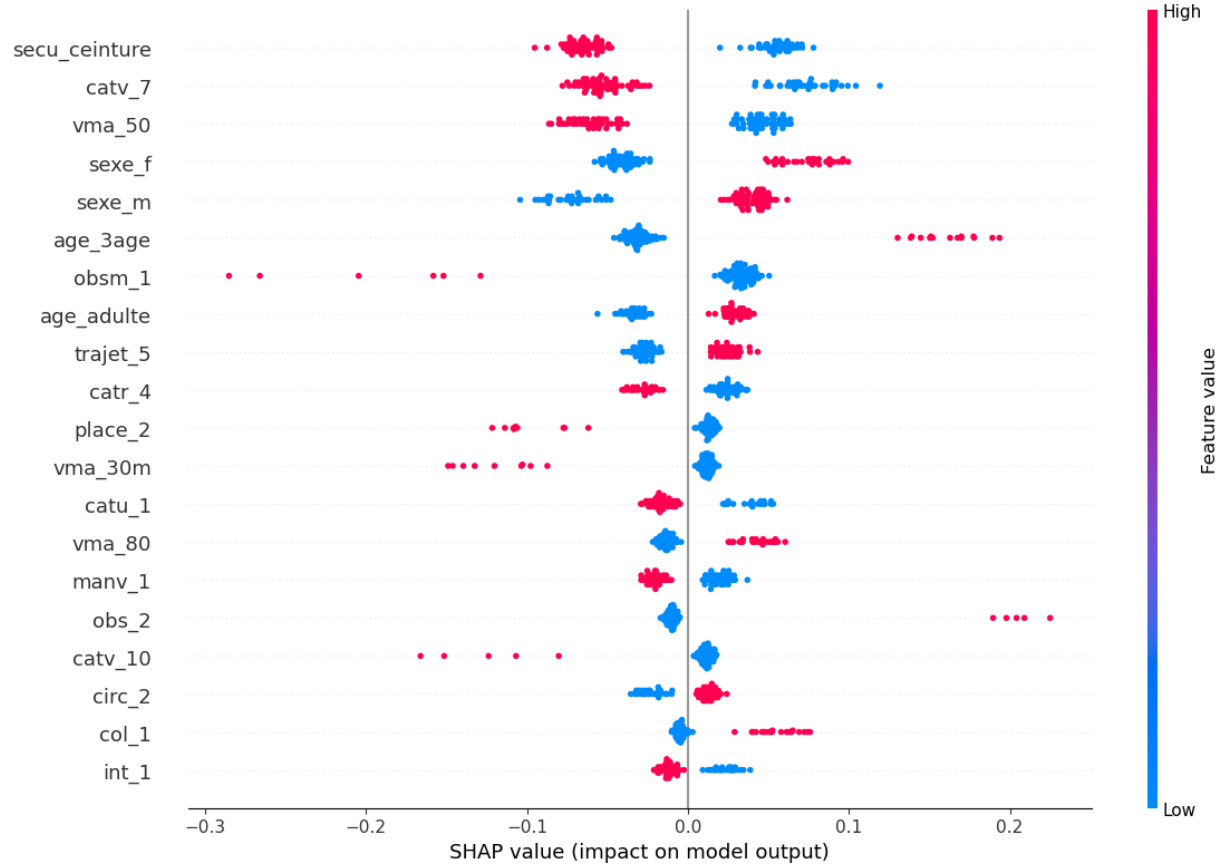
— Les dimensions des valeurs SHAP et de l'échantillon de test sont vérifiées et affichées.

— Les valeurs SHAP sont ajustées si nécessaire pour s'assurer qu'elles sont dans le format approprié.

— Enfin, un graphique récapitulatif SHAP est généré, visualisant l'importance et l'impact des caractéristiques sur les prédictions du modèle.

Cette analyse SHAP fournit une interprétation détaillée de la façon dont chaque caractéristique influence les prédictions du modèle, offrant des insights sur l'importance et la direction de l'impact des variables.

Nous obtenons le résultat suivant :



Interprétation du graphique :

Ce graphique nous montre quels éléments sont les plus importants pour prédire si un accident de la route sera grave ou non. Chaque ligne représente un facteur différent, comme le port de la ceinture de sécurité ou le type de véhicule. Plus la ligne est haute dans le graphique, plus ce facteur est important. Les points sur chaque ligne nous disent deux choses :

— Leur position :
  — À gauche, ils réduisent le risque d'accident grave.
  — À droite, ils augmentent ce risque.
— Leur couleur :
  — Rouge signifie une valeur élevée pour ce facteur.
  — Bleu signifie une valeur basse.

Par exemple, pour la ceinture de sécurité (première ligne) :

— Les points rouges à gauche signifient que le port de la ceinture (valeur élevée) tend à réduire la gravité de l'accident.
— Les points bleus à droite indiquent que l'absence de ceinture (valeur basse) tend à augmenter la gravité de l'accident.

En résumé, ce graphique SHAP offre une vue détaillée de comment chaque caractéristique influence les prédictions du modèle sur la gravité des accidents. Il met en évidence l'importance de facteurs tels que l'utilisation de la ceinture de sécurité, le type de véhicule, la vitesse autorisée, et les caractéristiques démographiques des personnes impliquées dans l'accident.

4 Conclusion

La comparaison approfondie des différents modèles de classification pour prédire la gravité des accidents de la route a révélé plusieurs enseignements importants :

1. Performance des modèles : Le réseau de neurones et le SVC ont démontré les meilleures performances globales, avec des scores AUC-ROC respectifs de 0.8562 et 0.7829. Cela suggère que ces modèles sont particulièrement efficaces pour distinguer les accidents graves des non graves. Cependant, le Hist Gradient Boosting Classifier a également montré des résultats intéressants au regard de l'équilibre établi entre la performance et le temps de calcul.

2. Importance de l'interprétabilité : L'utilisation de techniques comme SHAP a permis d'identifier les facteurs les plus influents dans la prédiction de la gravité des accidents, offrant des insights précieux au-delà de la simple performance prédictive.

3. Stabilité des prédictions : Les différents modèles ont montré une certaine cohérence dans leurs prédictions, ce qui renforce la confiance dans les résultats obtenus et suggère que les caractéristiques identifiées comme importantes sont robustes à travers différentes approches de modélisation.

Les implications pratiques pour la prédiction de la gravité des accidents de la route :

1. Conception de politiques de sécurité routière ciblées : L'identification des facteurs les plus influents dans la gravité des accidents (grâce à l'interprétabilité des modèles) peut guider l'élaboration de politiques de sécurité routière plus efficaces. Par exemple, si certaines conditions météorologiques ou caractéristiques routières sont fortement associées à des accidents graves, des mesures préventives spécifiques peuvent être mises en place.

2. Personnalisation des campagnes de sensibilisation : Les insights tirés des modèles peuvent être utilisés pour créer des campagnes de sensibilisation à la sécurité routière plus ciblées et efficaces, en se concentrant sur les facteurs de risque les plus importants identifiés par les modèles.

3. Amélioration de la conception des véhicules : Les constructeurs automobiles pourraient exploiter ces résultats pour orienter leurs efforts de recherche et développement vers des technologies de sécurité innovantes. En se concentrant sur les facteurs identifiés comme les plus déterminants dans la gravité des accidents, ils seraient en mesure de concevoir et d'implémenter des dispositifs de sécurité plus efficaces et ciblés.

4. Amélioration des systèmes de triage et d'intervention : La capacité à prédire avec précision la gravité des accidents peut permettre aux services d'urgence d'optimiser l'allocation des ressources. Par exemple, en envoyant immédiatement des équipes médicales plus spécialisées pour les circonstances d'accidents prédits comme graves.

En conclusion, bien que chaque modèle présente ses propres forces et faiblesses, l'approche multi-modèles adoptée dans cette étude offre une compréhension riche et nuancée des facteurs influençant la gravité des accidents de la route. L'application pratique de ces connaissances a le potentiel d'améliorer la sécurité routière, en permettant des interventions plus ciblées et efficaces à plusieurs niveaux, de la prévention à la gestion des accidents.