

Accidents routiers en France

Rendu 2 : rapport de modélisation

Table des matières

1. Présentation du problème de classification.....	4
1.1 Contexte de la classification.....	5
2. Étude du Problème de Classification Multiclasse	6
2.1 Algorithme Random Forest.....	6
2.1.1 Gestion de l'échantillonnage	7
2.1.2 Recherche les hyperparamètres	9
2.1.3 Résultats.....	10
2.2 Algorithme Arbre de décision.....	11
2.2.1 Présentation du modèle	11
2.2.2 Gestion de l'échantillonnage	11
2.2.3 Recherche les hyperparamètres	12
2.2.4 Résultats.....	13
2.3 Algorithme Logistique régression	16
2.3.1 Présentation du modèle logistique.....	16
2.3.2 Entraînement du modèle avec les 5 solveurs et les hyperparamètres par défauts	16
2.3.3 Recherche des hyperparamètres.....	17
2.3.4 Gestion de l'échantillonage	18
2.3.5 Ultime technique pour améliorer nos résultats : le bagging	19
2.3.6 Synthèse des Résultats.....	20
2.4 Algorithme KNN	21
2.4.1 Présentation de l'algorithme	21
2.4.2 Entrainement avec différents métriques de distances	21
2.4.3 Recherche les hyperparamètres	22
2.4.4 Entrainement en apprentissage non supervisé	23
2.4.5 Gestion de l'échantillonnage	24
2.4.6 Synthèse des résultats.....	25
2.5 Algorithme deep learning.....	25
2.5.1 Entrainement et résultats	25
2.6 Comparaison des Résultats.....	27
3. Étude du Problème de Classification Binaire.....	28
3.1 Algorithme Random Forest.....	28
3.1.1 Gestion de l'échantillonnage	28
3.1.2 Recherche les hyperparamètres	28
3.1.3 Importance des caractéristiques avec Random Forest.....	30
3.1.4 Résultats.....	32
3.2 Algorithme Arbre de décision	33
3.2.1 Gestion de l'échantillonnage	33
3.2.2 Recherche les hyperparamètres	33
3.2.3 Résultats.....	33
3.3 Algorithme Logistique régression	35
3.3.1 Entraînement du modèle avec les 5 solveurs et les hyperparamètres par défauts	35
3.3.2 Recherche des meilleurs hyperparamètres.....	36
3.3.3 Gestion de l'échantillonage	36
3.3.4 Ultime technique pour améliorer les résultats : le bagging	37
3.3.5 Synthèse des résultats.....	38
3.4 Algorithme KNN	39
3.4.1 Entrainement avec différentes metrics de distance	39
3.4.2 Recherche des hyperparamètres.....	39
3.4.3 Synthèse des résultats.....	40
3.5 Algorithme deep learning.....	40

3.5.1	Entrainement et résultats	40
3.6	Algorithme XGBoost	41
3.6.1	Présentation de l'algorithme	41
3.6.2	Gestion de l'échantillonnage	43
3.6.3	Recherche des hyperparamètres.....	43
3.6.4	Résultats.....	43
3.7	Comparaison des Résultats	45
3.8	Exploration de l'Importance des Caractéristiques via SHAP dans la Prédition des Décès Routiers.....	45
3.8.1	SHAP global	45
3.8.2	SHAP local.....	47
3.9	Optimisation des Prédictions	48
3.9.1	Courbe ROC/AUC pour le seuil de Probabilité Optimal	48
3.9.2	Courbes de F-score pour l'équilibre Recall-Precision	50
4.	<i>Exemple d'Application des Résultats de la Prédition : Gestion des Risques Routiers.....</i>	52
5.	Synthèse de résultats de la modélisation.....	54
6.	Conclusion et Perspectives	55

1. Présentation du problème de classification

Mises à jour apportées au dataset

Nous avons ajouté des colonnes supplémentaires à notre ensemble de données existant. Ces nouvelles colonnes comprennent des informations sur le sexe (sexes_cond) et l'âge des conducteurs (age_cond) et des passagers (age_usag), ainsi que le nombre total de véhicules (nb_veh) impliqués dans chaque accident. De plus, nous avons transformé les données d'âge conducteurs/usagers en variables catégorielles pour une meilleure analyse.

Répartition de l'âge des usagers et des conducteurs :

Répartition (%) nombre usagers pour chaque tranche d'âge	
25-64	60.65
18-24	19.19
11-17	7.10
65-74	5.60
75-plus	4.04
7-10	1.41
3-6	1.21
0-2	0.81
Name: age_usag, dtype: float64	

Répartition (%) nombre conducteurs pour chaque tranche d'âge	
25-34	22.92
18-24	19.84
35-44	18.01
45-54	15.36
55-64	10.14
65-74	5.60
14-17	4.09
75-94	3.52
8-13	0.42
95-plus	0.06
0-7	0.05
Name: age_cond, dtype: float64	

Repartitions nombre usagers pour chaque tranche d'âge

Repartitions nombres conducteurs pour chaque tranche d'âge

Répartition de l'âge des conducteurs en fonction de la gravité des accidents :

age_cond	grav	1.0	2.0	3.0	4.0	Total
0-7	4.1	2.5	30.5	62.9	100.0	
8-13	7.0	2.1	31.3	59.6	100.0	
14-17	18.4	2.2	28.9	50.4	99.9	
18-24	39.0	2.4	16.0	42.6	100.0	
25-34	42.9	2.1	13.4	41.7	100.1	
35-44	46.3	2.0	12.4	39.3	100.0	
45-54	46.1	2.3	14.8	36.8	100.0	
55-64	45.8	3.1	17.1	34.0	100.0	
65-74	46.2	4.3	19.8	29.7	100.0	
75-94	43.5	6.9	21.5	28.0	99.9	
95-plus	56.0	4.1	7.8	32.1	100.0	

Une seule variable quantitative nb_veh a été normalisée de cette manière :

```
# Standardisation
num_var = ['nb_veh']
df[num_var] = StandardScaler().fit_transform(df[num_var])
```

Les variables catégorielles ont été encodées par get_dummies :

```
# Encoder les variables catégorielles
cat_var = df.select_dtypes(include='object').columns
encoded_cat = pd.get_dummies(df[cat_var], prefix=cat_var, drop_first=True).astype(int)
df = df.drop(columns=cat_var)
df = pd.concat([df, encoded_cat], axis=1)
```

Séparation du dataset en train/test :

```
X = df_c.drop('grav', axis=1)
y = df_c['grav']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = y ,random_state=42)
```

1.1 Contexte de la classification

Dans notre projet de prédition de la gravité des accidents routiers, notre objectif est de prédire la variable cible 'grav', qui représente la gravité des blessures subies par les usagers accidentés. Ces usagers sont classés en quatre catégories : 1 - Indemne, 2 - Tué, 3 - Blessé, 4 - Blessé léger.

Voici répartition de la variable grave dans notre dataset :

Repartition de la variable cible par default en [%]				
	1	2	3	4
Indemne	42.5	0.0	0.0	0.0
Tue	0.0	2.6	0.0	0.0
Blessé	0.0	0.0	15.8	0.0
Blesse leger	0.0	0.0	0.0	39.2

Dans le cadre de notre étude, nous explorons deux approches de classification distinctes :

- **Classification Multiclasse :** la classification multiclasse offre une perspective plus nuancée en classant les usagers accidentés en quatre catégories distinctes : "Indemne", "Tué", "Blessé", et "Blessé léger". Cette approche permet de saisir différentes nuances de gravité des blessures.
- **Classification Binaire :**

1 – Indemne , 4 – Blessé léger - catégorie 0 : Pas grave

2 – Tué , 3 – Blessé hospitalisé - catégorie 1 : Grave

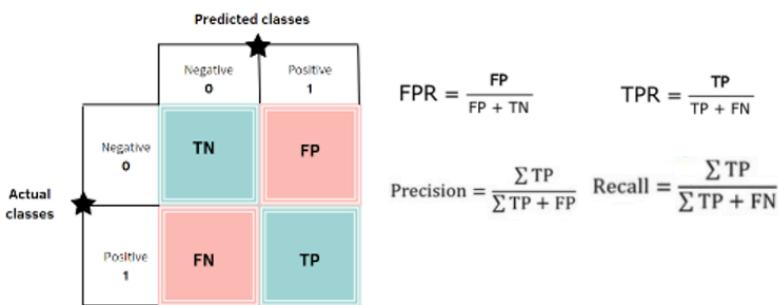
Cette approche simplifiée regroupe les usagers accidentés en deux catégories : 0 -"pas grave" et 1- "grave". Les avantages de cette approche résident dans sa simplicité, sa facilité d'interprétation et sa capacité à distinguer clairement les cas non graves des cas graves.

Repartition de la variable cible par default en [%]

	Pas Grave	Graves
Pas grave	81.7	0.0
Graves	0.0	18.3

Dans notre contexte, nous attachons une grande importance à minimiser les faux négatifs (FN). c'est-à-dire à éviter les erreurs où le modèle prédit que l'accident n'est pas grave alors qu'il l'est en réalité.

Matrice de confusion



Au-dessus, la composition de la matrice de confusion, ainsi que les formules pour la précision et le rappel, sont présentées. Comme on peut le constater, pour réduire le nombre de faux négatifs (FN), il est nécessaire d'augmenter le rappel, mais cette augmentation peut entraîner une diminution de la précision et une augmentation des faux positifs (FP), ce qui peut avoir un impact économique.

En fin de compte, le choix des métriques et la définition de leurs valeurs optimales dépendent de chaque cas spécifique en fonction des besoins particuliers

L'objectif de notre étude est de comparer les performances des modèles de machine learning dans ces deux contextes (binaire et multiclasse). Nous cherchons à déterminer lequel de ces deux scénarios offre la prédiction la plus précise de la gravité des accidents tout en minimisant les erreurs de faux négatifs (FN).

2. Étude du Problème de Classification Multiclasse

Pour rappel, ci-dessous est présentée la répartition de la variable cible dans le cas multiclasse :

Repartition de la variable cible par default en [%]				
	1	2	3	4
Indemne	42.5	0.0	0.0	0.0
Tue	0.0	2.6	0.0	0.0
Blessé	0.0	0.0	15.8	0.0
Blesse leger	0.0	0.0	0.0	39.2

2.1 Algorithme Random Forest

Présentation du model

Le modèle Random Forest est une méthode d'ensemble, basée sur des arbres de décision, qui combine les prédictions de plusieurs arbres individuels pour améliorer la précision et la robustesse des modèles de classification. Chaque arbre de décision est formé sur un échantillon aléatoire du jeu de données, et les prédictions de ces arbres sont agrégées pour obtenir une prédiction globale. Cette approche présente l'avantage de réduire le surapprentissage, d'offrir une meilleure généralisation et de gérer efficacement les données bruitées.

Avantages et Limites

Le modèle Random Forest présente des avantages significatifs, tels que sa capacité à gérer des ensembles de données complexes, à résister au surapprentissage, à offrir une certaine interprétabilité des résultats et à gérer efficacement des données déséquilibrées, ce qui en fait un choix précieux pour notre tâche de prédiction de la gravité des accidents routiers. Cependant, il peut devenir complexe avec un grand nombre d'arbres, ce qui peut rallonger les temps de formation et de prédiction, et il peut être moins interprétable que les arbres de décision individuels en raison de l'agrégation des prédictions de plusieurs arbres. Ainsi, une configuration minutieuse et une évaluation approfondie sont essentielles pour tirer pleinement parti de ses avantages tout en atténuant ses limites.

Hyperparamètres :

- **'n_estimators':** [100, 300, 500] : Ce paramètre détermine le nombre d'arbres dans la forêt. Augmenter le nombre d'arbres peut améliorer les performances, mais cela entraîne également une augmentation du temps de formation.
- **'criterion':** ['gini', 'entropy'] : Il s'agit de la fonction de mesure de qualité de la partition utilisée par les arbres de décision individuels. Le choix entre 'gini' et 'entropy' affecte la façon dont les arbres sont construits.
- **'max_depth':** [None, 5, 10, 20] : Ce paramètre définit la profondeur maximale des arbres de décision. Il peut contrôler la complexité du modèle et prévenir le surapprentissage.
- **'min_samples_split':** [2, 5, 10] : Il détermine le nombre minimum d'échantillons requis pour diviser un nœud interne de l'arbre.
- **'min_samples_leaf':** [1, 2, 4] : Ce paramètre spécifie le nombre minimum d'échantillons requis pour qu'un nœud soit considéré comme une feuille.
- **'max_features':** ['auto', 'sqrt', 'log2', None] : Il définit le nombre de fonctionnalités à considérer lors de la recherche de la meilleure partition à chaque division. Le choix dépend de la nature des données.
- **'bootstrap':** [True, False] : Ce paramètre indique si les échantillons doivent être tirés avec remplacement lors de la construction des arbres.
- **'class_weight':** [None, 'balanced', 'balanced_subsample'] : Il permet de définir les poids des classes pour l'équilibrage, ce qui est essentiel pour notre tâche de prédiction de la gravité des accidents routiers.

- '**max_samples**': [None, 0.5, 0.8] : Ce paramètre spécifie le nombre d'échantillons à tirer lors de l'apprentissage pour chaque arbre.
- '**max_leaf_nodes**': [None, 5, 10, 20] : Il limite le nombre maximum de feuilles dans chaque arbre.
- '**ccp_alpha**': [0.0, 0.1, 0.2] : Ce paramètre contrôle la complexité de l'élargissement minimal des arbres.
- '**oob_score**': [True, False] : Ce paramètre indique si les échantillons out-of-bag doivent être utilisés pour estimer l'erreur de généralisation.

2.1.1 Gestion de l'échantillonnage

Le déséquilibre dans un jeu de données peut entraîner des biais dans les prédictions, car le modèle peut avoir du mal à capturer les exemples des classes minoritaires. Pour améliorer la qualité de nos prédictions nous mettons en place des méthodes de rééchantillonnage.

Pour gérer efficacement l'échantillonnage, nous avons testé différentes méthodes et évalué leurs performances à l'aide de plusieurs métriques d'évaluation. L'objectif était de trouver une approche qui améliorera la performance de notre modèle en prédisant avec précision les classes minoritaires tout en évitant le surapprentissage. Voici un résumé des méthodes de rééchantillonnage que nous avons appliquées, ainsi que les résultats associés :

Method	Nom	MCC Train Score	MCC Test Score	F1 Weighted Train Score	F1 Weighted Test Score	F1 Train Score	F1 Test Score	Metric_train	Metric_test	
0	Meth_0	Sans équilibrage	0.399176	0.400032	0.591649	0.592305	0.406428	0.406704	0.119533	0.119213
1	Meth_1	Balanced Subsample	0.382414	0.378412	0.567476	0.564053	0.446536	0.445109	0.263676	0.265434
2	Meth_2	Random OverSampling	0.427351	0.379118	0.539490	0.564856	0.539490	0.443516	0.545495	0.259991
3	Meth_3	Random UnderSampling	0.392741	0.372906	0.517251	0.561266	0.517250	0.439456	0.498762	0.255315
4	Meth_4	Stratified K-Fold	0.398586	NaN	0.591177	NaN	0.405470	NaN	0.117771	NaN
5	Meth_5	Balanced	0.381530	0.380063	0.567363	0.568303	0.446183	0.446074	0.262905	0.260937

Les hyperparamètres de RF sont par défaut sauf '**max_depth**' = 10. Nous avons utilisé `cross_validate` avec `cv=5`.

Hyper paramètres

	Valeur
bootstrap	True
ccp_alpha	0.0
class_weight	None
criterion	gini
max_depth	10
max_features	sqrt
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.0
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	None
oob_score	False
random_state	None
verbose	0
warm_start	False

Chaque méthode a été évaluée en utilisant un ensemble de métriques : le score MCC (Matthews Correlation Coefficient), le score F1 pondéré, le score F1 et une métrique personnalisée (Metric). La métrique personnalisée a été conçue pour accorder une importance particulière à la prédiction des classes 2 et 3, qui représentent les cas de blessures graves, et est calculée comme suit :

```
return (1.5 * f1_score_class_2 + f1_score_class_3) / 2.5
```

La méthode "Balanced" (Meth_5) a été choisie comme la méthode de gestion de l'échantillonnage la plus appropriée. Nous l'avons sélectionnée en raison de sa capacité à équilibrer les classes minoritaires tout en évitant potentiellement le **surapprentissage**.

Cette méthode a montré des performances relativement équilibrées avec un score MCC de 0.380 et des scores F1 de 0.446 pour l'ensemble d'entraînement et de 0.446 pour l'ensemble de test. De plus, la méthode "Balanced" (Meth_5) a également généré une métrique personnalisée (Metric) de 0.260937, mettant en évidence une nette amélioration des performances pour les classes minoritaires (classe 2 et classe 3) par rapport à la méthode sans équilibrage (Meth_0).

Évaluation de la sensibilité des métriques

Comme expliqué précédemment, pour améliorer la prédiction de la gravité des accidents routiers, nous avons introduit une métrique personnalisée visant à accorder une importance particulière aux classes 2 et 3, qui représentent les cas de blessures graves. Pour évaluer la performance de cette métrique personnalisée et examiner la sensibilité des métriques aux résultats de prédictions pour future recherche les meilleurs hyperparamètres, nous avons réalisé une analyse comparative des métriques entre différentes méthodes d'échantillonnage (Meth_5 et Meth_3) par rapport à une méthode de base non équilibrée (Meth_0).

Pour chaque méthode d'échantillonnage (Meth_3 et Meth_5), nous avons calculé le changement en pourcentage des métriques par rapport à la méthode de base (Meth_0), qui ne comporte aucun équilibrage des classes. Cela nous a permis de quantifier l'impact de chaque méthode sur les performances du modèle. Les résultats de cette analyse sont présentés dans le tableau ci-dessous :

Method	Metric_change (%)	F1_change (%)	F1_Weighted_change (%)	MCC_change (%)
Random UnderSampling	53.3	7.5	-5.5	-7.3
Balanced	54.3	8.8	-4.2	-5.3

Les métriques F1_macro et la métrique personnalisée présentent des variations significatives de 53,3 % et 54,3 %, ainsi que de 7,5 % et 8,8 % respectivement pour les méthodes Undersampling et Balanced. Ces indicateurs se manifestent comme des outils sensibles pour mettre en évidence une amélioration notable dans la prédiction des classes minoritaires 2 et 3, en particulier la métrique personnalisée qui révèle une amélioration de 50 %. Cependant, il est essentiel de noter que les métriques F1 pondérées et MCC révèlent une dégradation, malgré l'amélioration de la prédiction pour les classes minoritaires.

Les rapports de classification ci-dessous détaillent les performances de chaque méthode d'échantillonnage. Ces résultats montrent que les méthodes d'échantillonnage permettent une meilleure prédiction des classes minoritaires, à savoir la classe 2 et la classe 3.

	precision	recall	f1-score	support
1.0	0.64	0.91	0.75	35927
2.0	1.00	0.00	0.00	2193
3.0	0.56	0.20	0.30	13334
4.0	0.61	0.54	0.58	33174
accuracy			0.63	84628
macro avg	0.70	0.41	0.41	84628
weighted avg	0.63	0.63	0.59	84628

Rapport Meth_0 sans équilibrage

	precision	recall	f1-score	support
1.0	0.68	0.86	0.76	35927
2.0	0.11	0.65	0.19	2193
3.0	0.38	0.32	0.35	13334
4.0	0.72	0.34	0.46	33174
accuracy			0.56	84628
macro avg	0.47	0.54	0.44	84628
weighted avg	0.63	0.56	0.56	84628

Rapport Under Sampling

	precision	recall	f1-score	support
1.0	0.67	0.86	0.76	35927
2.0	0.12	0.62	0.20	2193
3.0	0.40	0.32	0.36	13334
4.0	0.72	0.35	0.47	33174
accuracy			0.57	84628
macro avg	0.48	0.54	0.45	84628
weighted avg	0.63	0.57	0.57	84628

Rapport Balanced

En résumé, pour la recherche des hyperparamètres, nous utiliserons deux méthodes d'échantillonnage : 'balanced' et 'subbalanced', en nous concentrant sur les scores F1 macro et la métrique personnalisée."

2.1.2 Recherche les hyperparamètres

Pour réaliser la recherche des meilleurs hyperparamètres, nous avons appliqué la méthode RandomizedSearchCV, qui est une technique d'optimisation des hyperparamètres basée sur la recherche aléatoire. Cette méthode explore de manière efficace l'espace des hyperparamètres en effectuant une sélection aléatoire parmi un ensemble de combinaisons possibles, ce qui nous permet de trouver rapidement des paramètres optimaux pour notre modèle.

Nous avons utilisé diverses métriques pour évaluer de la performance. Nous avons examiné MCC, le rappel, le F1-score macro et F1score_classe2. De plus, nous avons introduit une métrique personnalisé (Cust_metric). Ces métriques étaient adaptées à notre cas où la détection des cas positifs est importante. Nous avons également créé une métrique personnalisée, 'Cust_metric', combinant rappel et précision de manière spécifique.

L'objectif était d'évaluer différents ensembles d'hyperparamètres de manière approfondie et de choisir ceux qui convenaient le mieux à notre tâche de prédiction.

```
# Définition des métriques de scoring pour RandomizedSearchCV
scoring = {
    'MCC': make_scorer(matthews_corrcoef),
    'F1': make_scorer(f1_score, average='macro'),
    'Cust_metric': make_scorer(my_metric),
    'F1_cl_2': make_scorer(f1_score, labels=[2], average=None)
}
```

Voici la liste des hyperparamètres à tester lors de la recherche (GridSearchCV), Nous avons utilisé cross_validate avec cv=3 :

```
# Définition de l'espace des hyperparamètres à rechercher
param_dist = {
    'n_estimators': [100, 300, 600, 1000, 2000],
    'max_depth': [3, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 4, 20],
    'class_weight': ['balanced_subsample'],
    'bootstrap': [True],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy'],
    'ccp_alpha': [0.0, 0.01],
    'min_impurity_decrease': [0.0, 0.03],
    'max_leaf_nodes': [None, 5, 10],
    'min_weight_fraction_leaf': [0.0, 0.03],
}
```

Exemple d'affichage des hyperparamètres avec leurs métriques :

results_df.sort_values(by='mean_test_F1', ascending=False).head(3)	Python			
params	mean_test_MCC	mean_test_F1	mean_test_Cust_metric	mean_test_F1_cl_2
0.0s				
703 {'n_estimators': 1000, 'min_weight_fraction_leaf': 0.0, 'min_samples_split': 10, 'min_samples_leaf': 1, 'min_impurity_decrease': 0.0, 'max_leaf_nodes': None, 'max_features': 'sqrt', 'max_depth': 20, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.443219	0.514583	0.334033	0.252169
572 {'n_estimators': 100, 'min_weight_fraction_leaf': 0.0, 'min_samples_split': 10, 'min_samples_leaf': 1, 'min_impurity_decrease': 0.0, 'max_leaf_nodes': None, 'max_features': 'sqrt', 'max_depth': 20, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.439890	0.512174	0.331727	0.249754
147 {'n_estimators': 1000, 'min_weight_fraction_leaf': 0.0, 'min_samples_split': 5, 'min_samples_leaf': 4, 'min_impurity_decrease': 0.0, 'max_leaf_nodes': None, 'max_features': 'sqrt', 'max_depth': 20, 'criterion': 'entropy', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.434466	0.508973	0.333095	0.259810

Les numéros de combinaisons d'hyperparamètres qui maximisent chaque métrique sont présentés ci-dessous :

```
Index de valeur max de MCC --- 703
Index de valeur max de f1 --- 703
Index de valeur max de Cust_metric --- 703
Index de valeur max de f1_cl2 --- 147
```

L'objectif est d'étudier comment chaque combinaison de paramètres impacte la prédiction. Nous avons identifié deux combinaisons d'hyperparamètres : la combinaison 147, qui maximise f1_classe2, la combinaison 703, qui maximise Score_personnalisé, F1_macro et MCC.

results_df.iloc[147,1:]	
<code>✓ 0.0s</code>	
mean_test_MCC	0.434466
mean_test_F1	0.508973
mean_test_Cust_metric	0.333095
mean_test_F1_cl_2	0.25981
Name: 147, dtype: object	

Métriques associées à l'index 147 des hyperparamètres

results_df.iloc[703,1:]	
<code>✓ 0.0s</code>	
mean_test_MCC	0.443219
mean_test_F1	0.514583
mean_test_Cust_metric	0.334033
mean_test_F1_cl_2	0.252169
Name: 703, dtype: object	

Métriques associées à l'index 703 des hyperparamètres

Ci-dessous, deux rapports de classification sont présentés pour deux combinaisons d'hyperparamètres, celles associées à l'index 147 et à l'index 703 :

	precision	recall	f1-score	support
1.0	0.72	0.83	0.77	143707
2.0	0.19	0.42	0.26	8772
3.0	0.41	0.48	0.44	53335
4.0	0.69	0.48	0.56	132694
accuracy			0.62	338508
macro avg	0.50	0.55	0.51	338508
weighted avg	0.65	0.62	0.62	338508

Métriques associées à l'index 147 des hyperparamètres

Rapport de classification :				
	precision	recall	f1-score	support
1.0	0.73	0.82	0.77	143707
2.0	0.20	0.34	0.25	8772
3.0	0.42	0.50	0.46	53335
4.0	0.69	0.50	0.58	132694
accuracy			0.63	338508
macro avg	0.51	0.54	0.51	338508
weighted avg	0.65	0.63	0.63	338508

Métriques associées à l'index 703 des hyperparamètres

Comme on peut le constater, la combinaison 147 permet d'obtenir un rappel (recall) plus élevé pour la classe 2, avec une valeur de 0,42, tandis que la combinaison 703 donne un rappel de 0,34 pour la même classe. Dans notre contexte de classification, nous attachons plus d'importance à un rappel élevé pour la classe 2, c'est pourquoi nous avons choisi la combinaison 147 des hyperparamètres. »

2.1.3 Résultats

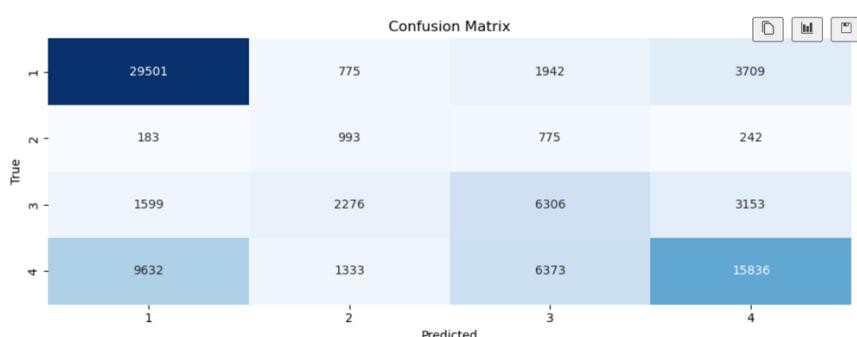
Ci-dessous, les résultats de la meilleure combinaison d'hyperparamètres (147) sont présentés pour le dataset d'entraînement et le dataset de test, ainsi que la matrice de confusion :

Rapport de classification :				
	precision	recall	f1-score	support
1.0	0.72	0.83	0.77	143707
2.0	0.19	0.42	0.26	8772
3.0	0.41	0.48	0.44	53335
4.0	0.69	0.48	0.56	132694
accuracy			0.62	338508
macro avg	0.50	0.55	0.51	338508
weighted avg	0.65	0.62	0.62	338508

Rapport classification sur dataset train pour combinitions hyperparamètres 147

Rapport de classification :				
	precision	recall	f1-score	support
1.0	0.72	0.82	0.77	35927
2.0	0.18	0.45	0.26	2193
3.0	0.41	0.47	0.44	13334
4.0	0.69	0.48	0.56	33174
accuracy			0.62	84628
macro avg	0.50	0.56	0.51	84628
weighted avg	0.65	0.62	0.62	84628

Rapport classification sur dataset Test pour combinitions hyperparamètres 147



2.2 Algorithme Arbre de décision

2.2.1 Présentation du modèle

Fondements théoriques

L'algorithme DecisionTreeClassifier fait partie de la famille des arbres de décision. Cet algorithme fonctionne en divisant l'espace des caractéristiques (features) en sous-espaces, de manière à minimiser l'hétérogénéité au sein des sous-groupes et maximiser l'hétérogénéité entre les différents sous-groupes. Pour construire l'arbre, l'algorithme utilise des critères tels que l'Entropy et l'indice Gini pour identifier la meilleure caractéristique pour la division l'ensemble de données. Le processus de division continue jusqu'à ce que certaines conditions d'arrêt (cf hyperparamètres) soient satisfaites. Après la construction de l'arbre, certaines branches peuvent être éliminées/élaguées (pruning) pour éviter le surajustement (overfitting).

Avantages

- Facilité d'interprétation: les arbres de décision sont faciles à visualiser et à comprendre. Néanmoins, cela est vite limité selon la taille de l'arbre et si la classification est binaire ou multiclasse.
- Normalisation non-obligatoire: pas besoin de normalisation des données, mais cela reste possible.
- Capable de gérer des données non linéaires, et des interactions entre caractéristiques.
- Utilisation pour des variables catégorielles ou continues

Limites

- Sensibilité aux données: les arbres de décision sont sensibles aux variations des données d'entraînement.
- Risque de surajustement: risque élevé d'overfitting, surtout avec des arbres profonds.
- Biais: les arbres peuvent être biaisés si une classe domine.

Hyperparamètres

- criterion: utilisé pour mesurer la qualité d'une division (e.g., "gini", "entropy").
- splitter: stratégie utilisée pour choisir la division à chaque nœud (e.g., "best", "random").
- max_depth: profondeur maximale de l'arbre.
- min_samples_split: nombre minimum d'échantillons requis pour diviser un nœud interne.
- min_samples_leaf: nombre minimum d'échantillons requis pour être à un nœud feuille.
- max_features: nombre de caractéristiques à considérer lors de la recherche de la meilleure division.
- max_leaf_nodes: nombre maximum de nœuds feuilles.
- min_impurity_decrease: une division sera considérée si la diminution de l'impureté est supérieure à cette valeur.
- class_weight: Poids associé aux classes, utile pour les déséquilibres de classes.
- ccp_alpha: paramètre de complexité de coût pour l'élagage (pruning).
- random_state: graine pour le générateur de nombres aléatoires.

2.2.2 Gestion de l'échantillonnage

Nous sommes dans une gestion multiclasse où les 4 classes présentes sont fortement déséquilibrées :

1.0	> 179634
2.0	> 10965
3.0	> 66669
4.0	> 165868

Nous avons opté pour un rééquilibrage par Oversampling : oversample = SMOTE()

La méthode SMOTE, qui signifie Synthetic Minority Over-sampling Technique, est une technique d'oversampling utilisée pour résoudre le problème de déséquilibre des classes dans un ensemble de données d'apprentissage.

Avantages

- Peut améliorer la performance du modèle sur la classe minoritaire.
- Moins de risque de surajustement comparé à l'oversampling simple (duplication d'exemples).

Limites

- Augmente la taille de l'ensemble de données et, par conséquent, la complexité du modèle.
- Risque de bruit: peut introduire du bruit si les exemples de la classe minoritaire ne sont pas bien représentés

2.2.3 Recherche les hyperparamètres

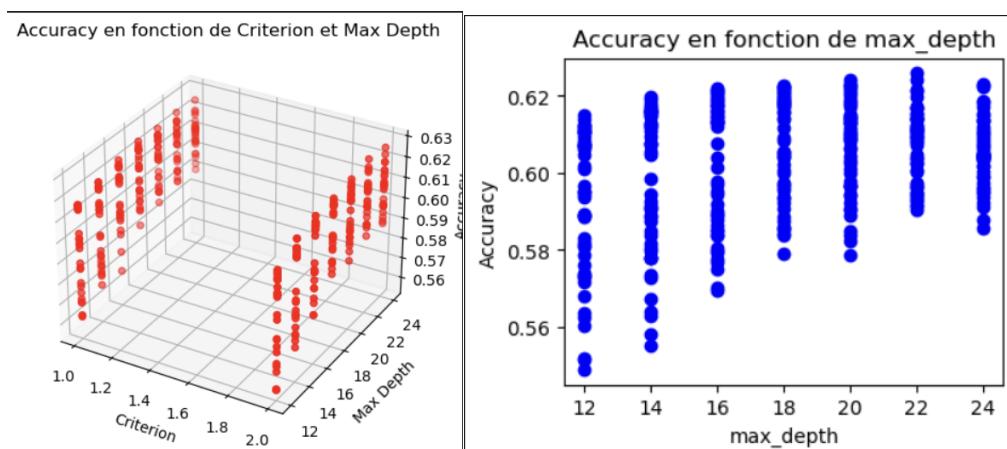
Méthode par GridSearchCV et RandomizedSearchCV

La recherche des hyperparamètres est la partie la plus couteuse en terme de temps lors de l'exécution de modèles. Cette partie d'appuie traditionnellement sur les GridSearchCV et RandomizedSearchCV. Le GridSearchCV permet de trouver la meilleure combinaison dans la grille définie, à condition que celle-ci soit bien définie. Le RandomizedSearchCV n'a pas pour but de trouver la meilleure combinaison, mais d'explorer un espace d'hyperparamètres plus large, et trouve parfois de meilleures combinaisons que la recherche exhaustive.

Méthode empirique

Néanmoins, ces deux méthodes sont fortement consommatrices de ressources, ne proposent pas de visualisation de sortie graphique et n'offrent pas de tendance/ordre de grandeur sur les hyperparamètres.

Pour cette raison, le Chapitre 1 (1.0 DT - Exploration préliminaire sur 4 classes - Métrique accuracy) dans le code Python fourni pour le DecisionTree (et XGBoost) *se veut empirique et exploratoire*. Le but étant de simplement dégager des ordres de grandeurs en faisant varier un hyperparamètre à la fois, puis en affichant un petit graph afin de voir l'évolution de l'accuracy en sortie en fonction de l'évolution de cette hyperparamètre. Ci-dessous quelques exemples :



Également, on a cherché à optimiser le modèle DecisionTree en gardant les 4 classes initiales de départ, sur différentes métriques (Chapitre 2 : 2. DT - RandomSearch - 4 classes - Plusieurs métriques) :

- MCC ou matthews_corrcoef
- F1 macro
- F1 macro classe 2 (morts)
- Spécifique / Custom

On cherchera dans un second temps au Chapitre 3 (3. DT - GridSearch - 4 classes - Métrique : F1 macro) à optimiser le F1 score, puis dans le Chapitre 4 (4. DT - GridSearch - 4 classes - Métrique : F1 classe 2) à optimiser le F1 score de la classe 2 (mort) afin de prédire au mieux ce qui conduit au décès. Nous verrons ensuite que la prédiction de cette classe 2 est difficile, et donne des résultats médiocres.

On décidera alors de voir comment se comporte l'algorithme dans un cas de décision binaire en fusionnant ensemble les classes : Les indemnes et bléssés légers formeront la classe 0, tandis que la classe 1 sera composée des bléssés graves et décès. Cela sera détaillé dans la section de ce rapport dédié à l'étude des algorithmes sur une classification binaire.

2.2.4 Résultats

Chapitre : 2. DT - RandomSearch - 4 classes - Plusieurs métriques

On a utilisé ici un RandomizedSearchCV (pour limiter les temps de traitement) afin de voir quelle serait la métrique la plus performance parmi les 4 énoncées précédemment.

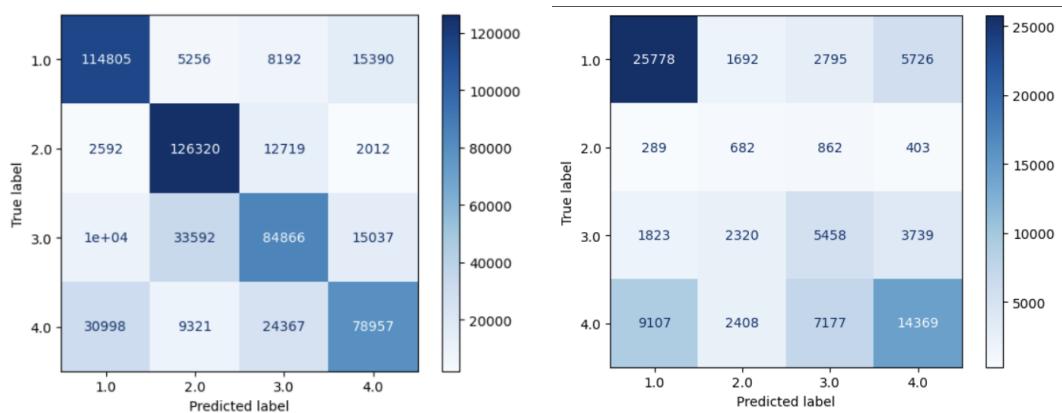
Hyperparamètres							Métriques			
splitter	random_state	min_samples_split	min_samples_leaf	max_features	max_depth	criterion	MCC	F1	Cust_metric	F1_class_2
best	193	82	11	sqrt	24	entropy	0.399604	0.442709	0.183200	0.045360
best	65	22	1	sqrt	30	entropy	0.375568	0.446841	0.215605	0.108203
best	65	2	1	sqrt	28	entropy	0.353960	0.440313	0.220304	0.124650
best	65	2	1	sqrt	28	entropy	0.353960	0.440313	0.220304	0.124650

On remarque immédiatement que la métrique F1 classe 2 est particulièrement difficile à prédire (0,124650). Il s'agit à l'origine de la classe fortement déséquilibrée et minoritaire. La métrique personnalisée est assez arbitraire, elle ne donne pas de bonnes performances non plus (0,220304). Les résultats s'améliorent avec les métriques MMC (0,399604) et F1 (0,446841).

On tentera d'expliquer plus tard dans les conclusions du rapport pourquoi les performances de la prédiction de la classe 2 sont faibles. Pour le moment, **le choix se porte sur l'étude de la métrique F1** qui sera disponible quelque soit l'algorithme ML choisi, et que la classification soit binaire ou multiclasse.

En reprenant les hyperparamètres trouvé pour F1, on va regarder de plus près les résultats obtenus :

	Train				Test			
	1.0	2.0	3.0	4.0	1.0	2.0	3.0	4.0
Classe réelle								
1.0	114805	5256	8192	15390	25778	1692	2795	5726
2.0	2592	126320	12719	2012	289	682	862	403
3.0	10148	33592	84866	15037	1823	2320	5458	3739
4.0	30998	9321	24367	78957	9107	2408	7177	14369



On se rend vite compte que les classes les mieux prédites lors de l'entraînement, ne sont pas celles qui sont le mieux prédites sur l'ensemble de test/validation.

Si l'on s'intéresse au F1 score (macro), nous obtenons les résultats suivants :

Train F1 score (macro): 69.83%

Validation F1 score (macro): 43.06%

Il n'est pas nécessaire d'aller regarder plus en détail chacune des classes, non avons ici ***un phénomène d'overfitting***. On va donc tenter d'optimiser/corriger cela dans les deux chapitres suivant tant sur la métrique F1 macro que la métrique F1 classe 2.

Chapitre : 3. DT - GridSearch - 4 classes - Métrique : F1 macro

On vise dans cette partie à améliorer le résultat pour la métrique F1-macro. Pour cela, on a lancé un gridsearchCV, qui a sorti les meilleurs paramètres suivants :

best_params

```
{'criterion': 'entropy',
'max_depth': 24,
'max_features': 'sqrt',
'min_samples_leaf': 4,
'min_samples_split': 16,
'random_state': 16,
'splitter': 'best'}
```

Résultats

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
1.0	0.71	0.85	0.78	0.68	0.81	0.74
2.0	0.57	0.23	0.32	0.24	0.07	0.10
3.0	0.61	0.44	0.51	0.44	0.34	0.38
4.0	0.67	0.63	0.65	0.60	0.55	0.57
accuracy			0.68			0.62
macro avg	0.64	0.54	0.56	0.49	0.44	0.45
weighted avg	0.67	0.68	0.67	0.60	0.62	0.60

On remarque une forte diminution de l'overfitting, même celui-ci reste malgré tout présent dans une proportion non-négligeable. Parallèlement, le score F1 macro s'est légèrement amélioré sur l'ensemble de test, mais à diminuer sur l'ensemble d'entraînement. Globalement, on prédit donc légèrement mieux. Les classes 1 et 4 sont toujours celles les mieux prédictes alors que l'algorithme ne parvient toujours pas à prédire les classes 2 et 3 de façon convaincante, tant les scores restent faibles pour ces deux dernières classes.

Chapitre : 4. DT - GridSearch - 4 classes - Métrique : F1 classe 2

On vise dans cette partie à améliorer le résultat pour la métrique F1-classe 2 (décès). Pour cela, on a lancé un gridsearchCV, qui a sorti les meilleurs paramètres suivants :

best_params

```
{'criterion': 'entropy',
'max_depth': 64,
'max_features': 'sqrt',
'min_samples_leaf': 2,
'min_samples_split': 1,
'random_state': 128,
'splitter': 'random'}
```

Résultats

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
1.0	0.78	0.90	0.83	0.66	0.77	0.71
2.0	0.54	0.46	0.50	0.15	0.12	0.13
3.0	0.66	0.62	0.64	0.38	0.36	0.37
4.0	0.80	0.68	0.73	0.57	0.49	0.53
accuracy			0.76			0.58
macro avg	0.69	0.67	0.68	0.44	0.43	0.44
weighted avg	0.76	0.76	0.76	0.57	0.58	0.57

On remarque une amélioration de près de 30% du F1-score de la classe 2 par rapport au chapitre précédent puisque celui-ci passe de 0,10 à 0,13. Néanmoins, ces valeurs restent malgré tout très faibles. En même temps, cela a conduit à une légère diminution du F1-score macro et l'accentuation de l'overfitting.

Conclusions du DecisionTreeClassifier

Cet algorithme semble avoir des difficultés à prédire correctement une problématique de classification multiclass lorsque des données sont fortement déséquilibrées. Un rééquilibrage par oversampling permet d'améliorer sensiblement les performances, mais compte tenu du nombre de features l'arbre reste assez profond et devient sensible à l'overfitting. Il conviendrait de tester l'élagage/pruning afin de voir comment cela pourrait améliorer le scoring. Néanmoins, faute de temps cela n'a pas été fait, afin de pouvoir se concentrer sur les apports/différences de ce même algorithme lors d'une classification binaire, et lors de l'utilisation du XGBoost également sous forme binaire.

2.3 Algorithme Logistique régression

2.3.1 Présentation du modèle logistique

Fondements théoriques

La régression logistique est un modèle statistique qui prédit la probabilité qu'une observation appartienne à une catégorie sur la base de variables indépendantes. Elle s'applique notamment à la classification binaire (deux classes) et multiclasse (plusieurs classes). Le modèle utilise la fonction sigmoïde pour transformer une combinaison linéaire des caractéristiques en une probabilité.

Avantages et limites

Avantages :

La régression logistique est relativement simple et interprétable. Les coefficients des variables peuvent être utilisés pour comprendre l'influence relative de chaque variable sur la probabilité de la cible. La régression logistique est efficace sur de grands datasets et a la capacité de fournir des probabilités pour les prédictions.

Limites :

Cependant, elle repose sur l'hypothèse que les variables indépendantes sont linéairement liées au logit de la variable dépendante, et peut avoir des difficultés avec des caractéristiques hautement corrélées (multicollinéarité).

2.3.2 Entraînement du modèle avec les 5 solveurs et les hyperparamètres par défauts

Description des solveurs

Pour garantir la robustesse et la performance de notre modèle de régression logistique, nous allons tester différents solveurs. Un solveur est une méthode algorithmique utilisée pour trouver les paramètres (par exemple, les poids) qui minimisent l'erreur de prédiction.

Voici une brève description des cinq solveurs que nous avons utilisés :

- liblinear : Adapté pour de petits ensembles de données, ce solveur utilise une méthode de descente coordonnée. Il gère la pénalité L1.
- newton-cg : Ce solveur emploie une méthode basée sur la minimisation de Newton. Il est adapté pour des ensembles de données de taille moyenne à grande. Il gère la pénalité L2.
- lbfgs : Acronyme de « limited-memory Broyden-Fletcher-Goldfarb-Shanno », c'est une approximation de l'algorithme de Broyden-Fletcher-Goldfarb-Shanno (BFGS), utilisant une quantité limitée de mémoire. Il est recommandé pour des ensembles de données de plus grande taille. Il ne gère que la pénalité L2
- sag : « Stochastic Average Gradient descent ». C'est une variante de la descente de gradient, qui est plus rapide pour de grands ensembles de données. Il prend en charge la pénalité L1 et la pénalité « elasticnet ».
- saga : Une version améliorée du « sag », adaptée aussi pour la pénalisation l1, ce qui la rend utile pour gérer la perte multimoniale pour des problèmes mult classes.

Il est essentiel de tester ces différents solveurs car leurs performances peuvent varier en fonction de la nature et de la taille de l'ensemble de données.

Nous avons entraîné le modèle de régression logistique avec différentes configurations pour évaluer les performances, en particulier pour la classe 2, sur les ensembles de test et d'entraînement. Ces évaluations ont été réalisées selon une méthode directe, sans stratégie d'échantillonnage spécifique, et avec une validation croisée en trois plis.

Résultats

Ci-après, un détail des performances obtenues pour chaque solveur en mettant l'accent sur la classe 2 :

Id	Méthode	Échantillonage	CV	Test								test				
				Hyperparamètres						test						
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (classe 2)
1	Simple	None	3	newton-cg	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877
2	Simple	None	3	lbfgs	1,00	1000	ovr	l2			0,484	0,028	0,053	0,463	0,652	0,877
3	Simple	None	3	liblinear	1,00	1000	ovr	l2			0,833	0,005	0,009	0,442	0,649	0,875
4	Simple	None	3	sag	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877
5	Simple	None	3	saga	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877
Max											0,833	0,028	0,053	0,463	0,652	0,877

Lorsque nous examinons les métriques de performance pour la classe 2, nous constatons que tous les solveurs peinent à identifier correctement cette classe, comme le montrent les valeurs de rappel et de score F1. En effet, le rappel maximal observé pour cette classe est de 0,028, et le F1-score le plus élevé est également de 0,053.

Remarquons aussi que le solveur "Liblinear" présente des résultats très différents des autres solveurs avec une précision très élevée de 0,833, mais un rappel de 0,005 et un F1-score pour la classe 2 de 0,009.

Conclusion

L'analyse des performances des différents solveurs révèle que les métriques de performance pour la classe 2 sont très basses, quel que soit le solveur employé.

D'une part, une des premières mesures à envisager est l'ajustement des hyperparamètres. La performance d'un modèle de machine learning, en particulier la régression logistique, est grandement influencée par le choix des hyperparamètres.

D'autre part, la sous-représentation de la classe 2 dans nos données complique sa classification. Compte tenu de l'importance que nous accordons à la classe 2, il serait judicieux d'envisager des techniques comme l'ajustement des poids de classe, le suréchantillonnage, ou l'ajout de nouvelles caractéristiques pour améliorer les performances.

En conclusion, malgré de légères variations entre les solveurs, le principal défi est d'optimiser la classification de la classe 2, tant au niveau des hyperparamètres que de la gestion des données. La prochaine étape de notre démarche sera d'utiliser GridSearchCV afin d'identifier la combinaison optimale d'hyperparamètres pour chaque solveur. L'un de ces paramètres ajustera le poids des classes, ce qui pourrait contribuer à résoudre une partie du problème observé.

2.3.3 Récherche des hyperparamètres

Méthodologie

Pour optimiser notre modèle de régression logistique, nous avons utilisé la méthode GridSearchCV. Cette méthode permet de tester systématiquement diverses combinaisons d'hyperparamètres pour identifier celle qui produit le meilleur modèle. Dans notre cas, nous avons considéré les hyperparamètres suivants :

- Pénalité : L1, L2, elasticnet et aucune.
- Solveur : liblinear, saga, newton-cg, lbfgs, sag.
- Nombre maximum d'itérations : 1000, 10000, 100000.
- C : 0.001, 0.01, 0.1, 1, 10.
- multi_class : ovr, multinomial.
- Poids de classe : balanced, None.

Résultats

Id	Méthode	Échantillonage	CV	Hyperparamètres								Test					
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (classe 2)	
				Max							0,492	0,028	0,053	0,463	0,652	0,877	
6	GridsearchCV	None	3	lbfgs	0,00	10000	multinomial		-								

À partir du GridSearchCV, nous avons identifié la meilleure combinaison d'hyperparamètres pour notre modèle de régression logistique.

Cette combinaison utilise le solver lbfgs avec une pénalité None, un C nul, une itération maximale de 10.000, et une stratégie multiclass de type multinomial. Ces paramètres ont été déterminés pour maximiser la performance du modèle sur nos métriques.

À noter qu'à l'entraînement, les meilleurs résultats ont été obtenus avec l'hyperparamètre « class-weight » fixé à « balanced ». Toutefois, les résultats sur le test ont été très décevants, traduisant un surapprentissage très prononcé.

Conclusion

Les résultats obtenus en utilisant la meilleure combinaison d'hyperparamètres de GridSearchCV ont été comparés à ceux obtenus avec des configurations simples. En se concentrant sur la classe 2, les mesures clés pour la comparaison étaient l'exactitude, le F1, le rappel et la précision pour cette classe spécifique.

Le modèle optimisé avec GridSearchCV a montré des performances similaires à celles des modèles avec des configurations simples.

Cependant, malgré ces optimisations, la sous-représentation de la classe 2 demeure une problématique centrale, poussant à la réflexion sur la gestion de l'échantillonnage.

2.3.4 Gestion de l'échantillonnage

Méthodologie

Face à la sous-représentation de la classe 2 dans nos données, il est impératif d'adopter des stratégies d'équilibrage des classes pour améliorer sa classification. Dans cette optique, nous avons exploré trois techniques principales : le sous-échantillonnage, le suréchantillonnage et le smote.

- Sous-échantillonnage :** Cette stratégie réduit de manière aléatoire les échantillons des classes majoritaires pour équilibrer l'ensemble d'entraînement.
- Suréchantillonnage :** À l'inverse, cette méthode augmente le nombre d'échantillons de la classe minoritaire en employant la technique RandomOverSampler, sans modifier les échantillons des autres classes.
- SMOTE :** Technique d'échantillonnage qui génère des échantillons artificiels pour la classe minoritaire afin d'équilibrer la distribution.

Résultats

Id	Méthode	Échantillonage	CV	Hyperparamètres								Test					
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (classe 2)	
				Max							0,121	0,648	0,205	0,466	0,583	0,873	
7	GridsearchCV	UnderSamp.	5	saga	1,00	1000	ovr	elasticnet	0,50								
8	RandomsearchCV	OverSamp.	3	saga	0,10	100000	ovr	elasticnet	0,50								
9	RandomsearchCV	SMOTE	3	saga	1,00	1000	multinomial	elasticnet	0,50	balanced							

Conclusion

Les modèles obtenus par rééchantillonnage en oversampling et undersampling montrent des performances similaires et supérieur au modèle smote.

Les techniques d'échantillonnage ont apporté une nette amélioration à la classification de la classe 2, particulièrement en ce qui concerne le rappel, par rapport au modèle précédent. Toutefois, cette progression s'est accompagnée d'une baisse de la précision. Le score F1 pour la classe 2 est passé de 0,053 à 0,205. Il est essentiel de mentionner qu'un overfitting a été observé durant la phase d'apprentissage avec un score F1 pour la classe 2 de 0.590.

En complément à ces constatations relatives aux techniques de sous-échantillonnage et de suréchantillonnage, nous envisageons de poursuivre nos explorations avec une approche combinée utilisant le bagging. Cette approche prometteuse sera détaillée dans la section suivante.

2.3.5 Ultime technique pour améliorer nos résultats : le bagging

Méthodologie :

La sous-représentation de la classe 2 dans nos données a motivé l'exploration de techniques d'équilibrage avancées. Après avoir essayé le sous-échantillonnage et le suréchantillonnage simple, nous combinons ici le Bagging à la fois avec le sous-échantillonnage et le suréchantillonnage. Le bagging, également connu sous le nom de « bootstrap aggregating », vise à améliorer la stabilité et la précision des modèles en les combinant.

- Bagging avec sous-échantillonnage : La classe majoritaire est réduite pour équilibrer la distribution des classes, suivi de l'application du Bagging.
- Bagging suréchantillonnage : Augmentation de la classe minoritaire, suivi de l'application du Bagging.

Résultats

Id	Méthode	Échantillonage	CV	Test												
				Hyperparamètres								test				
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (classe 2)
10	Bagging	UnderSamp.	3	saga	1,00	10000	ovr	elasticnet	0,50		0,118	0,640	0,199	0,463	0,580	0,873
11	Bagging	OverSamp.	3	saga	0,10	10000	ovr	elasticnet	0,50		0,120	0,640	0,202	0,468	0,585	0,874
		Max									0,120	0,640	0,202	0,468	0,585	0,874

Conclusion

L'analyse comparative des méthodes de Bagging, associées soit au suréchantillonnage (OverSampling) soit au sous-échantillonnage (UnderSampling), ne montre pas d'amélioration significatives en termes de score F1, rappel et précision par rapport au simple méthodes de rééchantillonnage. Les performances globales de ces modèles restent modestes.

2.3.6 Synthèse des Résultats

Test																	
Id	Méthode	Échantillonage	CV	Hyperparamètres							test						
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (classe 2)	
1	Simple	None	3	newton-cg	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877	
2	Simple	None	3	lbfgs	1,00	1000	ovr	l2			0,484	0,028	0,053	0,463	0,652	0,877	
3	Simple	None	3	liblinear	1,00	1000	ovr	l2			0,833	0,005	0,009	0,442	0,649	0,875	
4	Simple	None	3	sag	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877	
5	Simple	None	3	saga	1,00	1000	ovr	l2			0,492	0,028	0,053	0,463	0,652	0,877	
6	GridsearchCV	None	3	lbfgs	0,00	10000	multinomial	-			0,492	0,028	0,053	0,463	0,652	0,877	
7	GridsearchCV	UnderSamp.	5	saga	1,00	1000	ovr	elasticnet	0,50		0,121	0,648	0,205	0,466	0,583	0,873	
8	RandomsearchCV	Oversamp.	3	saga	0,10	100000	ovr	elasticnet	0,50		0,121	0,647	0,204	0,467	0,585	0,874	
9	RandomsearchCV	SMOTE	3	saga	1,00	1000	multinomial	elasticnet	0,50	balanced	0,220	0,104	0,141	0,483	0,640	0,843	
10	Bagging	UnderSamp.	3	saga	1,00	10000	ovr	elasticnet	0,50		0,118	0,640	0,199	0,463	0,580	0,873	
11	Bagging	OverSamp.	3	saga	0,10	10000	ovr	elasticnet	0,50		0,120	0,640	0,202	0,468	0,585	0,874	
											0,833	0,648	0,205	0,483	0,652	0,877	
Entraînement													Entraînement				
Id	Méthode	échantillonage	CV	Hyperparamètres							Entraînement					Entraînement	
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 2)	recall (classe 2)	F1-score (classe 2)	F1-score (global)	Accuracy (global)	ROC (moyenne)	
1	Simple	None	3	newton-cg	1,00	1000	ovr	l2			0,414	0,026	0,048	0,459	0,648	0,836	
2	Simple	None	3	lbfgs	1,00	1000	ovr	l2			0,418	0,026	0,048	0,459	0,649	0,836	
3	Simple	None	3	liblinear	1,00	1000	ovr	l2			0,462	0,004	0,008	0,438	0,645	0,833	
4	Simple	None	3	sag	1,00	1000	ovr	l2			0,414	0,026	0,048	0,459	0,648	0,836	
5	Simple	None	3	saga	1,00	1000	ovr	l2			0,416	0,026	0,048	0,459	0,648	0,836	
6	GridsearchCV	None	3	lbfgs	-	10000	multinomial	-			0,409	0,026	0,049	0,459	0,649	0,836	
7	GridsearchCV	UnderSampling	5	saga	1,00	1000	ovr	elasticnet	0,50		0,554	0,631	0,590	0,537	0,548	0,804	
8	RandomsearchCV	Oversampling	3	saga	0,10	100000	ovr	elasticnet	0,50		0,560	0,643	0,599	0,545	0,557	0,809	
9	RandomsearchCV	SMOTE	3	saga	1,00	1000	multinomial	elasticnet	0,50		0,742	0,790	0,738	0,609	0,623	0,860	
10	Bagging	UnderSampling	3	saga	1,00	10000	ovr	elasticnet	0,50		0,554	0,633	0,591	0,463	0,580	0,804	
11	Bagging	Oversampling	3	saga	0,10	10000	ovr	elasticnet	0,50		0,560	0,643	0,599	0,469	0,586	0,809	
											0,742	0,790	0,738	0,609	0,649	0,860	

- Précision (classe 2) et Exactitude (global) :

Les modèles 1, 2, 3, 4, 5 et 6 affichent la meilleure précision pour la classe 2 et l'exactitude globale, atteignant des valeurs maximales de 0,49206 (si l'on exclut la valeur de 0.83333) et 0,65203 respectivement.

- Rappel (classe 2) et F1-score (classe 2) :

Les modèles 7, 8, 10 et 11, qui utilisent des techniques d'échantillonnage, présentent le rappel le plus élevé pour la classe 2 et le F1-score le plus élevé pour la classe 2, avec des scores maximaux de 0.648 et 0.205 respectivement. Il est à noter que les techniques de bagging n'ont pas engendré de meilleures performances.

- F1-score (global) :

Aucun modèle ne se démarque nettement. Le F1-score global maximal atteint est de 0.483 par le modèle 9. Ceci suggère que les techniques d'échantillonnage ont permis d'améliorer le rappel au détriment de la précision.

Dans le contexte critique de prédition des décès résultant d'accidents routiers, un F1-score classe 2 maximum de 20% est loin d'être idéal. Néanmoins, même avec un score aussi bas, le modèle peut s'avérer instructif en mettant en lumière certains scénarios ou régions particulièrement risqués qui pourraient autrement passer inaperçus. Pour approfondir notre compréhension de ces situations, l'utilisation de SHAP serait bénéfique.

2.4 Algorithme KNN

2.4.1 Présentation de l'algorithme

L'intuition derrière l'algorithme des K plus proches voisins est l'une des plus simples de tous les algorithmes de Machine Learning supervisé :

- Étape 1 : Sélectionnez le nombre K de voisins
- Étape 2 : Calculez la distance (euclidienne, Manhattan...) du point non classifié aux autres points.
- Étape 3 : Prenez les K voisins les plus proches selon la distance calculée.
- Étape 4 : Parmi ces K voisins, comptez le nombre de points appartenant à chaque catégorie.
- Étape 5 : Attribuez le nouveau point à la catégorie la plus présente parmi ces K voisins.
- Étape 6 : Notre modèle est prêt

Fondements théoriques

L'algorithme KNN figure parmi les plus simples algorithmes d'apprentissage artificiel. Dans un contexte de classification d'une nouvelle observation x , l'idée fondatrice simple est de faire voter les plus proches voisins de cette observation. La classe de x est déterminée en fonction de la classe majoritaire parmi les k plus proches voisins de l'observation x .

La méthode KNN est donc une méthode à base de voisinage, non-paramétrique ; Ceci signifiant que l'algorithme permet de faire une classification sans faire d'hypothèse sur la fonction $y=f(x_1, x_2, \dots, x_p)$ qui relie la variable dépendante aux variables indépendantes.

Avantages et limites

Avantages :

L'algorithme est simple et facile à mettre en œuvre.

Il n'est pas nécessaire de créer un modèle, de régler plusieurs paramètres ou de formuler des hypothèses supplémentaires.

L'algorithme est polyvalent. Il peut être utilisé pour la classification ou la régression.

Inconvénients :

L'algorithme devient beaucoup plus lent à mesure que le nombre d'observation et de variables indépendantes augmente.

Une faiblesse de la classification dans sa version de base par vote majoritaire apparaît quand la distribution de classe est asymétrique. C'est-à-dire, des exemples d'une classe plus fréquente tendent à dominer la prédiction de classification du nouvel entrant, car elle serait statistiquement plus fréquente parmi les k plus proches voisins par définition.

2.4.2 Entrainement avec différents métriques de distances

Nous avons testé l'algorithme KNN avec plusieurs métriques de distance :

Distance euclidienne : distance du « vol d'oiseau », la ligne droite. De manière générale, c'est la distance la plus utilisée car elle offre un bon compromis entre la gestion des valeurs faibles et celle des valeurs élevées. Formule : $\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

Distance de Manhattan : Nommée ainsi car elle sert à « mesurer » la distance parcourue par une voiture dans la ville de Manhattan, elle est la plus simple : on additionne la distance parcourue sur chaque axe de l'espace (dans un plan, ce sera (en valeur absolue) la coordonnée en x + la coordonnée en y).

Il y a donc beaucoup de chemins qui ont la même longueur ! Formule : $\sum_{i=1}^n |x_i - y_i|$

Distance de Minkowski : Il s'agit d'une généralisation des distances de Manhattan et euclidienne : on élève à une puissance p notre écart entre les coordonnées ($p = 1$ pour Manhattan et 2 pour Euclidienne), puis on prend la racine p-ième : $\sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p}$. Cette distance « p » est la distance de Minkowski.

Distance de Tchebychev : Lorsque p tend vers l'infini, la distance devient une limite et on parle de distance de Tchebychev : $\lim_{p \rightarrow \infty} \sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p} = \sup_{1 \leq i \leq n} |x_i - y_i|$

Nous avons entraîné un modèle pour chaque distance et nous l'avons comparé à un DummyClassifier.

Nous avons comparé les résultats sur le train et le test.

Résultats :

	Dummy Classifier	Distance euclienne	Distance Minkowski	Distance Manhattan	Distance Chebyshev
Accuracy train	-	0.71	0.71	0.72	0.68
F1-score train	-	Classe 1 : 0.79 Classe 2 : 0.37 Classe 3 : 0.60 Classe 4 : 0.66	Classe 1 : 0.79 Classe 2 : 0.37 Classe 3 : 0.60 Classe 4 : 0.66	Classe 1 : 0.81 Classe 2 : 0.40 Classe 3 : 0.61 Classe 4 : 0.67	Classe 1 : 0.76 Classe 2 : 0.31 Classe 3 : 0.55 Classe 4 : 0.63
Accuracy test	0.42	0.56	0.56	0.58	0.51
F1-score test	Classe 1 : 0.60 Classe 2 : 0 Classe 3 : 0 Classe 4 : 0	Classe 1 : 0.69 Classe 2 : 0.13 Classe 3 : 0.36 Classe 4 : 0.48	Classe 1 : 0.69 Classe 2 : 0.13 Classe 3 : 0.36 Classe 4 : 0.48	Classe 1 : 0.71 Classe 2 : 0.13 Classe 3 : 0.38 Classe 4 : 0.50	Classe 1 : 0.63 Classe 2 : 0.08 Classe 3 : 0.32 Classe 4 : 0.45

Interprétation :

- Chaque modèle KNN est plus performant que le DummyClassifier.
- La comparaison entre les résultats du train et du test ne montre pas d'overfitting.
- Le meilleur modèle pour le F1-score de la classe 2 est Manhattan (13%) ou Minkowski (13%).
- Le meilleur modèle pour l'accuracy global est Manhattan (58%).

2.4.3 Recherche les hyperparamètres

Pour optimiser notre modèle, nous avons utilisé la méthode GridSearchCV. Cette méthode permet de tester systématiquement diverses combinaisons d'hyperparamètres pour identifier celle qui produit le meilleur modèle. Dans notre cas, nous avons considéré les hyperparamètres suivants :

- n_neighbors: [1, 20]
- weights : ['uniform','distance'],
- metric : ['minkowski','euclidean','manhattan']

Résultats :

Rapport de classification :				Rapport de classification :					
precision	recall	f1-score	support	precision	recall	f1-score	support		
1.0	0.993620	1.000000	0.996800	161670.000000	1.0	0.676124	0.811846	0.737795	17964.000000
2.0	0.990745	0.997973	0.994346	9869.000000	2.0	0.277778	0.031934	0.057283	1096.000000
3.0	0.994457	0.995600	0.995028	60002.000000	3.0	0.474693	0.341833	0.397454	6667.000000
4.0	1.000000	0.992102	0.996035	149281.000000	4.0	0.601568	0.573642	0.587273	16587.000000
accuracy	0.996158	0.996158	0.996158	0.996158	accuracy	0.624214	0.624214	0.624214	0.624214
macro avg	0.994706	0.996419	0.995552	380822.000000	macro avg	0.507541	0.439814	0.444951	42314.000000
weighted avg	0.996178	0.996158	0.996158	380822.000000	weighted avg	0.604843	0.624214	0.607540	42314.000000
Sur le train				Sur le test					

Nom de Algo	Hyperparamètres	Accuracy
KNN basique	{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}	Train : 0.99 Test : 0.62

Interprétation :

Le modèle est clairement en overfitting.

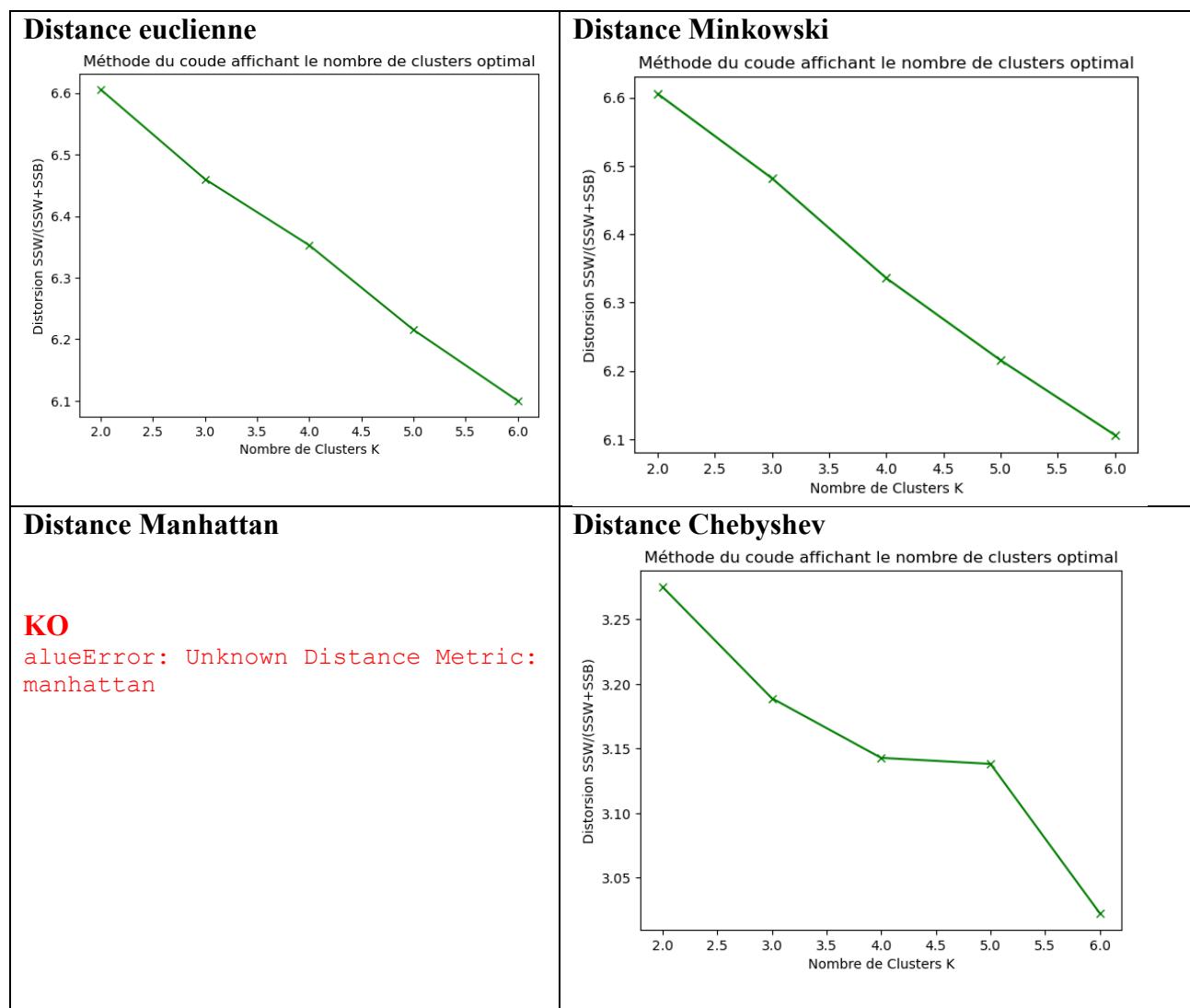
Le modèle a une accuracy sensiblement supérieure aux modèles sans optimisation des paramètres.

2.4.4 Entrainement en apprentissage non supervisé

Nous avons testé l'algorithme KNN sans tenir compte de la variable cible, en apprentissage non supervisé. Nous avons testé sur les 4 metrics de distance.

Nous avons cherché pour une valeur de k entre 1 et 6.

Résultats :



Interprétation :

Seule la distance de Chebyshev fait apparaître un coude autour de la valeur 4.

2.4.5 Gestion de l'échantillonnage

Méthodologie

Face à la sous-représentation de la classe 2 dans nos données, il est impératif d'adopter des stratégies d'équilibrage des classes pour améliorer sa classification. Dans cette optique, nous avons exploré deux techniques principales : le sous-échantillonnage et le suréchantillonnage.

- Sous-échantillonnage :** Cette approche consiste à réduire de manière aléatoire le nombre d'échantillons des classes sur-représentées pour équilibrer l'ensemble d'entraînement.
- Suréchantillonnage :** Ici, nous augmentons le nombre d'échantillons de la classe sous-représentée en utilisant la méthode RandomOverSampler, sans toucher aux échantillons des autres classes.

Résultats :

Nous avons entraîné un modèle oversampling et un modèle undersampling pour chaque metric de distance et nous l'avons comparé à un DummyClassifier.

Nous avons comparé les résultats sur le train et le test.

	Dummy Classifier	Distance euclidienne	Distance Minkowski	Distance Manhattan	Distance Chebyshev
KNN + oversampling					
Accuracy train	-	0.81	0.81	0.82	0.79
F1-score train	-	Classe 1 : 0.79 Classe 2 : 0.96 Classe 3 : 0.83 Classe 4 : 0.62	Classe 1 : 0.79 Classe 2 : 0.96 Classe 3 : 0.83 Classe 4 : 0.62	Classe 1 : 0.81 Classe 2 : 0.97 Classe 3 : 0.84 Classe 4 : 0.64	Classe 1 : 0.76 Classe 2 : 0.95 Classe 3 : 0.82 Classe 4 : 0.60
Accuracy test	0.42	0.51	0.51	0.53	0.44
F1-score test	Classe 1 : 0.59 Classe 2 : 0 Classe 3 : 0 Classe 4 : 0	Classe 1 : 0.68 Classe 2 : 0.14 Classe 3 : 0.38 Classe 4 : 0.40	Classe 1 : 0.68 Classe 2 : 0.14 Classe 3 : 0.37 Classe 4 : 0.40	Classe 1 : 0.70 Classe 2 : 0.14 Classe 3 : 0.39 Classe 4 : 0.42	Classe 1 : 0.61 Classe 2 : 0.10 Classe 3 : 0.32 Classe 4 : 0.36
KNN + undersampling					
Accuracy train	-	0.64	0.64	0.66	0.60
F1-score train	-	Classe 1 : 0.72 Classe 2 : 0.69 Classe 3 : 0.58 Classe 4 : 0.55	Classe 1 : 0.72 Classe 2 : 0.69 Classe 3 : 0.58 Classe 4 : 0.55	Classe 1 : 0.74 Classe 2 : 0.70 Classe 3 : 0.59 Classe 4 : 0.56	Classe 1 : 0.66 Classe 2 : 0.65 Classe 3 : 0.55 Classe 4 : 0.52
Accuracy test	0.42	0.44	0.44	0.46	0.38
F1-score test	Classe 1 : 0.60 Classe 2 : 0 Classe 3 : 0 Classe 4 : 0	Classe 1 : 0.63 Classe 2 : 0.14 Classe 3 : 0.28 Classe 4 : 0.35	Classe 1 : 0.63 Classe 2 : 0.14 Classe 3 : 0.28 Classe 4 : 0.35	Classe 1 : 0.67 Classe 2 : 0.16 Classe 3 : 0.29 Classe 4 : 0.36	Classe 1 : 0.55 Classe 2 : 0.11 Classe 3 : 0.25 Classe 4 : 0.32

Interprétation :

Les modèles oversampling sont légèrement moins bons (autour de 51%) que les modèles par défaut (autour de 56%). Le F1-score de la classe 2 sont légèrement meilleurs (14% au lieu de 13%). Les modèles sont en overfitting.

Les modèles undersampling significativement moins bons (autour de 44%) que les modèles par défaut (autour de 56%). Le F1-score de la classe 2 sont légèrement meilleurs (14% au lieu de 13%). Les modèles sont en overfitting.

Pour optimiser notre modèle, nous avons utilisé la méthode GridSearchCV. Cette méthode permet de tester systématiquement diverses combinaisons d'hyperparamètres pour identifier celle qui produit le meilleur modèle. Dans notre cas, nous avons considéré les hyperparamètres suivants :

- n_neighbors: [1, 20]
- weights : ['uniform','distance'],
- metric : ['minkowski','euclidean','manhattan']
-

Nom de Algo	Hyperparamètres	Accuracy
KNN oversampling test 1	Sur dico complet L'algorithme n'a jamais pu finir de s'exécuter	
KNN oversampling test 2	Test sur dico réduit avec juste recherche de k entre [1,10] L'algorithme n'a jamais pu finir de s'exécuter	
KNN undersampling	{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}	Train : 0.99 Test : 0.51

Interprétation :

Le modèle under sampling est clairement en overfitting.

Le modèle a une accuracy inférieure (51%) au modèle avec optimisation des paramètres (62%).

2.4.6 Synthèse des résultats

Le meilleur algorithme est :

Nom de Algo	Hyperparamètres	Accuracy
KNN basique	{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}	Train : 0.99 Test : 0.62

2.5 Algorithme deep learning

2.5.1 Entrainement et résultats

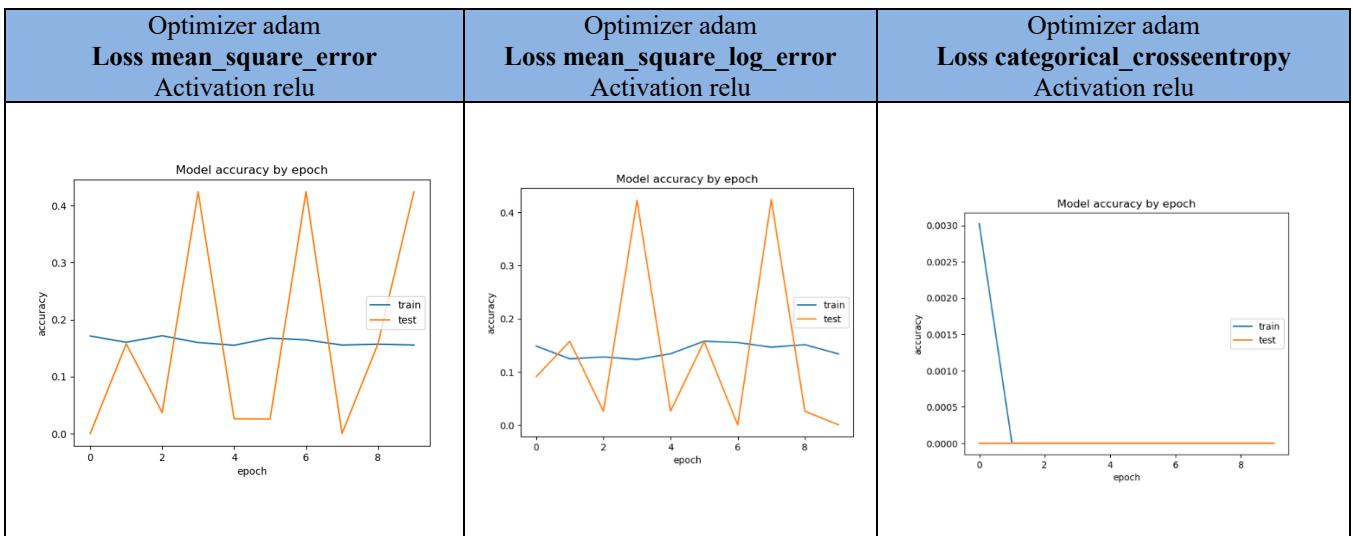
Nous avons testé un algorithme de deep learning avec la bibliothèque keras.

Nous avons un réseau de 25 couches.

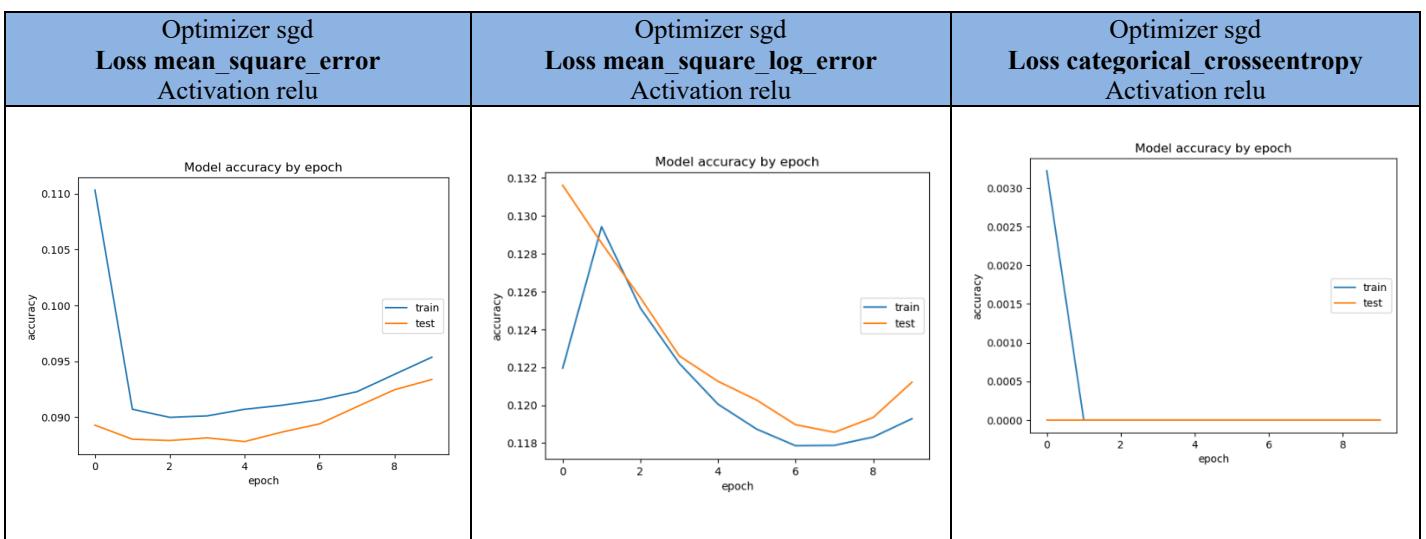
Nous l'avons entraîné sur 10 epochs et avec un batch_size de 50.

Nous avons d'abord entraîné l'algorithme avec l'optimizer adam et nous avons testé 3 loss function différentes :

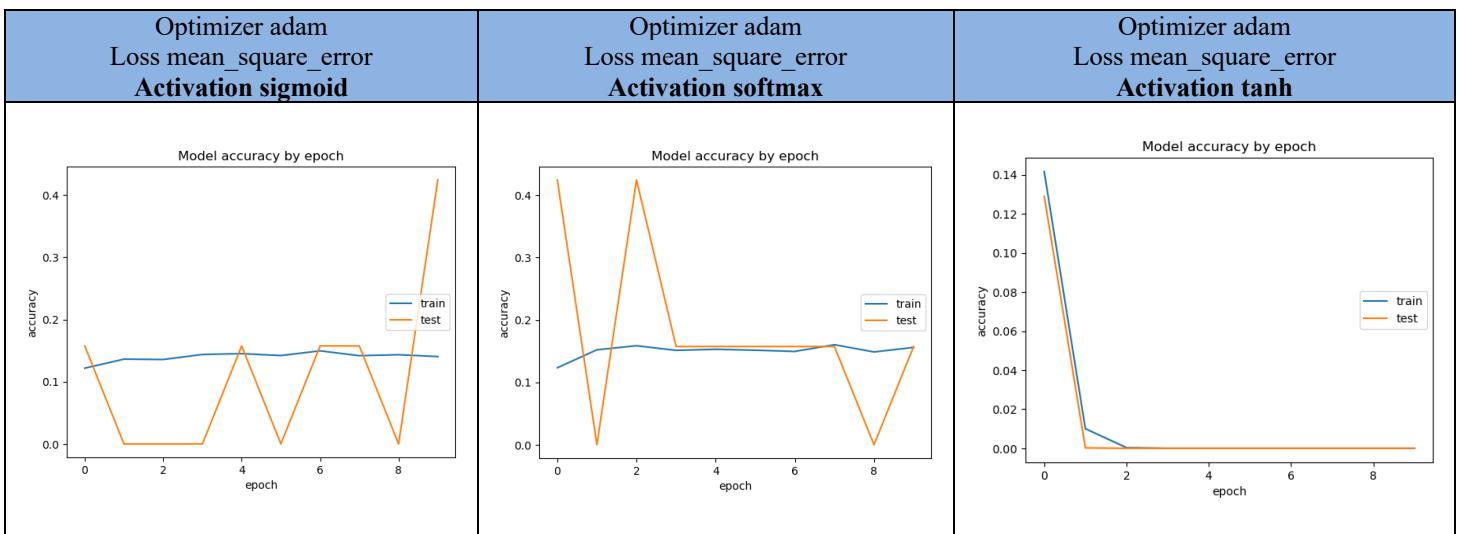
Les résultats sont peu probants et instables.



Nous avons d'abord entraîné l'algorithme avec l'optimizer sgd et nous avons testé 3 loss function différentes :



Nous avons d'abord entraîné l'algorithme avec l'optimizer adam et nous avons testé 3 fonction d'activation différentes :



2.6 Comparaison des Résultats

Dans cette section, nous comparons les performances des différents algorithmes de classification utilisés pour prédire la gravité des accidents dans le cas multiclass. Nous évaluons les modèles sur l'ensemble de test.

Algorithm	Précision classe_2	Recall classe_2	f1-score classe_2	f1-score classe_3	f1-score macro	Sur-apprentissage
Random Forest	0.18	0.45	0.26	0.44	0.51	non
Arbre de décision	0.15	0.12	0.13	0.37	0.44	oui
Logistique régression	0.121	0.648	0.205	0.358	0.466	oui
KNN	0.27	0.03	0.05	0.39	0.62	oui

Les performances de chaque algorithme sont évaluées en termes de précision classe_2, de rappel classe_2, de F1-score classe_2, de F1-score classe_3 et du score F1-macro global. Nous examinons également la possibilité de surapprentissage pour chaque modèle. Les meilleurs algorithmes sont les suivants : Random Forest

3. Étude du Problème de Classification Binaire

Pour rappel, ci-dessous est présentée la répartition de la variable cible dans le cas binaire :

Repartition de la variable cible par default en [%]

	Pas Grave	Graves
Pas grave	81.7	0.0
Graves	0.0	18.3

1 – Indemne , 4 – Blessé léger - categorie 0 : Pas grave
2 – Tué , 3 – Blessé hospitalisé - categorie 1 : Grave

3.1 Algorithme Random Forest

L'algorithme de Random Forest a été présenté dans la partie 2.1

3.1.1 Gestion de l'échantillonnage

Comme cela a été expliqué dans le paragraphe 2.1.1 pour le cas multiclasse, le déséquilibre dans un jeu de données peut entraîner des biais dans les prédictions. Par conséquent, nous avons testé différentes méthodes d'échantillonnage afin de les appliquer ultérieurement dans la recherche d'hyperparamètres.

Voici un résumé des méthodes de rééchantillonnage que nous avons appliquées, ainsi que les résultats associés :

	Method	Nom	MCC Train Score	MCC Test Score	F1 Train Score	F1 Test Score
0	Méthode 0	Sans équilibrage	0.294818	0.295240	0.266626	0.270959
1	Méthode 1	Balanced Subsample	0.429528	0.433874	0.537248	0.540861
2	Méthode 2	Random OverSampling	0.541960	0.429016	0.777398	0.536319
3	Méthode 3	Random UnderSampling	0.531346	0.430545	0.773128	0.537356
4	Méthode 4	Stratified K-Fold	0.293863	NaN	0.266666	NaN
5	Méthode 5	Balanced	0.428728	0.435856	0.536835	0.541854

Les hyperparamètres de RF sont par défaut sauf '`max_depth`' = 10. Nous avons utilisé `cross_validate` avec `cv=5`.

Hyper paramètres

	Valeur
bootstrap	True
<code>ccp_alpha</code>	0.0
<code>class_weight</code>	None
<code>criterion</code>	gini
<code>max_depth</code>	10
<code>max_features</code>	sqrt
<code>max_leaf_nodes</code>	None
<code>max_samples</code>	None
<code>min_impurity_decrease</code>	0.0
<code>min_samples_leaf</code>	1
<code>min_samples_split</code>	2
<code>min_weight_fraction_leaf</code>	0.0
<code>n_estimators</code>	100
<code>n_jobs</code>	None
<code>oob_score</code>	False
<code>random_state</code>	None
<code>verbose</code>	0
<code>warm_start</code>	False

Les méthodes "Balanced" (Meth_5) et « Balanced Subsample » (Meth_1) ont été choisies comme les méthodes de gestion de l'échantillonnage les plus appropriées. Nous les avons sélectionnées en raison de leur capacité à équilibrer les classes minoritaires tout en évitant potentiellement le surapprentissage.

3.1.2 Recherche les hyperparamètres

Pour réaliser la recherche des meilleurs hyperparamètres, nous avons appliqué la méthode `RandomizedSearchCV`, qui est une technique d'optimisation des hyperparamètres basée sur la recherche aléatoire.

Cette méthode explore de manière efficace l'espace des hyperparamètres en effectuant une sélection aléatoire parmi un ensemble de combinaisons possibles, ce qui nous permet de trouver rapidement des paramètres optimaux pour notre modèle.

Nous avons utilisé diverses métriques pour évaluer de la performance. Nous avons examiné MCC, la précision, le rappel, le F1-score, l'AUC-ROC et d'autres métriques courantes en classification. De plus, nous avons introduit deux métriques personnalisées, le F-score beta2 et le F-score beta3, mettant l'accent sur le rappel. Ces métriques étaient adaptées à notre cas où la détection des cas positifs est importante.

Nous avons également créé une métrique personnalisée, 'Cust_metric', combinant rappel et précision de manière spécifique.

L'objectif était d'évaluer différents ensembles d'hyperparamètres de manière approfondie et de choisir ceux qui convenaient le mieux à notre tâche de prédiction.

```

def f_score_beta2(y_true, y_pred):
    return fbeta_score(y_true, y_pred, beta=2, pos_label=1)

def f_score_beta3(y_true, y_pred):
    return fbeta_score(y_true, y_pred, beta=3, pos_label=1)

def my_metric(y_true, y_pred):
    recall_pos = recall_score(y_true, y_pred, pos_label=1)
    precision_pos = precision_score(y_true, y_pred, pos_label=1)

    my_score = (2*recall_pos + precision_pos) / 3 * 100 # Normalisation à 100%
    return my_score

scoring = {
    'MCC': make_scorer(matthews_corrcoef),
    'Recall': make_scorer(recall_score),
    'Precision': make_scorer(precision_score, pos_label=1), # positive Precision
    'F1': make_scorer(f1_score),
    'AUC': make_scorer(roc_auc_score),
    'Fscore_beta2': make_scorer(f_score_beta2),
    'Fscore_beta3': make_scorer(f_score_beta3),
    'Cust_metric': make_scorer(my_metric),
}

```

Voici la liste des hyperparamètres à tester lors de la recherche (GridSearchCV). Nous avons utilisé `cross_validate` avec `cv=3`:

```

param_dist = {
    'n_estimators': [100, 300, 600, 1000, 1500, 2000, 3000],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': ['balanced_subsample'],
    'bootstrap': [True],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy'],
    'ccp_alpha': [0.0, 0.01]
}

```

Exemple d'affichage des hyperparamètres avec leurs métriques :

# Affichage des résultats de gridsearch_cv									
Python									
	params	mean_test_MCC	mean_test_Recall	mean_test_Precision	mean_test_F1	mean_test_AUC	mean_test_Fscore_beta2	mean_test_Fscore_beta3	mean_test_Cust_metric
144	{'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.429981	0.791255	0.406927	0.537450	0.766093	0.665537	0.722971	99.471884
81	{'n_estimators': 600, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.429797	0.791851	0.406533	0.537243	0.766080	0.665662	0.723293	99.511789
136	{'n_estimators': 600, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 10, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'ccp_alpha': 0.0, 'bootstrap': True}	0.429257	0.790837	0.406430	0.536922	0.765687	0.665036	0.722501	99.405178

Les numéros de combinaisons d'hyperparamètres qui maximisent chaque métrique sont présentés ci-dessous :

```

Index de valeur max de MCC --- 144
Index de valeur max de AUC --- 144
Index de valeur max de mean_test_F1 --- 144
Index de valeur max de mean_test_F2 --- 45
Index de valeur max de mean_test_F3 --- 27
Index de valeur max de myscore --- 27

```

L'objectif est d'étudier comment chaque combinaison de paramètres impacte la prédiction. Nous avons identifié trois combinaisons d'hyperparamètres : la combinaison 144, qui maximise MCC, AUC et F1, la combinaison 27, qui maximise Score_personnalisé et F3, ainsi que la combinaison 45, qui maximise F2.

mean_test_MCC	0.428796
mean_test_Recall	0.793188
mean_test_Precision	0.405116
mean_test_F1	0.536312
mean_test_AUC	0.765766
mean_test_Fscore_beta2	0.665656
mean_test_Fscore_beta3	0.723848
mean_test_Cust_metric	99.574566
Name: 27, dtype: object	

Liste des scores pour la combinaison 27

mean_test_MCC	0.429069
mean_test_Recall	0.792962
mean_test_Precision	0.405448
mean_test_F1	0.536551
mean_test_AUC	0.765871
mean_test_Fscore_beta2	0.665708
mean_test_Fscore_beta3	0.723784
mean_test_Cust_metric	99.568627
Name: 45, dtype: object	

Liste des scores pour la combinaison 45

mean_test_MCC	0.429981
mean_test_Recall	0.791255
mean_test_Precision	0.406927
mean_test_F1	0.53745
mean_test_AUC	0.766093
mean_test_Fscore_beta2	0.665537
mean_test_Fscore_beta3	0.722971
mean_test_Cust_metric	99.471884
Name: 144, dtype: object	

Liste des scores pour la combinaison 144

On peut observer que les scores varient de quelques centièmes voire millièmes en fonction de chaque combinaison d'hyperparamètres. Les rapports de classification sur l'ensemble de données de test sont présentés pour évaluer l'impact de chaque combinaison d'hyperparamètres.

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport de classification pour la combinaison 144

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport de classification pour la combinaison 27

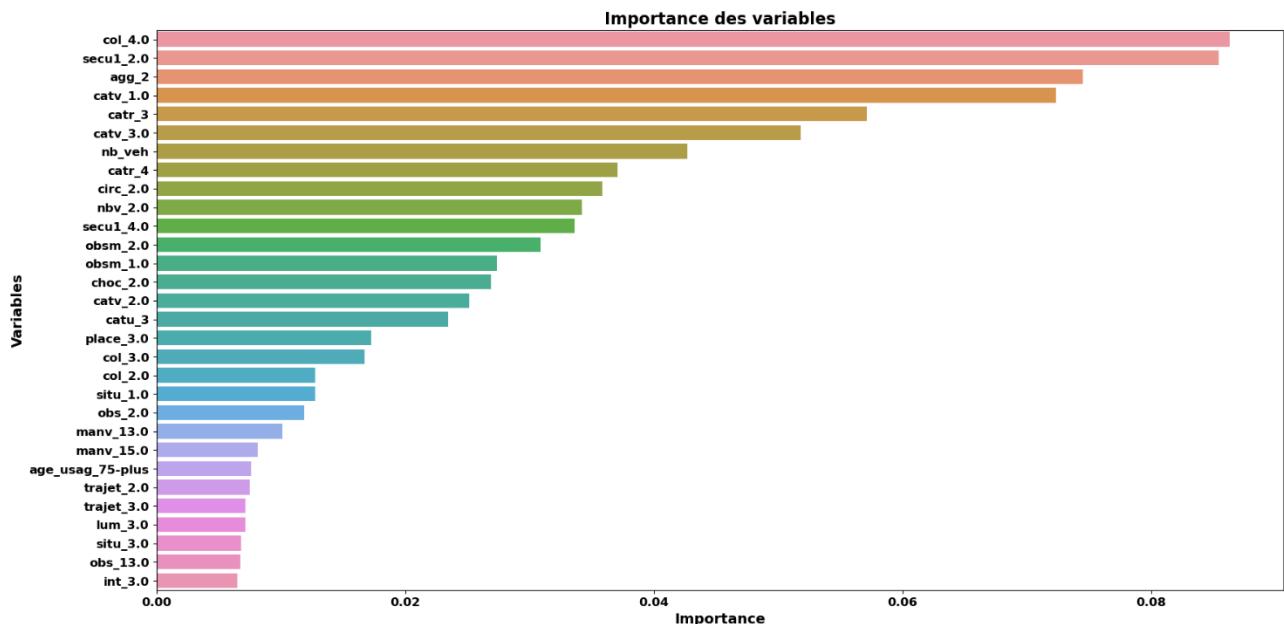
Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.68	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport de classification pour la combinaison 45

Comme on peut le constater, les rapports de classification sont identiques, ce qui signifie que les différences de score sont tellement minimes qu'elles n'ont pas d'impact significatif sur les résultats de l'ensemble de données de test.

3.1.3 Importance des caractéristiques avec Random Forest

Nous avons exploré l'importance des caractéristiques dans notre modèle en utilisant l'attribut de la méthode Random Forest. Pour visualiser cette importance, nous avons généré un graphique affichant les 30 premières caractéristiques les plus importantes. Le graphique présente les caractéristiques sur l'axe des ordonnées et leur importance sur l'axe des abscisses.



De plus, nous avons calculé l'importance de chaque caractéristique et les avons triées de manière descendante. Sur un total de 181 caractéristiques, nous avons constaté que 22 d'entre elles avaient une importance supérieure à 0,01. Ces caractéristiques jouent un rôle significatif dans notre modèle, contribuant ainsi de manière significative à la prédiction des résultats. Les dix caractéristiques les plus importantes sont énumérées ci-dessous, avec leur niveau d'importance respectif :

Nb de features total : 181
Nb de features avec importance > 0.01 : 22

	Features	Importance
23	col_4.0	0.086302
133	secul1_2.0	0.085436
14	agg_2	0.074484
67	catv_1.0	0.072341
25	catr_3	0.057123
69	catv_3.0	0.051817
0	nb_veh	0.042729
26	catr_4	0.037115
28	circ_2.0	0.035847
32	nbv_2.0	0.034208

Ces résultats mettent en évidence les caractéristiques qui ont le plus d'influence sur notre modèle de prédiction.

Pour évaluer l'importance des caractéristiques, nous avons entraîné des modèles en utilisant respectivement 10, 30 et 35 des caractéristiques les plus importantes, en les comparant aux modèles utilisant les 181 caractéristiques initiales.

Ci-dessous les rapports de classification des modèles sur l'ensemble de données de test. Ce qui est particulièrement intéressant à observer, c'est qu'à mesure que nous avons augmenté le nombre de caractéristiques utilisées dans nos modèles, nous avons pu constater une amélioration des scores de classification. Cependant, cette amélioration a atteint un plateau à partir de l'utilisation de 35 caractéristiques. Autrement dit les 35 caractéristiques sélectionnées capturent efficacement les informations essentielles pour notre tâche de prédiction, et l'inclusion d'autres caractéristiques n'apporte pas de bénéfices substantiels en termes de précision de la prédiction.

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.92	0.74	0.82	69089
1.0	0.38	0.70	0.49	15539
accuracy			0.73	84628
macro avg	0.65	0.72	0.65	84628
weighted avg	0.82	0.73	0.76	84628

Rapport classification pour le model avec **10 caractéristiques**

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.73	0.82	69089
1.0	0.40	0.80	0.53	15539
accuracy			0.74	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.74	0.77	84628

Rapport classification pour le model avec **30 caractéristiques**

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport classification pour le model avec **35 caractéristiques**

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport classification pour le model avec **181 caractéristiques**

3.1.4 Résultats

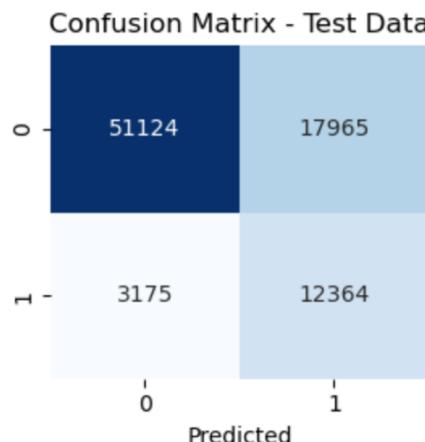
Ci-dessous, les résultats de la meilleure combinaison d'hyperparamètres sont présentés pour le dataset d'entraînement et le dataset de test, ainsi que la matrice de confusion :

Rapport de classification - Données d'entraînement :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	276413
1.0	0.41	0.80	0.54	62095
accuracy			0.75	338508
macro avg	0.68	0.77	0.69	338508
weighted avg	0.85	0.75	0.78	338508

Rapport classification sur dataset train pour combiniions hyperparamètres 144

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

Rapport classification sur dataset Test pour combiniions hyperparamètres 144



3.2 Algorithme Arbre de décision

3.2.1 Gestion de l'échantillonnage

Dans cette section, on regarde comment optimiser le F1 score macro en passant d'une classification multiclass à une classification binaire. Pour se faire, on effectue le regroupement en deux classes : 0 = pas grave, 1 = grave.

Ainsi, les anciennes classes « blessé grave » + « décès » deviennent la classe 1, et « indemne » + « blesser léger » deviennent la classe 0. La fusion de ces classes apporte un premier rééquilibrage des deux classes présentes, avec suffisement d'entrées dans chacune d'elle pour utiliser rééquilibrage complet via **Undersampling**. Cet Undersampling nous permettra d'avoir des temps d'exécution plus raisonnables que par Oversampling.

3.2.2 Recherche les hyperparamètres

Nous utiliserons à nouveau le GridsearchCV avec les paramètres suivants :

```
param_grid = {  
    "criterion": ["gini", "entropy"],  
    "splitter": ["best", "random"],  
    "max_depth": [None] + [12, 14, 16, 18, 20, 22, 24, 32, 64],  
    "max_features": ['sqrt', 'log2'],  
    "min_samples_split": [1, 2, 4, 8, 16, 32, 64, 128, 256],  
    "min_samples_leaf": [1, 2, 4, 8, 16, 32, 64, 128, 256],  
    "random_state": [128]  
}
```

3.2.3 Résultats

Chapitre : 5.0 - GridSearch - 2 classes - Métrique : F1 macro

On considère la métrique F1 macro, dans le cas de classification binaire, afin de pouvoir la comparer à celle obtenue lors de la classification multiclass.

Le gridsearchCV a sorti les meilleurs paramètres suivants :

```
best_params  
{'criterion': 'entropy',  
 'max_depth': 32,  
 'max_features': 'sqrt',  
 'min_samples_leaf': 1,  
 'min_samples_split': 256,  
 'random_state': 128,  
 'splitter': 'random'}
```

Résultats

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
0.0	0.78	0.73	0.75	0.93	0.72	0.81
1.0	0.75	0.79	0.77	0.38	0.77	0.51
accuracy			0.76			0.73
macro avg	0.76	0.76	0.76	0.66	0.75	0.66
weighted avg	0.76	0.76	0.76	0.83	0.73	0.76

On remarque une nette amélioration du F1-score entre la classification multiclasse et la classification binaire. Le DecisionTreeClassifier permet de meilleurs résultats en binaire, en passant de 0,45 à 0,66. Soit une amélioration significative d'environ 46%. Néanmoins, cette valeur donne un résultat encore assez décevant de 0,66, et si l'on regarde en détail nous avons encore un phénomène d'overfitting assez prononcé puisque Train est à 0,76. Egalement, si l'entraînement donne des résultats équilibrés pour chacun des deux classes, un déséquilibre apparaît pour la classe 1 lors de la phase de Test : la précision de la prédiction est plus faible que pour la classe 0, le recall et F1 sont assez éloignés.

Conclusions du DecisionTreeClassifier

L'algorithme a beaucoup mieux prédit les sorties dans le cas binaire que dans le cas multiclasse. Malgré cette amélioration un score de 0,66 n'est pas exceptionnel, et dans le détail une disparité de prédiction persiste entre les classes. La classe la moins bien prédite était encore la classe minoritaire qui a été rééquilibrée par Undersampling. Enfin, malgré un rééquilibrage par undersampling l'overfitting persiste, et cela compte tenu du fait du nombre important de features et que l'arbre reste assez profond. Là encore, il conviendrait de tester l'élagage/pruning afin de voir comment cela pourrait améliorer le scoring.

3.3 Algorithme Logistique régression

3.3.1 Entraînement du modèle avec les 5 solveurs et les hyperparamètres par défauts.

Méthodologie

Pour garantir la robustesse et la performance de notre modèle de régression logistique, nous allons tester les cinq différents solveurs.

Il est essentiel de tester ces différents solveurs car leurs performances peuvent varier en fonction de la nature et de la taille de l'ensemble de données.

Nous avons entraîné le modèle de régression logistique avec les configurations par défauts pour évaluer les performances, en particulier pour la classe 1, sur les ensembles de test et d'entraînement. Ces évaluations ont été réalisées selon une méthode directe, sans stratégie d'échantillonnage spécifique, et avec une validation croisée en trois plis. Ci-après, un détail des performances obtenues pour chaque solveur en mettant l'accent sur la classe 1 :

Résultat

Les performances détaillées sont les suivantes :

Id	Méthode	échantillonage	CV	Hyperparamètres							test					
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
1	Simple	None	3	newton-cg	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849
2	Simple	None	3	lbfgs	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849
3	Simple	None	3	liblinear	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849
4	Simple	None	3	sag	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849
5	Simple	None	3	saga	1,000	1000	ovr	l2		None	0,631	0,350	0,450	0,679	0,843	0,849
	Max										0,631	0,350	0,450	0,679	0,843	0,849

En analysant les métriques relatives à la classe 1, nous observons qu'aucun solveur ne se démarque nettement. Le meilleur score F1 pour la classe 1 avec le solveur Liblinear est de 0,450, et le score F1 global atteint 0,679. Il est clair que tous les solveurs ont du mal à identifier précisément la classe 1, comme le montrent les scores de rappel et F1.

Conclusion

Les métriques de performance pour la classe 1 sont, dans l'ensemble, peu convaincantes, quel que soit le solveur utilisé. Une priorité serait d'ajuster les hyperparamètres. En effet, la performance d'un modèle de machine learning, et en particulier la régression logistique, peut être grandement affectée par le choix des hyperparamètres. Ainsi, la prochaine étape consistera à utiliser GridSearchCV pour déterminer la combinaison idéale d'hyperparamètres pour chaque solveur. Une telle démarche pourrait contribuer à améliorer la situation.

La plus faible représentation de la classe 1 dans nos données pourrait être à l'origine de ces difficultés de classification. De plus, il semble que les attributs actuels ne soient pas suffisants pour la distinguer efficacement. Étant donné l'importance de la classe 1, des techniques telles que le suréchantillonnage, l'ajustement des poids de classe, ou l'introduction de nouvelles caractéristiques pourraient être envisagées pour améliorer les performances. En conclusion, bien qu'il y ait de légères variations entre les solveurs, l'enjeu majeur demeure l'optimisation de la classification de la classe 1, tant sur le plan des hyperparamètres que de la gestion des données.

3.3.2 Recherche des meilleurs hyperparamètres.

Méthodologie

Pour optimiser notre modèle de régression logistique, nous avons utilisé la méthode GridSearchCV. Cette méthode permet de tester systématiquement diverses combinaisons d'hyperparamètres pour identifier celle qui produit le meilleur modèle. Dans notre cas, nous avons considéré les hyperparamètres suivants :

- Pénalité : L1, L2, elasticnet et aucune.
- Solveur : liblinear, saga, newton-cg, lbfgs, sag.
- Nombre maximum d'itérations : 1000, 10000, 100000.
- C : 0.001, 0.01, 0.1, 1, 10.
- multi_class : ovr, multinomial.
- Poids de classe : balanced, None.

Résultats

Suite à l'utilisation de GridSearchCV, nous avons déterminé la meilleure combinaison d'hyperparamètres pour notre modèle de régression logistique :

Id	Méthode	échantillonage	CV	Hyperparamètres								test				
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
6	GridsearchCV	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,798	0,544	0,688	0,755	0,849
	Max										0,413	0,798	0,544	0,688	0,755	0,849

Cette combinaison opte pour le solver "saga" avec une pénalité L2, un C de 1, un maximum de 1000 itérations, une stratégie multiclass "ovr" et un poids de classe défini sur "balanced". Ces paramètres ont été sélectionnés pour optimiser les performances du modèle selon notre métrique personnalisée, qui accorde une importance accrue à la classe 1.

Conclusion

Les performances obtenues avec la meilleure combinaison d'hyperparamètres déduite de GridSearchCV ont été confrontées à celles issues de configurations par défaut. En mettant l'accent sur la classe 1, nous nous sommes principalement appuyés sur l'exactitude, le score F1, le rappel et la précision pour cette classe pour effectuer cette comparaison. Le score F1 pour la classe 1 s'est nettement amélioré, passant de 0,450 à 0,544. Bien que la précision ait légèrement baissé, de 0,630 à 0,413, le rappel a considérablement augmenté, passant de 0,350 à 0,798. Suite à notre analyse, nous avons déterminé que l'hyperparamètre décisif a été "class_weight" réglé sur "balanced". Ceci a permis de rééquilibrer le poids de la classe 1, compensant ainsi sa sous-représentation. Toutefois, cette compensation est une première étape. Pour affiner notre modèle, nous devons examiner les méthodes d'équilibrage direct des échantillons.

3.3.3 Gestion de l'échantillonage.

Méthodologie

La sous-représentation de la classe 1 dans notre ensemble de données nous incite à envisager des stratégies d'équilibrage des classes pour renforcer sa classification. À cette fin, nous avons expérimenté trois techniques majeures : le sous-échantillonnage, le suréchantillonnage et le smote.

- Sous-échantillonnage : Cette stratégie réduit de manière aléatoire les échantillons des classes majoritaires pour équilibrer l'ensemble d'entraînement.

- Suréchantillonnage : À l'inverse, cette méthode augmente le nombre d'échantillons de la classe minoritaire en employant la technique RandomOverSampler, sans modifier les échantillons des autres classes.
- SMOTE : Technique d'échantillonnage qui génère des échantillons artificiels pour la classe minoritaire afin d'équilibrer la distribution.

Résultats

Id	Méthode	échantillonage	CV	Hyperparamètres							test					
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
7	GridsearchCV	UnderSamp.	3	lbfgs	1,000	1000	ovr		-		0,413	0,799	0,545	0,689	0,755	0,849
8	RandomsearchCV	Oversamp.	3	liblinear	0,100	100000	ovr	l1	-	balanced	0,413	0,797	0,544	0,689	0,755	0,850
9	RandomsearchCV	SMOTE	3	saga	0,001	10000	ovr	l2	-	balanced	0,474	0,554	0,511	0,695	0,805	0,814
	Max										0,474	0,799	0,545	0,695	0,805	0,850

Conclusion

Les performances des trois modèles obtenus à partir des stratégies de rééchantillonnage sont semblables. Le choix entre ces méthodes devrait donc être basé sur d'autres facteurs, tels que l'efficacité de l'exécution ou les coûts en termes de calcul.

Cependant, en comparant ces modèles au modèle issu de la méthode GridSearch sans rééchantillonnage, on ne constate pas d'amélioration significative.

En effet l'option `class_weight = « balanced »` dans la régression logistique vise à équilibrer les classes déséquilibrées en ajustant les poids selon l'inverse de la fréquence de chaque classe. Ainsi, une classe moins représentée recevra un poids plus important, orientant le modèle à lui donner davantage d'importance durant l'entraînement.

Par conséquent, si l'option `class_weight = « balanced »` est déjà activée, l'ajout d'une stratégie de rééchantillonnage (que ce soit le sous-échantillonnage ou le suréchantillonnage) s'est avéré superflu. La raison en est que la régression logistique compense déjà en interne les déséquilibres de classes.

En complément à ces constatations relatives aux techniques de sous-échantillonnage et de suréchantillonnage, nous envisageons de poursuivre nos explorations avec une approche utilisant le bagging. Le bagging, aussi connu sous le nom de "bootstrap aggregating", est une technique conçue pour renforcer la stabilité et la précision des modèles en les combinant. Une telle approche pourrait nous fournir une solution plus efficace pour classifier la classe 1. La prochaine section approfondira cette exploration.

3.3.4 Ultime technique pour améliorer les résultats : le bagging.

Méthodologie :

Le bagging, ou « Bootstrap Aggregating », est une stratégie d'ensemble où plusieurs sous-ensembles d'échantillons sont créés à partir de l'ensemble d'entraînement, avec remise. Un modèle est entraîné sur chaque sous-ensemble, puis toutes les prédictions sont agrégées pour produire un résultat final. Ici, nous utiliserons le bagging avec l'option `class_weight = « balanced »` pour gérer le déséquilibre des classes.

Résultats

Id	Méthode	échantillonage	CV	Hyperparamètres							test					
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight	Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
10	Bagging	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,797	0,544	0,688	0,755	0,849
	Max										0,413	0,797	0,544	0,688	0,755	0,849

Conclusion

L'analyse comparative des méthodes de Bagging montre des variations peu significatives en termes de score F1, rappel, et précision. Malgré la promesse du bagging pour améliorer la robustesse du modèle, les performances globales restent modestes.

3.3.5 Synthèse des résultats.

Id	Méthode	échantillonage	CV	Test										test					
				Hyperparamètres										Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight									
1	Simple	None	3	newton-cg	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849			
2	Simple	None	3	lbfgs	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849			
3	Simple	None	3	liblinear	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849			
4	Simple	None	3	sag	1,000	1000	ovr	l2		None	0,630	0,350	0,450	0,679	0,843	0,849			
5	Simple	None	3	saga	1,000	1000	ovr	l2		None	0,631	0,350	0,450	0,679	0,843	0,849			
6	GridsearchCV	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,798	0,544	0,688	0,755	0,849			
7	GridsearchCV	UnderSamp.	3	lbfgs	1,000	1000	ovr		-		0,413	0,799	0,545	0,689	0,755	0,849			
8	RandomsearchCV	OverSamp.	3	liblinear	0,100	100000	ovr	l1	-	balanced	0,413	0,797	0,544	0,689	0,755	0,850			
9	RandomsearchCV	SMOTE	3	saga	0,001	10000	ovr	l2	-	balanced	0,474	0,554	0,511	0,695	0,805	0,814			
10	Bagging	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,797	0,544	0,688	0,755	0,849			
Max											0,631	0,799	0,545	0,695	0,843	0,850			

Id	Méthode	échantillonage	CV	Entraînement										Entraînement					
				Hyperparamètres										Précision (classe 1)	recall (classe 1)	F1-score (classe 1)	F1-score (global)	Accuracy (global)	ROC (classe 1)
				Solver	C	max_iter	multi_class	penalty	l1_ratio	Class_weight									
1	Simple	None	3	newton-cg	1,000	1000	ovr	l2		None	0,640	0,351	0,453	0,681	0,845	0,850			
2	Simple	None	3	lbfgs	1,000	1000	ovr	l2		None	0,640	0,351	0,453	0,681	0,845	0,850			
3	Simple	None	3	liblinear	1,000	1000	ovr	l2		None	0,640	0,351	0,453	0,681	0,845	0,850			
4	Simple	None	3	sag	1,000	1000	ovr	l2		None	0,640	0,351	0,453	0,681	0,845	0,850			
5	Simple	None	3	saga	1,000	1000	ovr	l2		None	0,640	0,351	0,453	0,681	0,845	0,850			
6	GridsearchCV	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,796	0,544	0,688	0,755	0,850			
7	GridsearchCV	UnderSamp.	3	lbfgs	1,000	1000	ovr		-		0,757	0,796	0,776	0,770	0,770	0,849			
8	RandomsearchCV	OverSamp.	3	liblinear	0,100	100000	ovr	l1	-	balanced	0,758	0,797	0,777	0,771	0,771	0,851			
9	RandomsearchCV	SMOTE	3	saga	0,001	10000	ovr	l2	-	balanced	0,850	0,813	0,819	0,831	0,834	0,918			
10	Bagging	None	3	saga	1,000	1000	ovr	l2	-	balanced	0,413	0,795	0,544	0,688	0,755	0,849			
Max											0,850	0,813	0,819	0,831	0,845	0,918			

- Précision (classe 1) et Accuracy (global) :

Les modèles 1, 2, 3, 4 et 5 présentent la précision (pour la classe 1) et l'accuracy (global) les plus élevées, avec des valeurs maximales de 0,631 et 0,843 respectivement.

- Rappel (classe 1), F1-score (classe 1), F1-score (global) :

Les modèles 6, 7, 8 et 10, qui utilisent soit GridSearchCV soit RandomSearchCV pour la recherche des hyperparamètres, offrent le rappel, le F1-score (pour la classe 1) et le F1-score (global) les plus élevés. Il faut toutefois noter que les modèles utilisant les techniques d'échantillonnage présentent un risque important d'overfitting. Le modèle 6, utilisant GridSearchCV sans échantillonnage, sera donc préféré. Ses scores sont notamment de 0,544 pour le F1-score de la classe 1 et de 0,688 pour le F1-score global.

Dans le contexte critique de la prédiction des décès résultant d'accidents routiers, un F1-score maximal de 54% pour la classe 1 est loin d'être idéal. Néanmoins, même avec un score aussi bas, le modèle peut s'avérer utile pour identifier des scénarios ou des régions particulièrement à risque qui pourraient autrement passer inaperçus. Pour approfondir notre compréhension de ces situations, l'utilisation de SHAP serait bénéfique.

3.4 Algorithme KNN

3.4.1 Entrainement avec différentes metrics de distance

Nous avons testé l'algorithme KNN avec plusieurs metrics de distance.

- Distance euclidienne
- Distance de Manhattan
- Distance de Minkowski
- Distance de Tchebychev

Nous avons entraîné un modèle pour chaque distance et nous l'avons comparé à un DummyClassifier. Nous avons comparé les résultats sur le train et le test.

Résultats :

	Dummy Classifier	Distance euclienne	Distance Minkowski	Distance Manhattan	Distance Chebyshev
Accuracy train		0.82	0.82	0.83	0.81
F1-score train		Classe 0 : 0.83 Classe 1 : 0.81	Classe 0 : 0.83 Classe 1 : 0.81	Classe 0 : 0.84 Classe 1 : 0.81	Classe 0 : 0.83 Classe 1 : 0.79
Accuracy test	0.42	0.64	0.64	0.66	0.60
F1-score test	Classe 0 : 0.60 Classe 1 : 0	Classe 0 : 0.67 Classe 1 : 0.61	Classe 0 : 0.67 Classe 1 : 0.61	Classe 0 : 0.69 Classe 1 : 0.63	Classe 0 : 0.63 Classe 1 : 0.57

Interprétation :

Chaque modèle KNN est plus performant (64%) que le DummyClassifier (42%).

La comparaison entre les résultats du train et du test montre clairement de l'overfitting.

Le meilleur modèle pour le F1-score de la classe 2 est Manhattan (63%).

Le meilleur modèle pour l'accuracy global est Manhattan (66%).

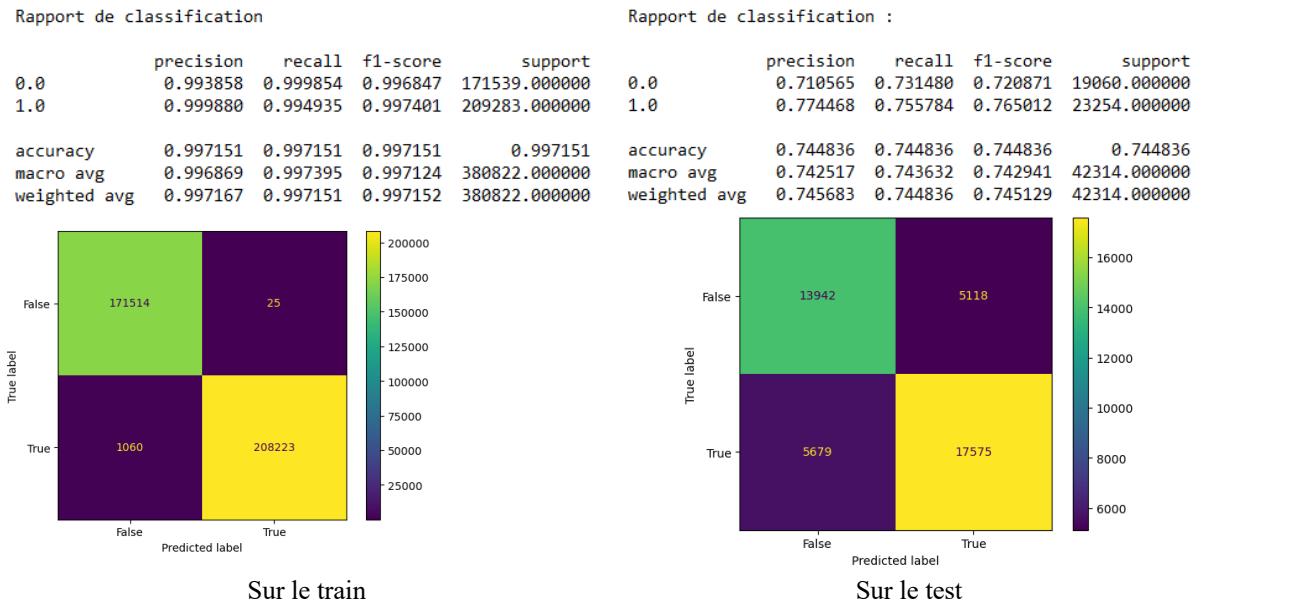
3.4.2 Recherche des hyperparamètres

Pour optimiser notre modèle, nous avons utilisé la méthode GridSearchCV. Cette méthode permet de tester systématiquement diverses combinaisons d'hyperparamètres pour identifier celle qui produit le meilleur modèle. Dans notre cas, nous avons considéré les hyperparamètres suivants :

- n_neighbors: [1, 20]
- weights : ['uniform','distance'],
- metric : ['minkowski','euclidean','manhattan']
-

Résultats :

Nom de Algo	Hyperparamètres	Accuracy
KNN basique	{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}	Train : 0.99 Test : 0.74



Interprétation :

Le modèle est clairement en overfitting.

Le modèle a une accuracy très supérieure (74%) aux modèles sans optimisation des paramètres (autour de 64%).

3.4.3 Synthèse des résultats

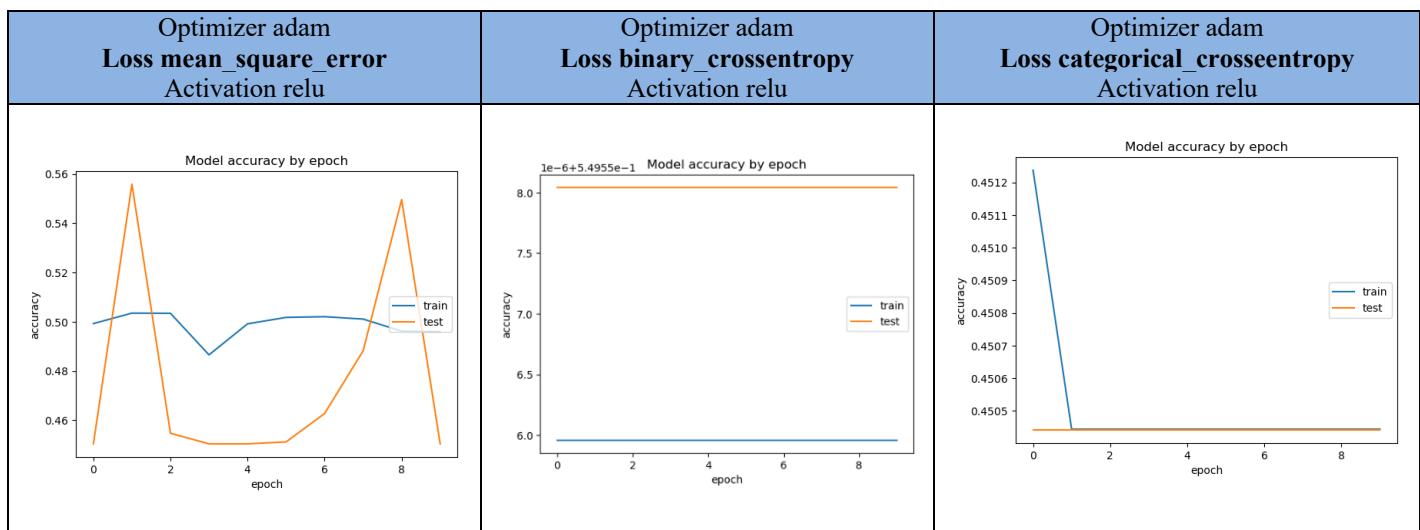
Le meilleur algorithme est :

Nom de Algo	Hyperparamètres	Accuracy
KNN basique	{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}	Train : 0.99 Test : 0.74

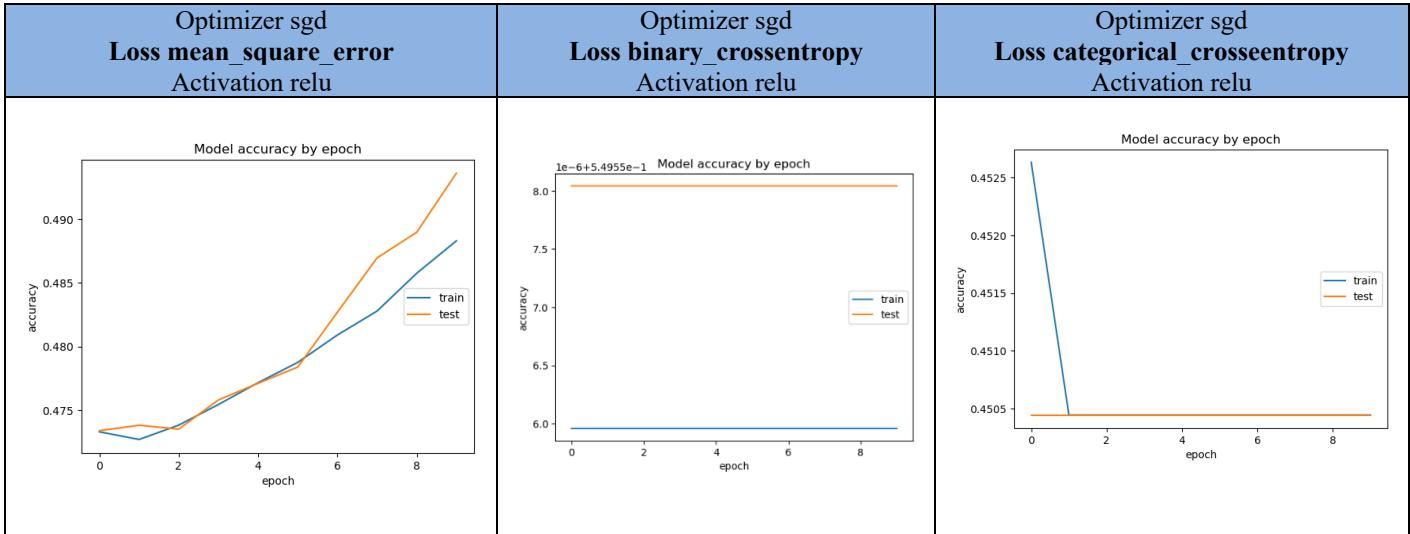
3.5 Algorithme deep learning

3.5.1 Entrainement et résultats

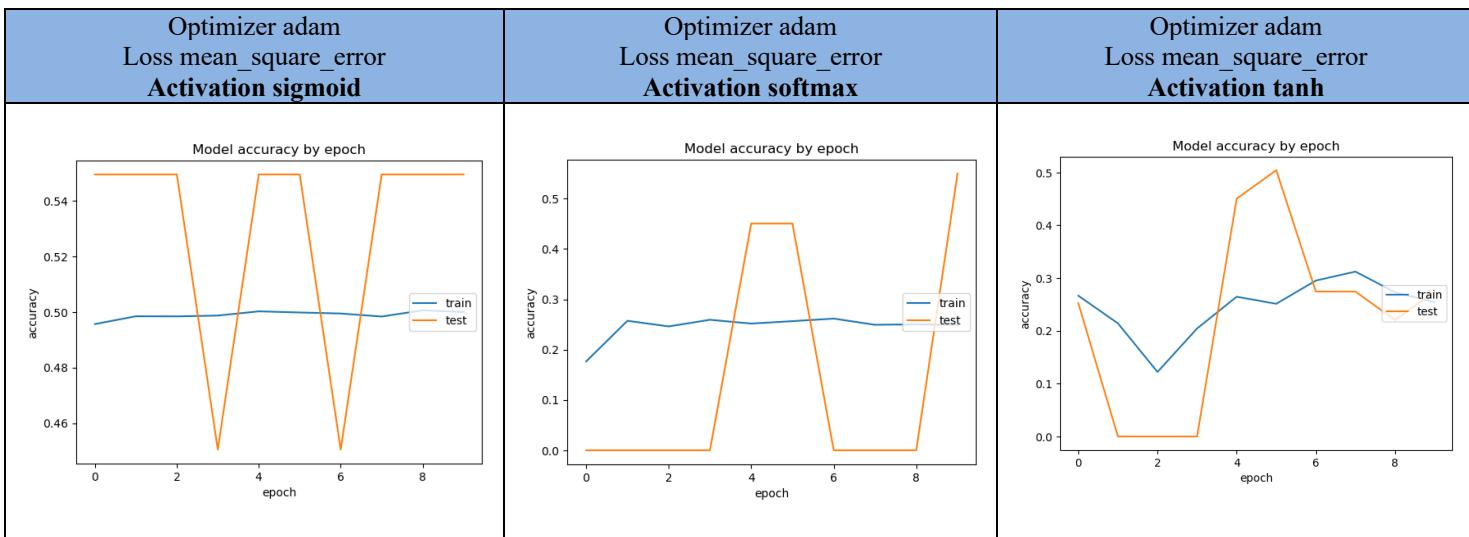
Nous avons testé un algorithme de deep learning avec la bibliothèque keras. Nous avons un réseau de 25 couches. Nous l'avons entraîné sur 10 epochs et avec un batch_size de 50. Nous avons d'abord entraîné l'algorithme avec l'optimizer adam et nous avons testé 3 loss function différentes :



Nous avons d'abord entraîné l'algorithme avec l'optimizer sgd et nous avons testé 3 loss function différentes :



Nous avons d'abord entraîné l'algorithme avec l'optimizer adam et nous avons testé 3 fonction d'activation différentes :



3.6 Algorithme XGBoost

3.6.1 Présentation de l'algorithme

L'algorithme XGBoost (eXtreme Gradient Boosting), est une implémentation efficace et performante de l'algorithme de boosting gradient. Il s'agit d'un algorithme d'apprentissage supervisé.

Principe et spécificités

- XGBoost est basé sur l'idée de boosting gradient, qui consiste à construire séquentiellement des modèles faibles (typiquement des arbres de décision) et à les combiner pour former un modèle puissant. Chaque arbre supplémentaire corrige les erreurs résiduelles du modèle combiné précédent.
- Il introduit un terme de régularisation dans la fonction objectif pour contrôler la complexité du modèle, ce qui aide à prévenir le surajustement (overfitting).

- Il peut gérer les données manquantes de manière native, ce qui permet de construire des modèles robustes même en présence de valeurs manquantes.
- Il est optimisé pour la performance et l'efficacité, tant en termes de temps de calcul que d'utilisation de la mémoire. La construction des arbres peut être parallélisée pour accélérer l'entraînement.

Avantages

- Il est flexible et peut être utilisé pour les problèmes de régression, de classification, de ranking et d'autres types de modélisation.
- Le terme de régularisation aide à prévenir le surajustement, surtout lorsqu'on travaille avec un ensemble de données de petite taille.
- Il permet aux utilisateurs de définir des fonctions objectif et d'évaluation personnalisées.

Inconvénients

- XGBoost est plus complexe à configurer et à régler par rapport à d'autres algorithmes en raison du grand nombre d'hyperparamètres.
- Bien qu'il soit optimisé pour la performance, l'entraînement peut être relativement long sur de grands ensembles de données, surtout si le réglage des hyperparamètres est nécessaire.
- Malgré la régularisation, XGBoost peut parfois surajuster les données, en particulier si les données sont bruitées.
- Il nécessite une attention particulière lors de la configuration des hyperparamètres et peut être gourmand en ressources sur de grands ensembles de données.

Hyperparamètres principaux de XGBoost :

Paramètres généraux

booster	Il spécifie le type de modèle à être exécuté à chaque itération. Il a 3 valeurs : gbtree, gblinear ou dart. gbtree et dart utilisent l'approche d'arbre tandis que gblinear utilise l'approche linéaire
verbosity	Le degré de verbosité pendant l'entraînement. Les valeurs possibles sont 0 (silencieux), 1 (avertissement), 2 (info), 3 (debug).
n_jobs	Le nombre de processus parallèles à exécuter pour l'entraînement.

Paramètres Booster

Paramètres communs aux boosters

n_estimator	Nombre de cycles de 'stimulation' (boost)
learning_rate (ou eta)	Étape de rétrécissement pour éviter le surajustement. Les valeurs varient généralement de 0,01 à 0,3.
gamma	Seuil minimal de la perte requise pour effectuer une partition supplémentaire sur un noeud feuille de l'arbre. Aide à réduire le surajustement
max_depth	Profondeur maximale de l'arbre. Son augmentation améliore le résultat mais conduit à de l'overfitting
reg_lambda (alias lambda)	Terme de régularisation L2 sur les poids > Faute de temps, on ne l'utilisera pas
reg_alpha (alias alpha)	Terme de régularisation L1 sur les poids > Faute de temps, on ne l'utilisera pas
subsample	Fraction de l'ensemble d'entraînement à utiliser pour chaque boosting, évite le surajustement en n'utilisant qu'un sous-ensemble des données pour chaque arbre

Paramètres spécifiques à gbtree et dart

colsample_bytree	Fraction de colonnes à échantillonner pour chaque arbre. Fraction des caractéristiques à utiliser pour chaque arbre, contribue à la randomisation du modèle et réduisant le surajustement
colsample_bylevel	Fraction de colonnes à échantillonner pour chaque niveau de profondeur. > Faute de temps, on ne l'utilisera pas. Réglage très fin.
colsample_bynode	Fraction de colonnes à échantillonner pour chaque noeud.

	> Faute de temps, on ne l'utilisera pas. Réglage très fin.
--	--

Paramètres spécifiques à dart

sample_type	Type de technique d'échantillonnage. Il peut être «uniforme» ou «pondéré».
normalize_type	Type de normalisation à utiliser. Il peut être «arbre» ou «forêt».
rate_drop	Part de dropout.

Paramètres spécifiques à gblinear

feature_selector	Méthode de sélection des fonctionnalités pour le booster linéaire
------------------	---

Paramètres d'apprentissage

objective	Spécifie la fonction de perte à minimiser. Il y a plusieurs choix disponibles tels que reg:squarederror pour la régression et binary:logistic
eval_metric	Métrique d'évaluation à utiliser pour le processus de validation

Autres paramètres

random_state	La graine utilisée pour générer la randomisation.
--------------	---

3.6.2 Gestion de l'échantillonnage

Nous sommes toujours dans un cas de classification binaire avec un déséquilibre que l'on traite de la même façon que lors de l'utilisation de l'algorithme DecisionTreeClassifier : un undersampling. Le but étant de rééquilibrer les classes et aussi de ne pas plomber les performances de l'algorithme qui, bien qu'optimisé, reste somme toute assez long à exécuter.

3.6.3 Recherche des hyperparamètres

Nous avons ainsi défini nos hyperparamètres pour le gridsearchCV :

```

booster = gbtree          arbre
verbosity = 2              info
n_jobs = -1                tous les process
learning_rate ou eta      [0.01, 0.1, 0.3]
max_depth                  [4, 8, 16, 32, 64]
gamma                      [0, 0.1, 0.2]
subsample                   [0.5, 0.7, 1.0]
colsample_bytree            [0.5, 0.7, 1.0]
objective = binary:logistic classification binaire
eval_metric = logloss
random_state = 42           Standard

```

3.6.4 Résultats

Chapitre : 6. - XGBOOST - binaire sans rééchantillonnage - Métrique : F1 macro

best_params

```

{'colsample_bytree': 1.0,
'gamma': 0.2,
'learning_rate': 0.1,
'max_depth': 16,
'min_child_weight': 10,

```

```
'n_estimators': 200,
'subsample': 0.7}
```

Résultats

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
0.0	0.90	0.97	0.93	0.88	0.95	0.91
1.0	0.79	0.53	0.63	0.65	0.42	0.51
accuracy			0.89			0.85
macro avg	0.84	0.75	0.78	0.76	0.68	0.71
weighted avg	0.88	0.89	0.88	0.84	0.85	0.84

Si l'on compare avec les résultats obtenus pour le DecisionTree, on remarque bonne amélioration du F1-score qui passe de 0,66 à 0,71 tout en réduisant le phénomène d'overfitting.

Chapitre : 7. - XGBOOST - binaire avec undersampling - Métrique : F1 macro

best_params

```
{'colsample_bytree': 0.7,
'gamma': 0,
'learning_rate': 0.1,
'max_depth': 8,
'min_child_weight': 5,
'n_estimators': 200,
'subsample': 0.7}
```

Résultats

	Train			Test		
	precision	recall	f1-score	precision	recall	f1-score
0.0	0.83	0.77	0.80	0.95	0.75	0.84
1.0	0.79	0.84	0.82	0.42	0.82	0.56
accuracy			0.81			0.76
macro avg	0.81	0.81	0.81	0.69	0.78	0.70
weighted avg	0.81	0.81	0.81	0.85	0.76	0.78

Si l'on compare avec les résultats obtenus pour le XGBoost précédent, on remarque très légère dégradation du F1-score, avec une légère augmentation de l'overfitting. Assez curieusement l'undersampling et donc la réduction du déséquilibre de classes n'a pas permis d'améliorer les performances du XGBoost, mais a légèrement dégrader le F1-score et l'overfitting.

Conclusions du XGBoost

Le XGBoost confirme sa réputation de sensible et complexe, il nécessite un temps supérieur à d'autres pour pouvoir le maîtriser. Il comporte de nombreux hyperparamètres dont le fine tuning n'est pas évident, et qui peut consommer un temps important. Il est aussi remarquable qu'un rééquilibrage de classe n'apporte pas nécessairement de meilleures performances de prédictions. Cet algorithme garde une certaine sensibilité à l'overfitting comme le DecisionTree, mais offre en revanche des résultats de prédiction bien meilleures.

3.7 Comparaison des Résultats

Dans cette section, nous comparons les performances des différents algorithmes de classification utilisés pour prédire la gravité des accidents dans le cas binaire. Nous évaluons les modèles sur l'ensemble de test.

Algorithm	Positive précision	Positive recall	Negative f1-score	Positive f1-score	F1-macro	Sur-apprentissage
Random Forest	0.41	0.80	0.83	0.54	0.68	non
Arbre de décision	0.38	0.77	0.81	0.51	0.66	oui
Logistique régression	0.41	0.798	0.831	0.544	0.688	non
KNN	0.71	0.73	0.76	0.72	0.74	oui
XGBoost	0.65	0.42	0.91	0.51	0.71	oui

Les performances de chaque algorithme sont évaluées en termes de précision positive, de rappel positif, de F1-score négatif, de F1-score positif et du score F1-macro global. Nous examinons également la possibilité de surapprentissage pour chaque modèle. Les meilleurs algorithmes sont les suivants : Random Forest et Logistique régression.

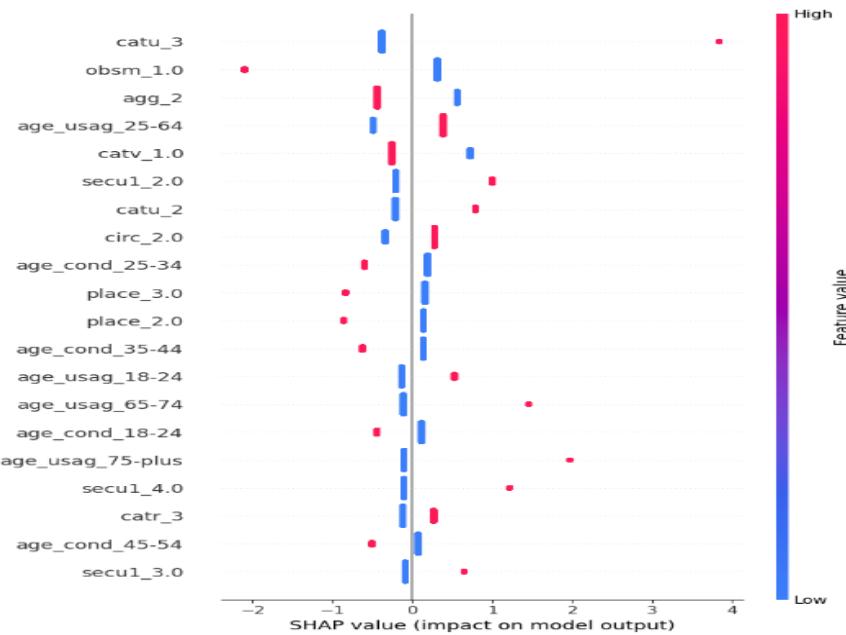
3.8 Exploration de l'Importance des Caractéristiques via SHAP dans la Prédiction des Décès Routiers

Dans le contexte critique de la prédiction des décès résultant d'accidents routiers, un F1-score maximal de 54% pour la classe positive est loin d'être idéal. Néanmoins, même avec un score aussi bas, le modèle peut s'avérer utile pour identifier des scénarios ou des caractéristiques particulièrement à risque qui pourraient autrement passer inaperçus. Pour approfondir notre compréhension de ces situations, l'utilisation de SHAP serait bénéfique. SHAP permet de comprendre l'impact de chaque caractéristique sur les prédictions du modèle. SHAP décompose la contribution de chaque caractéristique en deux niveaux principaux : global et local.

3.8.1 SHAP global

Le SHAP global donne une vision d'ensemble de l'impact des caractéristiques sur le modèle. Il permet d'identifier quelles sont les caractéristiques qui ont, en moyenne, le plus d'impact sur les prédictions sur l'ensemble du jeu de données.

Pour notre jeu de données le diagramme SHAP est le suivant :



Comment interpréter le positionnement des points ?

Rouge à Droite :

Si une valeur est placée à droite et colorée en rouge, cela indique généralement que lorsque la valeur de cette caractéristique augmente, la prédiction du modèle augmente également.

Rouge à Gauche :

Si une valeur est placée à gauche et colorée en rouge, cela indique généralement que lorsque la valeur de cette caractéristique diminue, la prédiction du modèle augmente.

Bleu à Droite :

Si une valeur est placée à droite et colorée en bleu, cela indique généralement que lorsque la valeur de cette caractéristique augmente, la prédiction du modèle diminue.

Bleu à Gauche :

Si une valeur est placée à gauche et colorée en bleu, cela indique généralement que lorsque la valeur de cette caractéristique diminue, la prédiction du modèle diminue également.

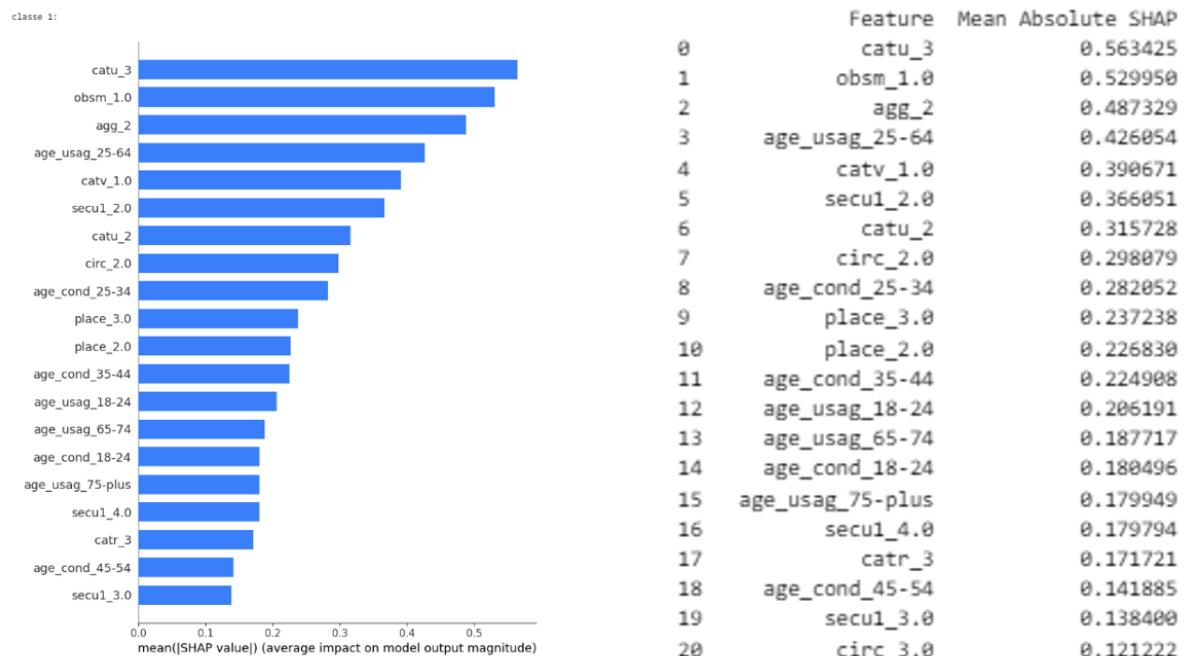
Analyse des résultats

À partir de l'analyse du diagramme SHAP, nous pouvons identifier les variables ayant le plus d'impact sur les prédictions du modèle. Les variables les plus impactantes sont celles relatives à la catégorie d'usager, l'obstacle mobile heurté, la localisation en agglomération ou non, la catégorie du véhicule, la présence d'équipement de sécurité, ainsi que l'âge du conducteur et de l'usager.

Toutefois, il est à noter que des valeurs bleues et rouges sont présentes pour chacune de ces caractéristiques, ce qui reflète la complexité des relations dans les données.

Évaluation de l'impact moyen des caractéristiques

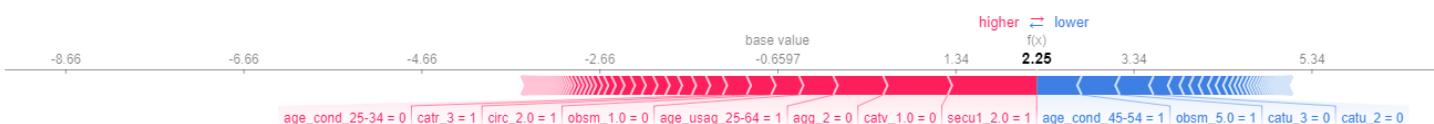
En considérant les valeurs moyennes absolues, nous pouvons obtenir une idée de l'impact moyen de chaque caractéristique sur les prédictions du modèle à travers l'ensemble du jeu de données. L'affichage des moyennes absolues des valeurs SHAP pour chaque caractéristique offre une vision synthétique et globale de l'importance relative de ces caractéristiques. Ceci est utile pour comprendre rapidement quelles sont les caractéristiques les plus influentes selon le modèle, et comment elles contribuent en moyenne à la prédiction, permettant ainsi une quantification de leur impact.



3.8.2 SHAP local

Le SHAP local se concentre sur l'interprétation des prédictions pour des observations individuelles. Il décompose la contribution de chaque caractéristique à la prédiction du modèle pour une instance spécifique. Cela permet ainsi de comprendre pourquoi le modèle fait une certaine prédiction pour cette instance.

Par exemple, dans le cas particulier ci-dessous (instance ou observation 0), le fait que l'incident implique un conducteur agé de 45 à 54 ans pourrait avoir augmenté la probabilité de décès.



L'observation est prédite dans la classe: 1.0
Vraie valeur pour l'observation : 1.0

3.9 Optimisation des Prédictions

Dans cette section notre objectif principal est d'améliorer les résultats de nos prédictions obtenus dans les chapitres précédents. Pour ce faire, nous prendrons l'exemple du modèle Random Forest et les résultats de prédiction générés par ce modèle.

Voici le rapport de la classification pour le cas binaire du model Random Forest (combinaison hyperparamètres 144) :

Rapport de classification - Données de test :				
	precision	recall	f1-score	support
0.0	0.94	0.74	0.83	69089
1.0	0.41	0.80	0.54	15539
accuracy			0.75	84628
macro avg	0.67	0.77	0.68	84628
weighted avg	0.84	0.75	0.78	84628

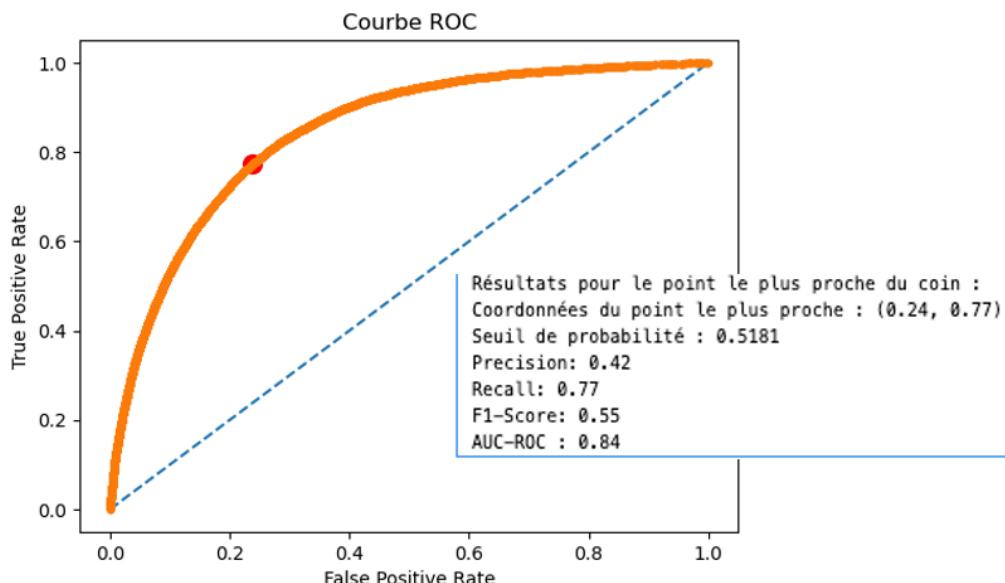
Dans la première partie nous utiliserons la méthode de la courbe ROC/AUC pour trouver le seuil de probabilité optimal. Cette approche nous permettra d'ajuster notre modèle afin d'obtenir un équilibre optimal entre le taux de vrais positifs et le taux de faux positifs. En identifiant le seuil de probabilité optimal, nous pourrons améliorer la performance de notre modèle en fonction de nos objectifs spécifiques.

Ensuite, dans le deuxième paragraphe, nous exploiterons les courbes de F-score pour améliorer notre rappel (recall) tout en maintenant une précision optimale. L'utilisation de ces courbes nous permettra d'ajuster notre modèle de manière à atteindre un équilibre optimal entre la capacité à détecter les véritables positifs et la minimisation des faux positifs.

3.9.1 Courbe ROC/AUC pour le seuil de Probabilité Optimal

Dans ce paragraphe, nous avons utilisé la courbe ROC/AUC pour optimiser nos prédictions. La courbe ROC/AUC est une méthode puissante pour évaluer la performance d'un modèle de classification, en particulier lorsqu'il y a un déséquilibre entre les classes. Nous utiliserons cette courbe pour déterminer le seuil de probabilité optimal, ce qui nous permettra d'ajuster notre modèle de manière à atteindre un équilibre optimal entre le rappel et la précision.

Nous avons d'abord calculé les scores ROC/AUC pour nos prédictions. Nous avons également tracé la courbe ROC, qui nous a montré la performance du modèle à différents seuils de probabilité (en dessous) :



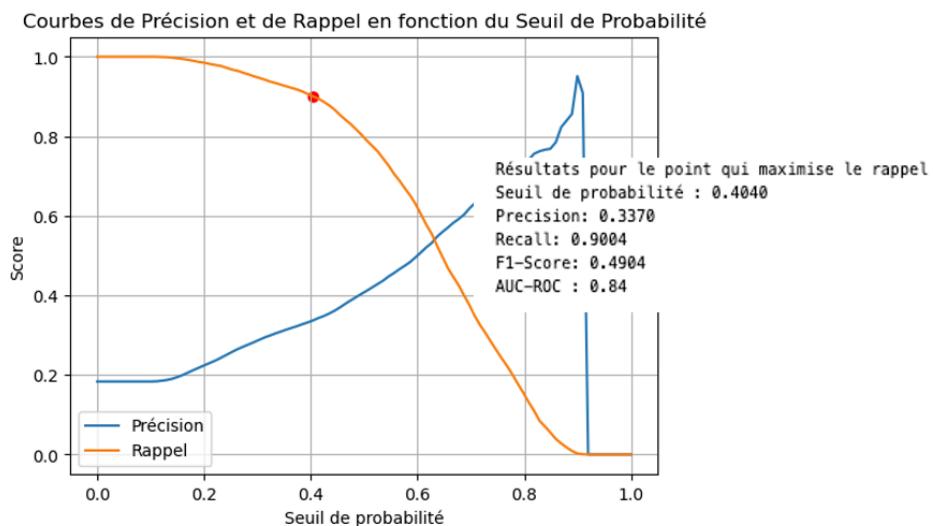
Cela nous a permis d'identifier le point le plus proche du coin supérieur gauche de la courbe ROC, qui représente un équilibre optimal entre la sensibilité et la spécificité.

Le seuil de probabilité optimal a été déterminé à environ 0.5181, ce qui a entraîné une précision de 0.42, un rappel de 0.77 et un F1-Score de 0.55. De plus, AUC-ROC a été estimée à 0.84, ce qui confirme la capacité du modèle à bien discriminer entre les classes.

Voici le rapport de la classification pour ce seuil de probabilité :

Rapport de classification:					
	precision	recall	f1-score	support	
0.0	0.94	0.76	0.84	69089	
1.0	0.42	0.77	0.55	15539	
accuracy			0.76	84628	
macro avg	0.68	0.77	0.69	84628	
weighted avg	0.84	0.76	0.79	84628	

Pour optimiser notre modèle en fonction de nos besoins spécifiques, nous avons cherché à maximiser le rappel tout en maintenant une précision acceptable. Notre objectif était d'identifier un seuil de probabilité optimal pour atteindre cet équilibre. La courbe ci-dessous montre comment la précision et le rappel évoluent en fonction du seuil de probabilité.



Pour atteindre cet équilibre, nous avons exploré différents seuils de probabilité pour décider comment classer les instances. Finalement, nous avons fixé le seuil de probabilité optimal à environ 0.4040, ce qui a entraîné un rappel de 0.9004, c'est-à-dire que notre modèle est capable de capturer 90 % des vrais cas positifs. Cependant, cette stratégie a entraîné une précision légèrement plus basse, mesurée à 0.3370, car il y avait également faux positifs.

Voici le rapport de la classification pour ce seuil de probabilité (0.404):

Matrice de confusion:					
[[41565 27524]					
[1548 13991]]					
Rapport de classification:					
	precision	recall	f1-score	support	
0.0	0.96	0.60	0.74	69089	
1.0	0.34	0.90	0.49	15539	
accuracy			0.66	84628	
macro avg	0.65	0.75	0.62	84628	
weighted avg	0.85	0.66	0.69	84628	

3.9.2 Courbes de F-score pour l'équilibre Recall-Precision

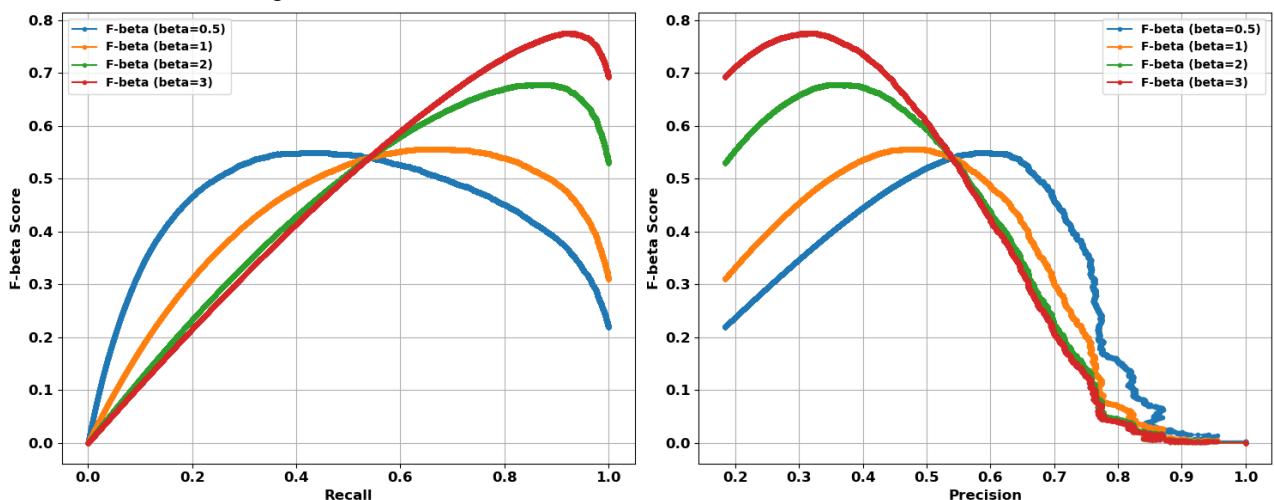
Dans ce paragraphe, nous explorerons l'utilisation des courbes de F-score pour optimiser notre modèle tout en maintenant un équilibre entre le rappel et la précision. Le F-score, contrairement à d'autres métriques qui se concentrent uniquement sur la précision ou le rappel, le F-score prend en compte à la fois les faux positifs et les faux négatifs, ce qui en fait un choix idéal lorsque les deux aspects sont importants.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Pour notre analyse, nous utilisons différentes valeurs de bêta (β) de F-score, telles que $\beta = 0.5$, $\beta = 1$, $\beta = 2$ et $\beta = 3$, qui déterminent le poids relatif que nous accordons à la précision par rapport au rappel. Chaque valeur de bêta représente un point de vue différent sur l'importance relative de la précision et du rappel dans notre modèle.

- $\beta < 1$: Favorise la précision sur le rappel, idéal pour réduire les faux positifs même au détriment de manquer certains vrais positifs.
- $\beta = 1$: Équilibre la précision et le rappel, souvent utilisé comme métrique par défaut pour un compromis équilibré.
- $\beta > 1$: Met l'accent sur le rappel, adapté lorsque la détection des vrais positifs est cruciale, même au prix d'un nombre plus élevé de faux positifs.

Les courbes F-score sont présentées en dessous.



Les graphiques sont divisés en deux parties. Dans le premier graphique, nous explorons les courbes F-beta score en fonction du rappel pour différentes valeurs de bêta (β). Chaque courbe F-beta représente un compromis entre la précision et le rappel, et nous pouvons observer comment elles évoluent en fonction de la sensibilité du modèle à la détection des vrais positifs.

Dans le deuxième graphique, nous examinons les courbes F-beta en fonction de la précision pour les mêmes valeurs de bêta. Cela nous permet de visualiser comment le F-score équilibre la précision et le rappel pour différents seuils de probabilité.

En combinant ces graphiques, nous pouvons choisir un seuil de probabilité qui correspond à nos besoins spécifiques en termes d'équilibre entre précision et rappel. Cela nous permet d'optimiser notre modèle pour une performance maximale tout en contrôlant le taux de faux positifs.

La démarche pour optimiser la prédiction en fonction du F-score β consiste à comparer ces résultats pour chaque valeur de β . En choisissant la valeur de β qui correspond le mieux à nos besoins spécifiques, nous pouvons contrôler l'équilibre entre précision et rappel pour notre modèle de classification.

Voici un récapitulatif de ces résultats :

Beta = 0.5, Précision = 0.6000, F0.5-score = 0.5495, Rappel = 0.4111
Seuil de probabilité optimal : 0.6830

Beta = 1, Précision = 0.4781, F1-score = 0.5562, Rappel = 0.6648
Seuil de probabilité optimal : 0.5789

Beta = 2, Précision = 0.3552, F2-score = 0.6778, Rappel = 0.8770
Seuil de probabilité optimal : 0.4355

Beta = 3, Précision = 0.3200, F3-score = 0.7751, Rappel = 0.9205
Seuil de probabilité optimal : 0.3680

Beta = 4, Précision = 0.2721, F4-score = 0.8353, Rappel = 0.9595
Seuil de probabilité optimal : 0.2752

Dans notre cas, nous avons choisi $\beta = 4$, car il favorise un rappel élevé tout en maintenant une précision légèrement plus basse. Cela signifie que notre modèle est capable de capturer 96 % des vrais cas positifs, même si la précision est réduite en raison du rappel élevé.

En utilisant le seuil de probabilité associé à $\beta = 4$ (0.2752), nous pouvons réaliser des prédictions sur l'ensemble de test.

Voici le rapport de la classification :

Rapport de classification:				
	precision	recall	f1-score	support
0.0	0.98	0.42	0.59	69089
1.0	0.27	0.96	0.42	15539
accuracy			0.52	84628
macro avg	0.63	0.69	0.51	84628
weighted avg	0.85	0.52	0.56	84628

La matrice de confusion :

Matrice de confusion:
[[29205 39884]
 [630 14909]]

Parmi les 84 628 cas de l'ensemble de données, notre principal objectif est la précision de la détection des cas graves de la classe 1.0, qui compte 15.539 cas. Bien que le modèle atteigne un rappel élevé de 96 % pour ces cas, il manque encore 630 cas graves.

Le rappel élevé de 96 % pour les cas graves s'accompagne de 39.884 faux positifs. Dans notre contexte, il est préférable d'accepter ces faux positifs que de manquer des cas graves, étant donné les graves conséquences de leur non-détection.

Dans le tableau ci-dessous, les résultats de prédiction pour différents seuils de décision sont présentés :

Threshold	Negative precision	Positive precision	Negative recall	Positive recall	Negative f1-score	Positive f1-score	F1-macro
0.68	0.88	0.60	0.94	0.41	0.91	0.49	0.70
0.58	0.92	0.48	0.84	0.66	0.88	0.56	0.72
0.44	0.96	0.36	0.64	0.88	0.77	0.51	0.64
0.37	0.97	0.32	0.56	0.92	0.71	0.47	0.59
0.28	0.98	0.27	0.42	0.96	0.59	0.42	0.51

4. Exemple d'Application des Résultats de la Prédition : Gestion des Risques Routiers

Afin de mieux comprendre la manière dont les résultats de la prédition peuvent être exploités, envisageons un scénario où le gouvernement est confronté à des problèmes de sécurité routière. Les accidents graves sur les routes suscitent de vives préoccupations en raison de leurs répercussions humaines et économiques considérables.

Pour faire face à cette problématique, le gouvernement a mis en place un modèle d'apprentissage automatique visant à prédire les accidents graves (dans un contexte binaire). Cependant, il est essentiel de noter que ce modèle ne peut garantir une précision de 100 % dans ses prédictions. Par conséquent, il est impératif d'optimiser la détection tout en minimisant les coûts associés à des prédictions incorrectes.

En raison de prédictions incorrectes, des coûts économiques peuvent se présenter sous forme :

Faux Positifs (Fausses Alarmes) :

Chaque faux positif, c'est-à-dire chaque fois que le modèle prédit un accident grave qui n'en est pas un, entraîne des coûts économiques.

Par exemple, cela peut entraîner des perturbations de la circulation, des ressources mobilisées inutilement et etc

Faux Négatifs (Accidents Graves Non DéTECTÉS) :

Chaque faux négatif, c'est-à-dire chaque fois que le modèle ne parvient pas à prédire un accident grave, peut entraîner des coûts humains et économiques importants.

- Pertes de vie humaines : Un accident grave non détecté peut entraîner la perte de vies humaines, ce qui est inestimable du point de vue humain.
- Coûts Médicaux : Les blessures graves nécessitent des soins médicaux coûteux et peuvent engendrer des dépenses importantes pour le système de santé.

Afin de calculer les coûts économiques associés aux différents scénarios d'optimisation de la prédition, nous utilisons les coûts économiques des FP et FN suivent

- le coût moyen par faux positif peut être estimé à 1.000 euros (mille)
- le coût moyen par faux négatif peut être estimé à 5.000 euros (cinq mille)

(les chiffres des coûts économiques ont été sélectionnés de manière aléatoire)

Nous allons examiner trois scénarios d'optimisation possibles : Scénario 1 - Optimisation de la Précision, Scénario 2 - Modèle Standard, Scénario 3 - Optimisation du Rappel.

Voici les résultats des prédictions pour chacun de ces scénarios, basés sur un ensemble de données de test comprenant un total de 84628 cas :

Scenario	Threshold	Negative precision	Positive precision	Negative recall	Positive recall	FN	FP	VN	VP
1	0.68	0.88	0.60	0.94	0.41	9153	4257	64832	6386
2	0.44	0.96	0.36	0.64	0.88	1911	24744	44345	13628
3	0.28	0.98	0.27	0.42	0.96	630	39884	29205	14909

Scénario 1 : Optimisation de la Précision

Faux Positifs (FP) : 4.257

Faux Négatifs (FN) : 9.153

Calcul des coûts économiques :

Coût Faux Positifs (CFP) = FP * Coût moyen par Faux Positif = 4.257 * 1.000 euros = 4.257.000 euros

Coût Faux Négatifs (CFN) = FN * Coût moyen par Faux Négatif = 9.153 * 5.000 euros = 45.765.000 euros

Coût économique total (CET) = CFP + CFN = 50.022.000 euros

Scénario 2 : Model standard

Faux Positifs (FP) : 24.744

Faux Négatifs (FN) : 1.911

Calcul des coûts économiques :

Coût Faux Positifs (CFP) = FP * Coût moyen par Faux Positif = 24.744 * 1.000 euros = 24.744.000 euros

Coût Faux Négatifs (CFN) = FN * Coût moyen par Faux Négatif = 1.911 * 5.000 euros = 9.555.000 euros

Coût économique total (CET) = CFP + CFN = 34.299.000 euros

Scénario 3 : Optimisation du Rappel

Faux Positifs (FP) : 39.884

Faux Négatifs (FN) : 630

Calcul des coûts économiques :

Coût Faux Positifs (CFP) = FP * Coût moyen par Faux Positif = 39.884 * 1.000 euros = 39.884.000 euros

Coût Faux Négatifs (CFN) = FN * Coût moyen par Faux Négatif = 630* 5.000 euros = 3.150.000 euros

Coût économique total (CET) = CFP + CFN = 43.034000 euros

Scénario 1 (Optimisation de la Précision) : Coût économique total = 50.022.000 euros

Scénario 2 (Modèle Standard) : Coût économique total = 34.299.000 euros

Scénario 3 (Optimisation du Rappel) : Coût économique total = 43.034.000 euros

Dans cette analyse, nous avons évalué et calculé le coût associé à différentes stratégies visant à optimiser le modèle de prédiction des accidents graves. Le scénario 2 a montré le coût économique total le plus bas. Cependant, il est important de noter que le scénario 3, axé sur l'optimisation du rappel, bien qu'en entraînant un coût légèrement plus élevé, permet de détecter davantage d'accidents graves.

Il est important de noter que les chiffres des coûts économiques ont été sélectionnés de manière arbitraire. La réalité des coûts peut varier considérablement. Par conséquent, cette analyse visait à illustrer les stratégies plutôt qu'à fournir une réponse précise. Il est essentiel de réaliser une évaluation approfondie des coûts réels pour prendre des décisions finales.

5. Synthèse de résultats de la modélisation

Cas multiclass

Algorithm	Précision classe_2	Recall classe_2	f1-score classe_2	f1-score classe_3	f1-score macro	Sur-apprentissage
Random Forest	0.18	0.45	0.26	0.44	0.51	non
Arbre de décision	0.15	0.12	0.13	0.37	0.44	oui
Logistique régression	0.121	0.648	0.205	0.358	0.466	oui
KNN	0.27	0.03	0.05	0.39	0.62	oui

Les performances de chaque algorithme sont évaluées en termes de précision classe_2, de rappel classe_2, de F1-score classe_2, de F1-score classe_3 et du score F1-macro global. Nous examinons également la possibilité de surapprentissage pour chaque modèle. Les meilleurs algorithmes sont les suivants : Random Forest

Cas binaire

Algorithm	Positive précision	Positive recall	Negative f1-score	Positive f1-score	F1-macro	Sur-apprentissage
Random Forest	0.41	0.80	0.83	0.54	0.68	non
Arbre de décision	0.38	0.77	0.81	0.51	0.66	oui
Logistique régression	0.41	0.798	0.831	0.544	0.688	non
KNN	0.71	0.73	0.76	0.72	0.74	oui
XGBoost	0.65	0.42	0.91	0.51	0.71	oui

Les performances de chaque algorithme sont évaluées en termes de précision positive, de rappel positif, de F1-score négatif, de F1-score positif et du score F1-macro global. Nous examinons également la possibilité de surapprentissage pour chaque modèle. Les meilleurs algorithmes sont les suivants : Random Forest et Logistique régression.

Optimisation

En utilisant les résultats de la prédiction du modèle Random Forest pour le cas binaire (avec la combinaison d'hyperparamètres 144), nous avons effectué une optimisation pour ajuster à la fois le rappel et la précision :

Threshold	Negative precision	Positive precision	Negative recall	Positive recall	Negative f1-score	Positive f1-score	F1-macro
0.68	0.88	0.60	0.94	0.41	0.91	0.49	0.70
0.58	0.92	0.48	0.84	0.66	0.88	0.56	0.72
0.44	0.96	0.36	0.64	0.88	0.77	0.51	0.64
0.37	0.97	0.32	0.56	0.92	0.71	0.47	0.59
0.28	0.98	0.27	0.42	0.96	0.59	0.42	0.51

L'ajustement du seuil de décision peut permettre de modifier le compromis entre la précision et le rappel, en fonction de l'importance relative de ces deux métriques pour votre application spécifique. Plus le seuil est élevé, plus la précision positive est élevée, mais le rappel positif diminue, et vice versa. Le choix du seuil dépendra de vos objectifs spécifiques en matière de prévision et de la manière dont vous souhaitez gérer les faux positifs et les faux négatifs.

6. Conclusion et Perspectives

L'objectif principal de notre projet a été de développer un modèle prédictif capable d'estimer la gravité des accidents routiers en France en se basant sur les données historiques. Nous avons élaboré des modèles de classification pour les cas multiclass et binaire, en utilisant des méthodes de rééchantillonnage pour gérer efficacement le déséquilibre des classes. Différents algorithmes de machine learning ont été testés, et nous avons mis en œuvre des techniques d'optimisation telles que RandomizedSearchCV et GridSearchCV en combinaison avec la validation croisée. Notre objectif principal était d'optimiser le modèle pour maximiser le rappel des cas graves tout en maintenant une précision acceptable.

En ce qui concerne les processus métier, notre modèle pourrait être mis à profit par les autorités de sécurité routière pour évaluer la gravité potentielle des accidents. Il pourrait également être utile pour les autorités de santé afin d'estimer les budgets nécessaires pour la prise en charge des victimes d'accidents. Enfin, il pourrait être intégré dans des systèmes de navigation pour avertir les conducteurs des risques potentiels sur leur itinéraire, améliorant ainsi la sécurité routière.

Le choix final de la stratégie d'optimisation de la prédiction dépendra des besoins spécifiques de chaque application ainsi que des contraintes économiques associées.

Difficultés rencontrées lors du projet

Organisation et planification :

Le cursus s'effectue à distance, en parallèle de nos activités professionnelles respectives. Également, le projet est à mener en parallèle des cours à suivre examens et examens à passer. Cela implique beaucoup de choses à gérer en parallèle, d'où la difficulté de pouvoir organiser efficacement les réunions, et le travail autour de ce projet. Également, l'absence d'expérience dans ce type de projet rend l'estimation de charge difficile, et donc sa planification. Nous avons donc essayé un certain nombre d'écueils, qui sont très formateurs in fine.

Définition d'un objectif

Bien souvent un projet se réalise lors d'un besoin client qui désire réaliser un objectif. Dans ce projet, nous avions des données, mais l'objectif était à établir par nous-même. Nous avons consacré du temps à débattre de l'objectif recherché, en prenant en compte l'agrégation du dataset, la meilleure façon de le structurer, et en examinant les options telles que : l'organisation par usager, par véhicule ou par accident.

Maturité, qualité et complétude des données :

Nous avons identifié des problèmes tels que les erreurs de frappe, et la perte de lignes contenant des informations importantes après avoir géré les valeurs manquantes. Également, nous avons un jeu de données qui est restreint : en effet, le gouvernement n'a pas mis à disposition du public toutes les features disponibles dans leur base. Certaines, d'entre elles auraient certainement été très intéressantes et pertinentes pour améliorer notre compréhension des accidents et l'amélioration de nos prévisions. Nous avons consacré du temps à la recherche de moyens pour enrichir le dataset par l'ajout de nouvelles variables, sans y parvenir faute de bases disponibles.

Volumétrie et performances :

Certaines étapes du projet ont pris plus de temps que prévu, en particulier la phase d'optimisation des hyperparamètres. Nous avons sous-estimé le nombre de combinaisons possibles à tester, ce qui a rallongé la durée de cette étape. De plus, nous avons consacré du temps à la recherche de moyens pour enrichir le dataset en ajoutant de nouvelles variables. Parallèlement, faute d'expérience, nous n'avons pas pu estimer les temps de traitements qui nous ont surpris par le temps nécessaire de calcul, et le fait qu'ils exploitaient nos ressources machines de sorte que l'on ne pouvait plus travailler dessus jusqu'à attendre la fin des calculs.

Suite du projet

Amélioration de la qualité du dataset:

L'une étape consistera à enrichir notre dataset en ajoutant des variables qui peuvent avoir un impact significatif sur la gravité des accidents. Cela inclut des informations sur le véhicule ou le conducteur en fuite, les

détails de l'assurance, l'état du permis de conduire, la date d'obtention du permis, la présence de produits stupéfiants identifiés, le taux d'alcoolémie, les résultats des vérifications d'alcoolémie, et d'autres encore. Ces variables existent dans le dataset, mais elles n'étaient pas disponibles pour le public. L'ajout de ces données contribuera à améliorer la qualité de nos modèles de prédiction.

Intégration de données météorologiques :

Nous envisageons également d'incorporer davantage de données météorologiques en utilisant des techniques de webscraping. Les conditions météorologiques peuvent jouer un rôle crucial dans la gravité des accidents, et en incluant ces données, nous pourrons affiner nos prédictions.

Exploration de différentes méthodes d'agrégation :

Une autre étape importante consistera à explorer différentes méthodes d'agrégation des données. Nous pourrons tester l'agrégation par usager, par véhicule et par accident afin de déterminer celle qui convient le mieux à notre problème de prédiction en fonction des besoins spécifiques.

Plateforme dédiée :

La mise à disposition d'un serveur, d'une plateforme dédiée, au traitement des algorithmes permettrait à chaque personne du projet d'accéder à un environnement performant. Il serait source de gain de temps, il permettrait de ne pas « bloquer » les PC respectifs de chacun, et serait une meilleure base de comparaison des performances algorithmiques.

Ces étapes visent à renforcer la qualité et la pertinence de notre modèle de prédiction de la gravité des accidents routiers, nous permettant ainsi d'obtenir des résultats plus précis et utiles pour les parties prenantes concernées.