

Projet A3 - IA

Antonin SOQUET, Maxence LAURENT, Martin LOBEL

23 juin 2023

ISEN
école
d'ingénieurs



Table des matières

1	Introduction	3
2	Organisation	4
2.1	Outils	4
2.2	Répartition du travail	4
3	Découverte et préparation des données	5
4	Apprentissage non-supervisé	6
5	Apprentissage supervisé	7
5.1	Répartition des données	7
5.2	Classification avec KNN	10
5.2.1	Notre implémentation avec KNN	10
5.3	Classification avec trois algorithmes de « haut niveau »	12
5.3.1	Analyse des resultats	12
6	Création de scripts	14
6.1	Script 1 : Script pour l'apprentissage non-supervisé	14
6.2	Script 2 : Script pour l'apprentissage supervisé	14
6.3	Script 3 : Script pour les méthodes de classification de « haut niveau »	14

1 Introduction

Ce projet est découpé en 3 matières : Big Data / IA / Web et à pour principal objectif de concevoir et développer une application d'étude des accidents de la route.

Dans le cadre de notre étude à l'ISEN Nantes il nous permettra d'approfondir les compétences acquises dans les modules Big Data, Intelligence Artificielle, Développement Web et Base de Données à travers une application complète de traitements et de visualisation de données concernant les accidents corporels de la circulation routière en France.

La partie Intelligence Artificielle du projet de fin d'année se concentre sur plusieurs points. Premièrement, elle vise à fournir une compréhension approfondie des étapes impliquées dans un projet d'apprentissage automatique, allant de la collecte des données à l'évaluation des modèles. Deuxièmement, elle vise à développer la capacité d'évaluer différentes méthodes d'apprentissage automatique afin de sélectionner la plus appropriée pour un problème donné. Cette partie nous permettra de maîtriser diverses techniques d'apprentissage automatique, notamment l'apprentissage non supervisé qui comprend la réduction de la dimensionnalité et le clustering, mais également l'apprentissage supervisé qui englobe la régression et la classification.

2 Organisation

Nous estimons notre organisation durant ce projet de Big Data plutôt bonne, puisque nous n'avons pas eu de soucis particuliers que ce soit dans la communication au sein du groupe ou dans la répartition des tâches par exemple.

Comme évoqué précédemment, nous avons réussi à bien répartir les tâches entre les membres de notre équipe, ce qui nous a permis d'avancer efficacement dans notre travail. De plus, nous avons mis un point d'honneur à communiquer clairement sur les besoins spécifiques de chacun.

2.1 Outils

Pour mener ce projet à bien nous avons utilisé plusieurs outils. Comme environnement de développement nous avons utilisé VS Code sur lequel nous utilisons Jupyter Notebook. Pour un travail collaboratif et partagé nous avons utilisé GitHub qui est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git.

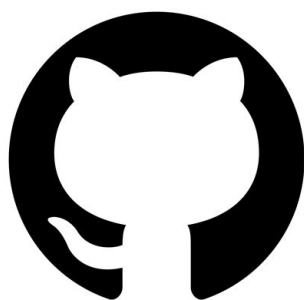


FIGURE 1 – GitHub



FIGURE 2 – Jupyter Notebook

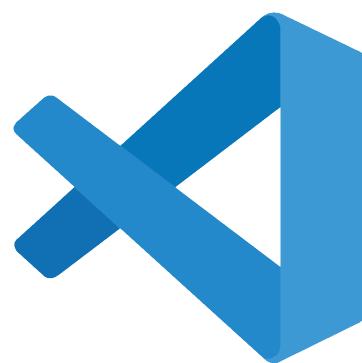


FIGURE 3 – Visual Studio Code

2.2 Répartition du travail

Nous avons décidé de créer un diagramme de Gantt sur GitHub pour des raisons organisationnelles. En utilisant GitHub, nous avons pu créer un référentiel centralisé pour notre projet, ce qui facilite la collaboration et la gestion des tâches. Chaque tâche peut être commentée et ainsi associée à un ou plusieurs commit, ce qui permet de suivre facilement les modifications et confère une traçabilité. En utilisant cette approche, nous pouvons efficacement coordonner nos travaux et travailler de manière organisée et construite.

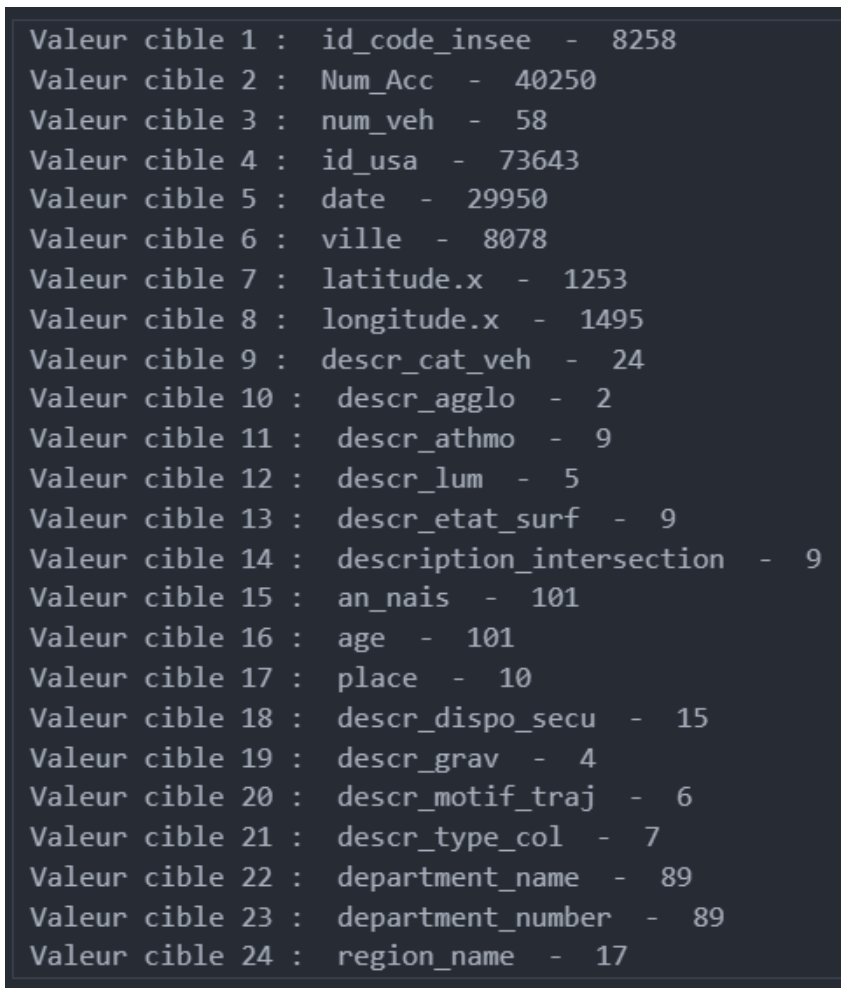
Si vous souhaitez consulter le diagramme de Gantt il est visualisable via [ce lien](#)

On peut également voir la répartition des tâches attribuées au début du projet ainsi :

Antonin	Maxence	Martin
Répartition des tâches	Répartition des tâches	Répartition des tâches
Apprentissage supervisé	Découverte et préparation des données	Découverte et préparation des données
Rédaction du rapport	Apprentissage non-supervisé	Apprentissage supervisé
Scripts py	Setup du Git	Rédaction du rapport
Préparation diapositive	Scripts py	Setup du Git (Gantt)
	Préparation diapositive	Scripts py

3 Découverte et préparation des données

Lorsque l'on a chargé la base de données, nous avons déterminé qu'il y avait 73643 accidents recensés, voici un récapitulatif du nombre de données différentes pour chaque feature.



```
Valeur cible 1 : id_code_insee - 8258
Valeur cible 2 : Num_Acc - 40250
Valeur cible 3 : num_veh - 58
Valeur cible 4 : id_usa - 73643
Valeur cible 5 : date - 29950
Valeur cible 6 : ville - 8078
Valeur cible 7 : latitude.x - 1253
Valeur cible 8 : longitude.x - 1495
Valeur cible 9 : descr_cat_veh - 24
Valeur cible 10 : descr_agglo - 2
Valeur cible 11 : descr_athmo - 9
Valeur cible 12 : descr_lum - 5
Valeur cible 13 : descr_etat_surf - 9
Valeur cible 14 : description_intersection - 9
Valeur cible 15 : an_nais - 101
Valeur cible 16 : age - 101
Valeur cible 17 : place - 10
Valeur cible 18 : descr_dispo_secu - 15
Valeur cible 19 : descr_grav - 4
Valeur cible 20 : descr_motif_traj - 6
Valeur cible 21 : descr_type_col - 7
Valeur cible 22 : department_name - 89
Valeur cible 23 : department_number - 89
Valeur cible 24 : region_name - 17
```

FIGURE 4 – Détail de la composition de la base de données

À la lecture des premiers retours d'analyse, on comprend que nous allons devoir retransformer certaines valeurs telles que la date et qu'il faudra retranscrire les données en valeur numérique, car elles n'auront pas encore eu de traitement.

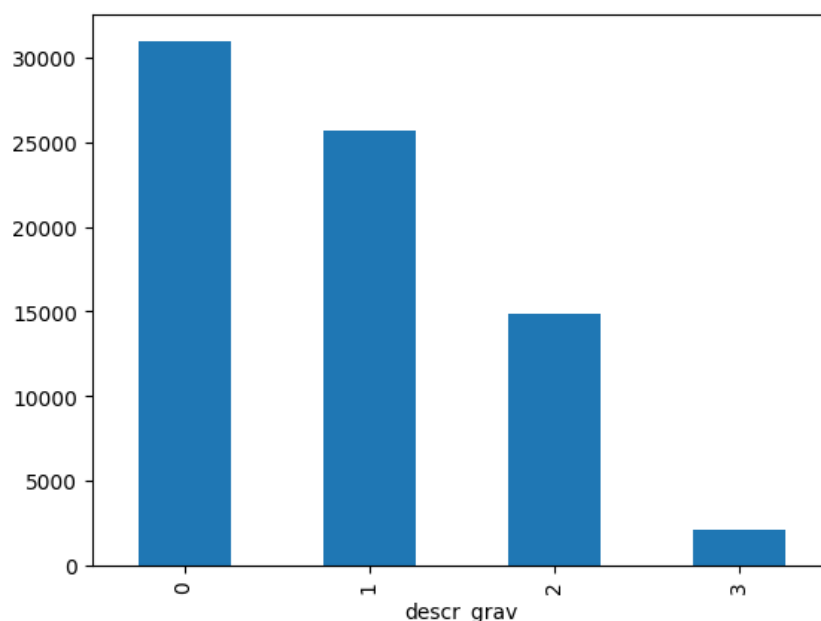


FIGURE 5 – Répartition des gravités dans la base de données : 0 = indemne 1 = blessé léger 2 = blessé hospitalisé 3 = tué

A la fin de notre traitement, les données sont toutes des valeurs numériques ce qui sera très important pour nos fonctions plus tard. Nous avons transformés la date pour qu'il n'y est plus d'espaces dans l'affichage de celui-ci et nous avons par sécurité bien précisé que les valeurs étaient des entiers ou des flottants.

4 Apprentissage non-supervisé

Avant de pouvoir réaliser le clustering avec Kmeans, il était primordial de réduire les données pour obtenir des résultats aussi proches que possible de la réalité. Pour réaliser cela, nous avons utilisé deux méthodes que nous comparerons à la fin de cette partie. Dans un premier temps, nous avons réalisé une réduction de dimension à la main avec une matrice de dimension de corrélation qui nous a permis de retirer les dimensions qui étaient suffisamment corrélées entre elles. Comme nous pouvons le voir sur la matrice ci-dessus, on peut déterminer que les colonnes : 'an_nais', 'id_usa', 'Num_Acc' et 'department_number' peuvent être supprimées, car elles sont corrélées à d'autres variables et cela serait de la redondance. Dans un second temps, nous avons réalisé la méthode PCA sur les données afin de réduire, notre jeu de données de dimension 21 à une dimension de 7, cependant, il faut garder à l'esprit que cette réduction transforme nos données qui ne sont donc plus compréhensibles facilement que le sont les données avec le traitement manuel. La méthode de Kmeans est de choisir aléatoirement plusieurs centroides, les différents centres des clusters et de calculer la distance de chaque point à ce centre et d'améliorer la distance de chaque point à son cluster le plus proche. Comme on peut le voir, le découpage n'est pas identique, la méthode de Kmeans utilisant de l'aléatoire pour choisir les centroides, cela explique les différences de découpage. Pour finir, nous allons comparer les différents de résultats donnés pour la réduction manuelle et pca ainsi que la méthode de Kmeans que nous avons codé à la main et celle de scikit learn. Lorsque nous lançons le calcul des différents scores, celui de Silhouette, de Calinski-Harabasz et de Davies-Bouldin qui vont nous permettre de déterminer quelle méthode aura été la plus efficace et pertinente.

Score de silhouette sur les données manuelles avec Kmeans recodé : -0.04255662179491376

Score de silhouette sur les données manuelles avec Kmeans scikit : 0.56718671213116044

Score de silhouette sur les données PCA avec Kmeans scikit : 0.8674884745935049

Score de Calinski-Harabasz sur les données manuelles avec Kmeans recodé : 6.264823420237192

Score de Calinski-Harabasz sur les données manuelles avec Kmeans scikit : 47433.09353543223

Score de Calinski-Harabasz sur les données PCA avec Kmeans scikit : 9363097.980487494

Score de Davies-Bouldin sur les données manuelles avec Kmeans recodé : 383.76158566961936

Score de Davies-Bouldin sur les données manuelles avec Kmeans scikit : 0.7204794124231818

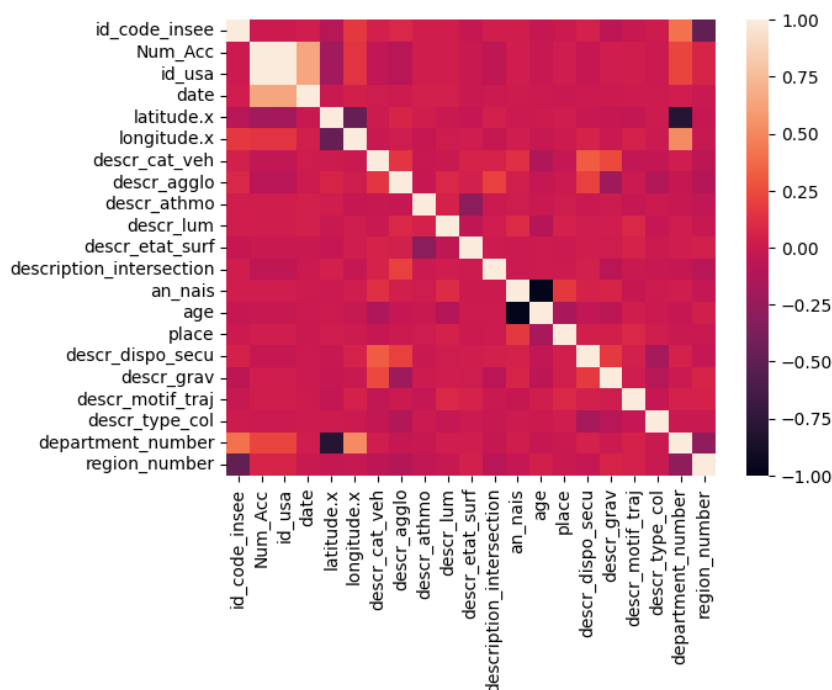


FIGURE 6 – Matrice de corrélation des dimension entre elles

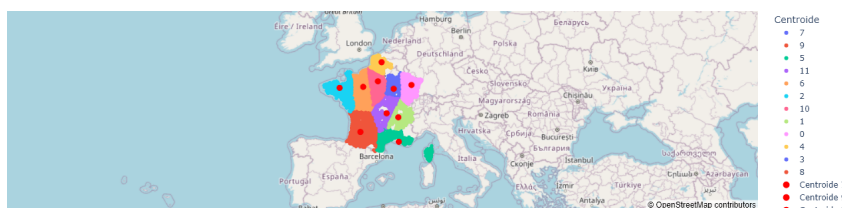


FIGURE 7 – Découpage de la France Métropolitaine en clusters avec la méthode manuel

Score de Davies-Bouldin sur les données PCA avec Kmeans scikit : 0.2071205154857655

Les résultats montrent que les données issues de PCA avec la méthode de scikit sont largement meilleurs que les données manuelles (que nous avons traité manuellement) avec notre méthode de Kmeans recodé. Mais on observe que les données manuelles avec la méthode de Kmeans de scikit restent assez compétitives, car leurs scores sont suffisamment intéressants et élevés. Cela, nous laisse à penser que les choix que nous avons réalisés un bon choix lorsque nous avons réduit les dimensions de notre jeu de données.

5 Apprentissage supervisé

5.1 Répartition des données

Afin de pouvoir utiliser les données et développer des méthodes de classification, il est important de bien répartir ses données préalablement. Pour chaque set, il nous faut une base d'entraînement et une base de test. Pour cela, nous réalisons une répartition à partir de plusieurs méthodes : La première méthode est la plus facile :

La méthode Holdout consiste à diviser le jeu de données en deux parties distinctes : un ensemble d'apprentissage (training set) et un ensemble de test (test set). Généralement, une proportion des données est allouée à l'apprentissage (dans notre cas 80 %) et le reste est utilisé pour le test (20 %). L'ensemble d'apprentissage est utilisé pour entraîner le modèle, tandis que l'ensemble de test est utilisé pour évaluer les performances du modèle sur des données non vues auparavant.

Cette méthode présente des limitations en termes d'optimisation. En effet, la répartition des données entre l'ensemble d'apprentissage et l'ensemble de test peut entraîner une variabilité dans les performances du modèle. Nous effectuons donc un test sur 5 échantillons afin d'obtenir une vue d'ensemble de la précision en utilisant cette méthode. Ensuite, nous calculons la moyenne des performances des algorithmes pour

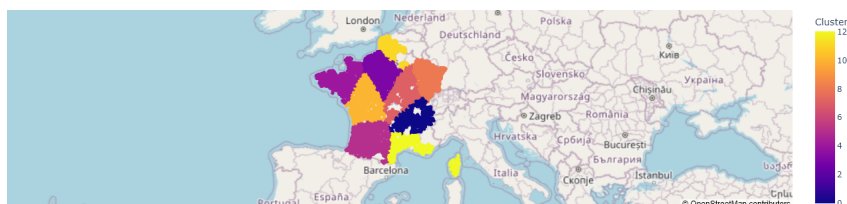


FIGURE 8 – Découpage de la France Métropolitaine en clusters avec la méthode de scikit learn

obtenir une valeur fiable permettant de comparer ces derniers entre eux. Cela nous permet d'obtenir une mesure de performance robuste et de confiance pour évaluer les algorithmes

```

Jeu de donné 1
Accuracy SVM: 0.4201235657546337
Accuracy RF: 0.6330368660465748
Accuracy MLP: 0.35338447959807184

Jeu de donné 2
Accuracy SVM: 0.4267092131169801
Accuracy RF: 0.6311358544368253
Accuracy MLP: 0.028786747233349175

Jeu de donné 3
Accuracy SVM: 0.42209247063615996
Accuracy RF: 0.6377893950709485
Accuracy MLP: 0.42209247063615996

Jeu de donné 4
Accuracy SVM: 0.42270351008215085
Accuracy RF: 0.6322900400570304
Accuracy MLP: 0.42270351008215085

Jeu de donné 5
Accuracy SVM: 0.41693258198112565
Accuracy RF: 0.6324937198723607
Accuracy MLP: 0.41693258198112565

Moyenne SVM: 0.4217122683142101
Moyenne RF: 0.6333491750967479
Moyenne MLP: 0.3287799579061715

```

Pour cette série de tests sur 5 échantillons, nous avons obtenu une précision moyenne de 0,42 pour SVM, 0,63 pour RandomForest et presque 0,33 pour MLP. En se basant sur ces résultats, nous pouvons conclure que le modèle RandomForest est plus performant que les deux autres modèles testés.

Une autre méthode utilisée est Leave One Out : Leave-One-Out est une approche de validation croisée où chaque échantillon individuel est utilisé comme ensemble de test, tandis que tous les autres échantillons sont utilisés comme ensemble d'apprentissage. On crée autant d'ensembles de test que le nombre total d'échantillons dans le jeu de données. Chaque modèle est entraîné sur l'ensemble des échantillons, à l'exception d'un seul, qui est réservé pour l'évaluation.

Repartition de la base de données from scratch. Pour répartir mes données sans utiliser la bibliothèque scikit-learn, nous avons dû créer un algorithme afin de les répartir correctement dans les proportions souhaitées. Notre algorithme permet donc de répartir les données avec 80% des valeurs appartenant à la base d'entraînement et 20% à la base de test. La première étape de notre algorithme mélange la base de données et calcule combien d'éléments correspondent à 80% (respectivement 20%) de la base de données. Ensuite, nous affectons 80% des valeurs à la base d'entraînement et le reste à la base de test. Il nous suffit ensuite de séparer les champs X et y de chaque base afin de pouvoir réaliser nos modèles et obtenir une précision sur nos prédictions.

Les résultats obtenus sont très similaires aux résultats obtenus en utilisant train_test_split de scikit-learn. Nous avons :

Jeu de données from scratch :
Accuracy SVM : 0.42317876298458823

Accuracy RF : 0.6364315296354132

Accuracy MLP : 0.3491750967479123

Encore une fois, nous pouvons constater que le modèle de Random Forest est supérieur aux modèles de SVM ou de MLP.

Pour réaliser les tests suivants, nous avons retiré certains paramètres de la base de données que nous avons jugé inutiles pour réaliser les prédictions, comme le numéro d'accident par exemple. La base utilisée contient alors les valeurs de : `descr_cat_veh`, `descr_agglo`, `descr_atmo`, `descr_lum`, `descr_etat_surf`, `age`, `descr_dispo_secu`, `descr_grav`, `descr_type_col`

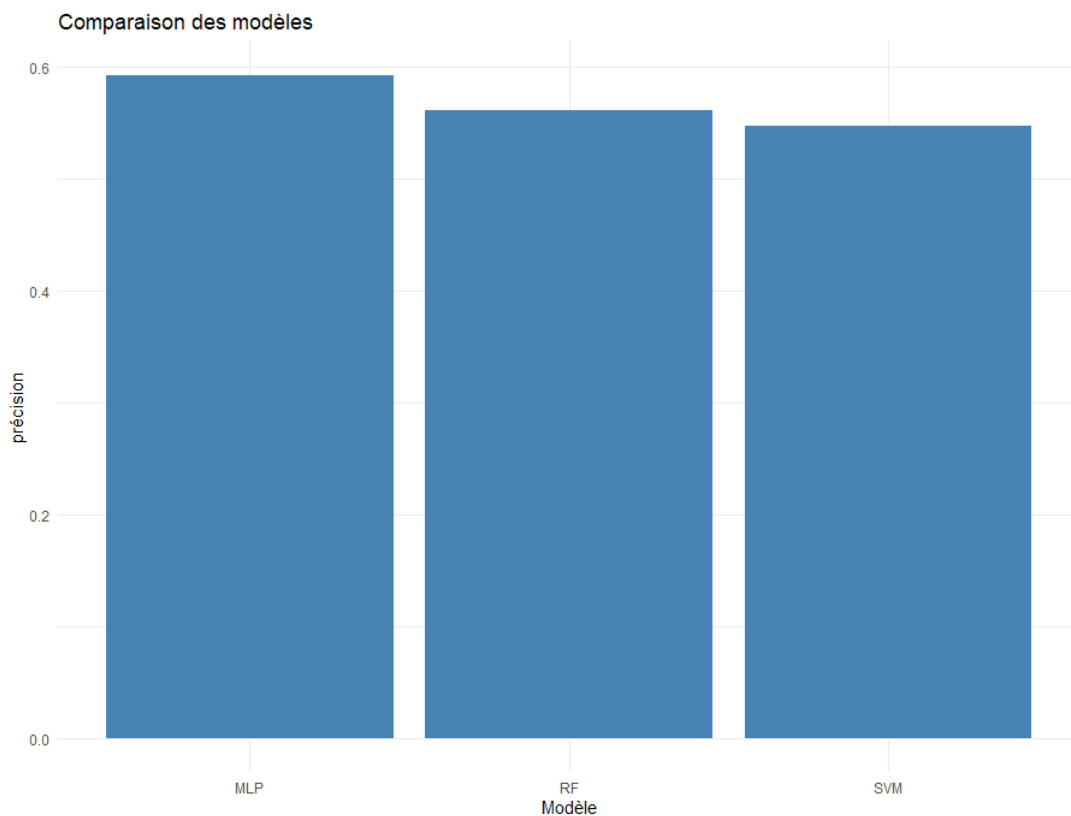
Nous avons constaté une amélioration majeure par certaines méthodes :

En utilisant `train_test_split` :

Moyenne SVM : 0.547939439201575

Moyenne RF : 0.5610428406544912

Moyenne MLP : 0.5936316111073393



Nous constatons donc que les modèles sont maintenant relativement similaires, MLP est même devenu plus précis que Random Forest. Le modèle SVM s'est également bien amélioré.

Pour le jeu de données from Scratch nous obtenons des précisions de :

Accuracy SVM : 0.5477629166949556

Accuracy RF : 0.5650078077262544

Accuracy MLP : 0.5900604250118813

Nous constatons bien l'amélioration de ce tri sur les valeurs des prédictions, pour la suite des résultats, nous garderons alors cette base de données triée.

5.2 Classification avec KNN

La méthode du k plus proches voisins (KNN, pour k-nearest neighbors en anglais) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. L'algorithme KNN est souvent utilisé dans des situations où les données sont relativement simples et bien structurées. Il peut être utilisé pour la classification de texte, la recommandation de produits, la reconnaissance de formes, la prédiction de valeurs continues (régression), etc.

L'objectif principal de l'algorithme KNN est de prédire la classe ou la valeur d'une nouvelle instance non étiquetée en se basant sur les exemples étiquetés présents dans l'ensemble de données d'entraînement. Il utilise la similarité entre les caractéristiques de l'instance à prédire et celles des exemples d'entraînement pour effectuer cette prédiction.

5.2.1 Notre implémentation avec KNN

Dans cette partie nous avons donc pu implémenter la méthode KNN de deux manières différentes.

- Premièrement, nous avons utilisé les fonctions mises à disposition dans la bibliothèque scikit-learn.
- Deuxièmement, nous avons créé la fonction "from scratch", c'est-à-dire à partir de zéro.

Nous allons donc comparer ces deux implémentations en terme de précision des résultats et de temps d'exécution. Au début nous avons lancé le programme sur la totalité de la base de données que nous avons mais au bout de presque 2h30 nous avons décidé de filtrer et de tester le code uniquement sur les 10 000 premières lignes. Nous avons également fait varier la valeur de K dans notre code from scratch afin de voir l'impacte sur les résultats obtenus. Nous avons choisi de faire varier K de 1 à 12.

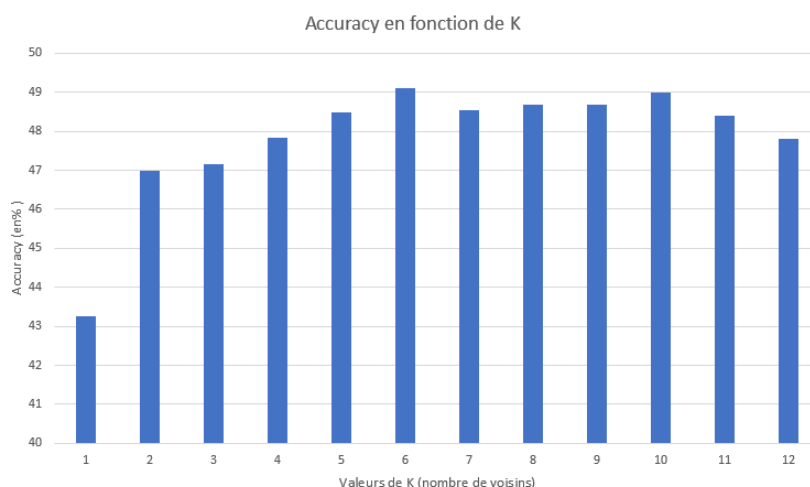


FIGURE 9 – Précision du modèle KNN en fonction du nombre de voisins - Version Scikit-learn

Il est important de noter aussi que chaque méthode du KNN doit faire appel à un calcul de distance (pour trouver les voisins les plus proches). Dans nos tests présentés au-dessus, nous avons utilisé la distance euclidienne.

Si on devait comparer nos deux modèles. D'un point de vue global on remarque que les prédictions sont très similaires à partir de $K=5$. On remarque aussi que l'Accuracy tourne autour des 48% à 49% de précision environ, que ce soit pour la version "from scratch" ou "scikit-learn" pour des valeurs de K similaires. La grosse différence revient surtout au niveau du temps d'exécution. D'un côté on a la version Scikit qui prend environ 1 seconde à l'exécution pour tester avec tous les K allant de 1 à la valeur entrée. Tandis que l'algorithme from scratch prend environ 2 min mais uniquement pour tester avec le K rentré en paramètre (ne teste pas avec les K inférieurs). La version from scratch est bien plus longue à exécuter que la version avec les bibliothèques de Scikit-learn.

```

Gravité prédite : [0 0 1 ... 1 2 0]
Accuracy pour la prédiction de gravité : 43.25 %
Gravité prédite : [0 0 0 ... 0 2 0]
Accuracy pour la prédiction de gravité : 47.0 %
Gravité prédite : [0 0 0 ... 1 2 0]
Accuracy pour la prédiction de gravité : 47.15 %
Gravité prédite : [0 0 0 ... 1 0 0]
Accuracy pour la prédiction de gravité : 47.85 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 48.5 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 49.1 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 48.55 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 48.699999999999996 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 48.699999999999996 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 49.0 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 48.4 %
Gravité prédite : [0 0 0 ... 0 0 0]
Accuracy pour la prédiction de gravité : 47.8 %
Gravité prédite : [0 0 0 ... 0 0 0]

```

FIGURE 10 – Retour de la fonction KNN (K 1-12) - Scikit-learn

Si on devait comparer nos deux modèles. D'un point de vue global on remarque que les predictions sont très similaires à partir de K=5. On remarque aussi que l'Accuracy tourne autour des 48% à 49% de precision environ, que ce soit pour la version "from scratch" ou "scikit-learn" pour des valeurs de K similaires. La grosse différence revient surtout au niveau du temps d'exécution. D'un coté on a la version Scikit qui prend environ 1 seconde a l'exécution pour tester avec tous les K allant de 1 à la valeur entrée. Tandis que l'algorithme from scratch prend environ 2 min mais uniquement pour tester avec le K rentré en paramètre (ne teste pas avec les K inférieurs). La version from scratch est bien plus longue à executer que la version avec les bibliothèques de Scikit-learn.

```

Accuracy: 48.75 %

[0,
 0,
 0,
 0,
 0,
 0,
 0,
 2,
 0,
 2,
 0,
 0,
 0,
 0,
 1,
 2,
 2,
 1,
 0,
 2,
 0,
 1,
 1,
 2,
...
 2,
 1,
 2,
 2,
...]
```

FIGURE 11 – Retour de la fonction KNN (K 10) - From Scratch

5.3 Classification avec trois algorithmes de « haut niveau »

L'objectif de cette partie est de réaliser la classification selon la gravité de la base de données en utilisant 3 méthodes de "haut niveau". Pour limiter le nombre d'entrée dans la base de données nous avons préalablement réalisé un échantillon de la base de données. Pour ce faire nous avons calculé les proportions de chaque cas de `descr_grav`. Les valeurs dans notre échantillon sont faites des mêmes proportions, ainsi, nous espérons que notre échantillon reflète les paramètres de la base entière. Les 3 méthodes utilisées pour notre GridSearch sont :

- Support Vector Machine (SVM)
- Random Forest (RF)
- Multilayer Perceptron (MLP)

Dans chaque cas de GridSearch, nous utiliserons notre échantillon de 4% des données. Nous avons choisi de procéder comme cela car selon certains paramètres (Comme `linear` dans le SVM), un temps de compilation de 6 heures n'était pas suffisant même avec cet échantillon. Une fois le GridSearch réalisé avec chaque méthode, nous obtenons des résultats qui sont traitables comme : La précision (`accuracy`) : Il s'agit du ratio des prédictions correctes positives sur le nombre total de prédictions positives effectuées par le modèle.

Le rappel (`Recall`) il s'agit du ratio des prédictions vrais positifs sur le nombre total d'échantillons réellement positifs.

Le score F1 (`F1 Score`) : Le score F1 est la moyenne harmonique de la précision et du rappel. Le score F1 se situe entre 0 et 1.

Les meilleurs paramètres (`best params`) : Les meilleures paramètres font référence aux meilleurs paramètres trouvés pour un modèle. Ce sont les paramètres optimaux.

La matrice de confusion (`confusion matrix`) : Une matrice de confusion est un tableau utilisé pour évaluer les performances d'un modèle de classification. Elle résume les prédictions faites par le modèle sur un ensemble de données de test et les compare aux vraies étiquettes des données.

`resultat_cv` : `resultat_cv` renvoie un dictionnaire qui contient les informations détaillées de la validation croisée effectuée lors du gridsearch.

Le vote majoritaire : le vote majoritaire est de prendre la prédiction de chaque classifieur et de sélectionner la classe qui obtient le plus grand nombre de votes. Le vote effectue les prédictions en utilisant chaque classifieur individuel et compte les votes pour chaque classe. La classe qui obtient le plus grand nombre de votes est alors choisie comme prédiction finale.

5.3.1 Analyse des résultats

Support Vector Machine (SVM)

Meilleures paramètres SVM : `'C'` : 10, `'gamma'` : `'scale'`, `'kernel'` : `'rbf'`
Meilleur estimateur : `SVC(C=10)`
SVM Accuracy : 0.5449915110356537
Precision : 0.5070794374471712
Recall : 0.5449915110356537
F1 Score : 0.4761175146157662

La matrice de confusion est :

204	31	1	0
104	114	1	0
55	59	3	0
10	5	2	0

Matrice des résultats du SVM

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C
0	0.123796	0.007306	0.066400	0.003440	0.1
1	0.147599	0.011160	0.068401	0.001952	0.1
2	0.131794	0.004925	0.065200	0.003310	1
3	0.153598	0.003506	0.063608	0.001197	1
4	0.150010	0.002818	0.058397	0.000800	10
5	0.208605	0.009769	0.062793	0.001725	10
	param_gamma	param_kernel	params		
0	scale	rbf	{ 'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf' }		
1	auto	rbf	{ 'C': 0.1, 'gamma': 'auto', 'kernel': 'rbf' }		
2	scale	rbf	{ 'C': 1, 'gamma': 'scale', 'kernel': 'rbf' }		
3	auto	rbf	{ 'C': 1, 'gamma': 'auto', 'kernel': 'rbf' }		
4	scale	rbf	{ 'C': 10, 'gamma': 'scale', 'kernel': 'rbf' }		

Random Forest (RF)

Pour la random forest, nous obtenons ces resultats :

Meilleures parametres RF : 'max_depth' : 10, 'min_samples_split' : 10, 'n_estimators' : 200

RandomForestAccuracy : 0.5959252971137521

Precision : 0.5688217536417226

Recall : 0.5959252971137521

F1Score : 0.5706149747118916

Ce sont des scores qui reflètent bien les score obtenus lors de nos premiers modèles.

La matrice de confusion est la suivante :

203	20	13	0
83	109	27	0
34	44	39	0
6	3	8	0

Matrice des resultats de Random Forest

	mean_fit_time	std_fit_time	mean_score_time	std_score_time
0	0.288003	0.011884	0.016804	0.000755
1	0.551795	0.011473	0.032405	0.002239
2	1.405382	0.037318	0.076677	0.003012
3	0.232923	0.008128	0.015979	0.000633
4	0.472222	0.012092	0.028857	0.001285
5	1.181350	0.017630	0.072835	0.002720
6	0.207160	0.007785	0.014999	0.001092
7	0.421300	0.022515	0.029039	0.001651
8	1.007578	0.006443	0.065709	0.001113

Multilayer Perceptron (MLP)

Meilleures parametres MLP : 'activation' : 'tanh', 'alpha' : 0.01, 'hidden_layer_sizes' : (100,)

MLPAccuracy : 0.5806451612903226

Precision : 0.5874117821387864

Recall : 0.5806451612903226

F1Score : 0.5642970825956002

La matrice de confusion pour le modèle MLP est :

187	28	21	0
84	108	27	0
33	38	46	0
5	4	7	1

Matrice des resultats de MLP

	mean_fit_time	std_fit_time	mean_score_time	std_score_time
0	0.697175	0.142880	0.002406	0.000499
1	1.744775	0.625020	0.002603	0.000799
2	2.506425	0.743312	0.002397	0.000492
3	0.848412	0.145941	0.001985	0.000017
4	1.616800	0.339351	0.002594	0.000497
5	2.775401	1.263517	0.002801	0.000399
6	0.843057	0.239801	0.002212	0.000405
7	2.067837	0.561719	0.002600	0.000489
8	2.436997	0.736881	0.002801	0.000748

Fusion des différents modèles en utilisant le vote majoritaire :

SVM : 0.5449915110356537

RandomForestClassifier : 0.5925297113752123

MLPClassifier : 0.5772495755517827

FusionClassifier : 0.5823429541595926

Nous constatons que notre vote majoritaire en utilisant les 3 modèles nous permet d'avoir une précision générale de 0.582. Les performances sont donc accrues en vote majoritaire que les prédictions avec les modèles de RF et SVM.

6 Création de scripts

Dans cette partie, nous avons développé des scripts permettant de réaliser des tâches spécifiques en ligne de commande. Ces scripts ont été conçus pour faciliter l'utilisation des méthodes d'apprentissage non-supervisé, supervisé et de classification de "haut niveau" dans le contexte des accidents.

6.1 Script 1 : Script pour l'apprentissage non-supervisé

Le premier script que nous avons créé concerne la méthode d'apprentissage non-supervisé k-means. Il est conçu pour prendre en entrée la latitude et la longitude d'un accident, ainsi que les centroides des clusters. Son objectif est de déterminer le cluster auquel appartient l'accident donné. Le script retourne le résultat sous forme d'un fichier JSON qui indique le cluster auquel l'accident est attribué.

La commande d'exécution est la suivante :

```
python3 partie1.py latitude longitude centroids.csv
```

6.2 Script 2 : Script pour l'apprentissage supervisé

Pour l'apprentissage supervisé, le second script que nous avons réalisé, celui-ci concerne la méthode KNN, il récupère en entrée la liste des accidents qu'il va traiter pour avoir des valeurs numériques sur chaque donnée comme nous l'avons réalisé à la préparation. Et il récupère en arguments du programme, les valeurs de l'accident dont on doit prédire la gravité, à savoir : 'id.code_insee', 'latitude', 'longitude', 'descr_cat_veh', 'descr_agglo', 'descr_athmo', 'descr_lum', 'descr_etat_surf', 'description_intersection', 'age', 'place', 'descr_dispo_secu', 'descr_motif_traj' et 'descr_type_col'. Tous les arguments doivent être déjà convertis en valeur numérique sinon le programme ne fonctionnera pas. La sortie que fournit le programme est un JSON indiquant la gravité de l'accident qui est le résultat de la fonction KNN de scikit.

La commande a utilisé est :

```
python3 partie2.py id_code_insee latitude longitude descr_cat_veh descr_agglo
descr_athmo descr_lum descr_etat_surf description_intersection age place
descr_dispo_secu descr_motif_traj descr_type_col
```

6.3 Script 3 : Script pour les méthodes de classification de « haut niveau »

Pour le script de classification supervisée, nous utilisons les modèles enregistrés dans la partie 5. La commande prend en entrée les 8 valeurs correspondant à une information sur un accident : descr_cat_veh, descr_agglo, descr_athmo, descr_lum, descr_etat_surf, descr_dispo_secu, descr_grav, descr_type_col. En deuxième argument, nous avons la méthode de prédiction, entre SVM, RF et MLP un exemple est :

```
python3 scripts/partie3.py "2,5,2,4,40.0,0,2" RF
```

Ces scripts ont été créés dans le but de fournir des informations à notre future application et de simplifier l'automatisation des tâches liées à l'analyse des accidents. Ils offrent la possibilité d'effectuer des calculs complexes et de traiter efficacement de grandes quantités de données en utilisant des commandes simples.