ISEN
ALL IS DIGITAL!
NANTES

yncréa

PROJET IA

Analyse des accidents de la route en 2009

Aubin Saunier

Maxime De Roquelle

Paul Poirier

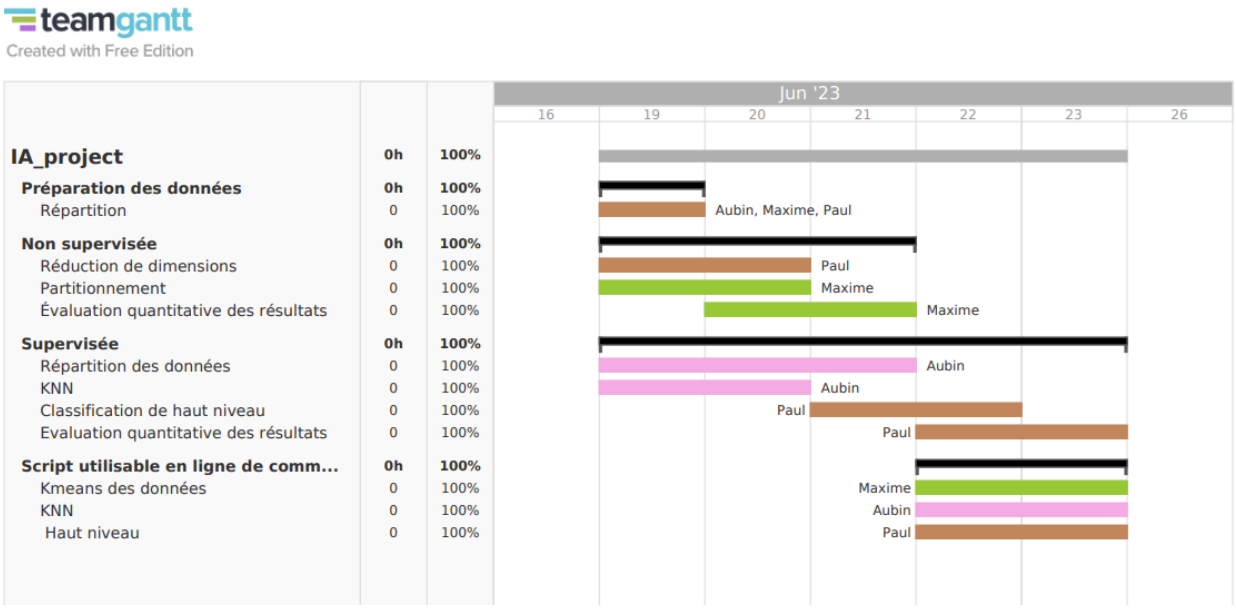
Table des matières

Présentation du projet

Présentation du projet

Ce projet de fin d'année A3 à ISEN Nantes consistait à étudier et analyser une base de données sur les accidents de la route de 2009. Dans ce projet, nous avions plusieurs tâches : préparer les données, utiliser des méthodes de classification, les analyser, puis des scripts utilisables pour la partie WEB. Le jeu de données a été préparé et modifié par les professeurs, sur une base de données du gouvernement.

Répartition du travail



Découverte et préparation des données

Organisation des données

Nous avons transformé les données se trouvant dans un fichier csv en dataframe pandas, grâce à la fonction `read_csv()` de la librairie pandas.

Découverte des données

Afin de rendre notre base de données exploitable, nous devons d'abord préparer ces données. Pour cela, il nous faut déjà les comprendre, et ainsi les modifier pour pouvoir rendre la prédiction et l'analyse plus facile.

Voilà ce qu'on retient de la base donnée appelé « stat_acc_V3.csv ». C'est un fichier CSV qui contient 22 colonnes et 73643 lignes. Celui-ci relate des informations sur des accidents de la route parus en 2009. Les 22 colonnes/classes en question représentent respectivement :

- Le numéro de l'accident, correspond l'identifiant de l'accident (de type **entier**)
- Le « numéro véhicule », identifiant du véhicule repris pour chacun des usagers occupant ce véhicule (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) (de type **caractère**)
- L'identifiant USA (de type **entier**) ?? on ne sait pas ce que cela est diantre
- La date de l'accident, on peut y voir le jour, le mois, l'année, l'heure, la minute et même les secondes (de type **caractère**)
- La commune ou s'est déroulé l'accident (de type **caractère**)
- Le code INSEE, les 2 premiers chiffres correspondant au code postal du département, et les 3 suivants le code unique pour chaque commune (de type **caractère**)
- La latitude précise ou s'est déroulé l'accident (de type **caractère**)
- La longitude précise ou s'est déroulé l'accident (de type **caractère**)
- Le type de véhicule (ex : poids lourd, bicyclette etc…) (de type **caractère**)
- La description si l'accident s'est déroulé en agglomération ou pas (de type **caractère**)
- La description atmosphérique lorsque l'accident s'est déroulé (de type **caractère**)
- La description de la lumière au moment de l'accident (de type **caractère**)
- La description de l'état de la surface au moment de l'accident (de type **caractère**)
- La description de l'intersection (ex : giratoire, hors intersection etc…) (de type **caractère**)
- L'année de naissance de la personne (de type **caractère**)
- L'âge de la personne au moment de l'accident (de type **caractère**)
- La place de la personne au moment de l'accident (ex) 1 correspondant à la place conducteur, 2 à la place passager etc…) (de type **caractère**)
- La description de l'utilisation de la sécurité (ex) utilisation de la ceinture de sécurité ou pas etc…) (de type **caractère**)
- La description de l'état de santé de la personne (de type **caractère**)
- La description du motif du trajet (de type **caractère**)
- La description de l'accident (ex) deux véhicules frontales / de cotés…) (de type **caractère**)

On comptabilise 73643 lignes, donc 73643 personnes impliqués dans un accident en 2009. A savoir qu'un accident peut toucher plusieurs personnes. Maintenant que nous avons compris notre base de données, nous allons pouvoir traiter ces informations.

Préparation des données

Dans cette partie nous n'avons rien fait en python car toute la préparation a été faite en big data avec le langage R. Pour pouvoir utiliser certaines données, nous avons numérisé plusieurs données qui étaient sous forme de chaines de caractère. Pour cela, nous avons attribué des valeurs équivalentes pour la catégorie du véhicule, la description de la gravité, l'état de la surface et la description atmosphérique comme suit :

Type	Masse (kg)
Autobus	20000
Autocar	20000
Autre véhicule	10
Bicyclette	20
Cyclomoteur <50cm3	55
Engin spécial	30000
Motocyclette > 125 cm3	150
Motocyclette > 50 cm3 et <= 125 cm3	125
PL > 3,5T + remorque	4500
PL seul > 7,5T	7500
PL seul 3,5T <PTCA <= 7,5T	11000
Quad léger <= 50 cm3 (Quadricycle à moteur non carrossée)	200
Quad lourd > 50 cm3 (Quadricycle à moteur non carrossée)	500
Scooter < 50 cm3	55
Scooter > 125 cm3	180
Scooter > 50 cm3 et <= 125 cm3	110
Tracteur agricole	8650
Tracteur routier + semi-remorque	38000
Tracteur routier seul	7000
Tramway	30000
VL seul	1240
Voiturette (Quadricycle à moteur carrossée) (Anciennement "voiturette ou tricycle à moteur")	800
VU seul 1,5T <= PTAC <= 3,5T avec ou sans remorque	2000
Train	1500000

Après modification du fichier csv. Nous possédons 10 classes supplémentaires de qui contient des entiers positifs.

Apprentissage non supervisée

Réduction de dimension

La réduction de la dimensionnalité est une méthode d'apprentissage non-supervisé, qui consiste à prendre des données dans un espace de grande dimension, puis à les remplacer par des données dans un espace de plus petite dimension. Pour que l'opération soit utile il faut que les données en sortie représentent bien les données d'entrée.

Tout d'abord on peut réduire le nombre de dimension de notre base de données en supprimant les classes que l'on juge non représentative dans notre étude. En l'occurrence nous cherchons à lier les caractéristiques d'un accident et la gravité de ce dernier. Nous pouvons donc enlever les classes qui n'ont pas de lien avec la gravité : la date, l'identifiant de l'accident, etc.

Voici les classes conservées pour le reste de l'étude :

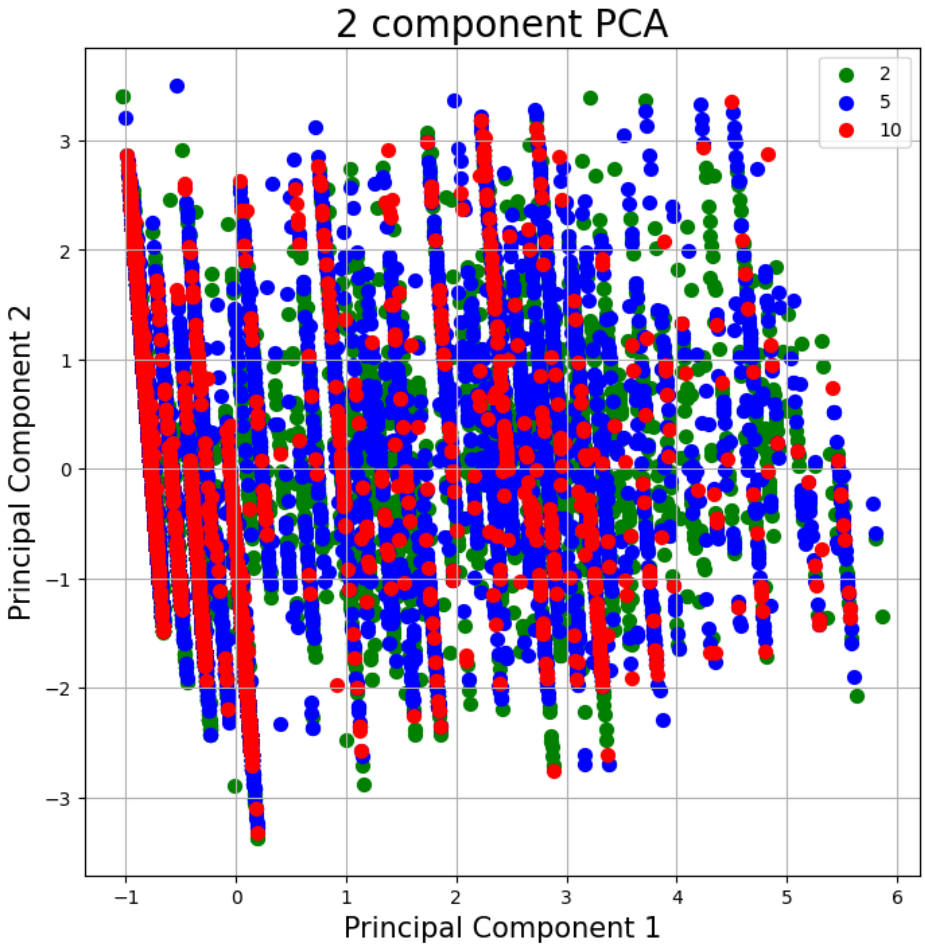
```
print(data.head())
```

```
    latitude  longitude  age  weight  gravity  hours  athmo_num  \
0  47.487718  -1.216426   40    7500        0      8          6
1  45.595814   5.655880   50    2000        0     11          5
2  45.595814   5.655880   49    1240        0     17          5
3  42.793565   3.002079   52    1240        0     20          4
4  45.970826   0.050529   58    1240        0     17          0

   etat_surf_num  lum_num  agglo_num
0              7        1          1
1              5        0          1
2              5        0          1
3              6        4          1
4              0        0          1
```

Une autre méthode pour réduire la dimension de nos données est d'utiliser un algorithme qui fait appel à l'Analyse en Composante Principale. Cette méthode consiste à transformer des variables liées entre elles en nouvelles variables décorrélées les unes des autres. Ainsi on obtient des données avec autant d'information qu'auparavant mais avec un nombre de classe réduit.

Avec python on utilisera la classe PCA de la librairie SKlearn. Avec la fonction associée, on transforme les 10 classes en 2 nouvelles classes. On obtient alors la répartition suivante sur un graphique.



Clustering

Une méthode non supervisée est appelée clustering ou partitionnement en français. Cette méthode a pour but de regrouper certaines instances dans des groupes appelés « clusters ». Nous avons utilisé deux fonctions pour ce partitionnement. La première vient de skit learn, et nous avons codé la seconde nous-même, dit « from scratch ».

Le code « from scratch » est une fonction qui prend en entrée la base de données, un nombre de clusters k, le type de distance, ainsi qu'un nombre d'itération. L'algorithme possède plusieurs étapes ;

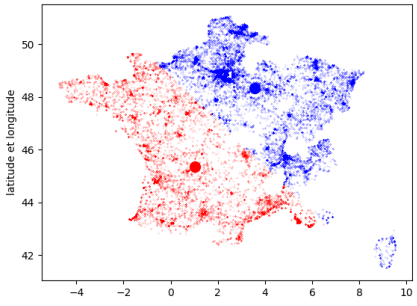
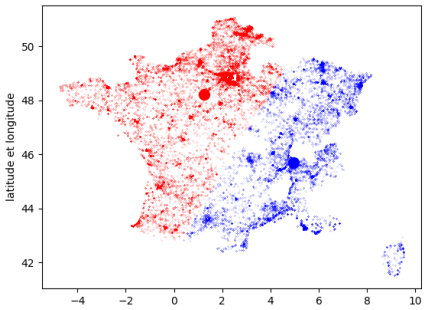
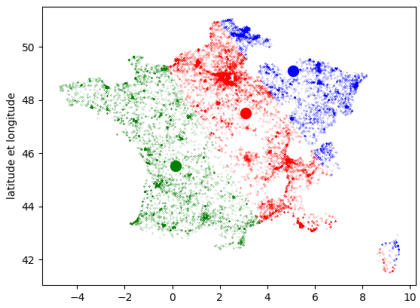
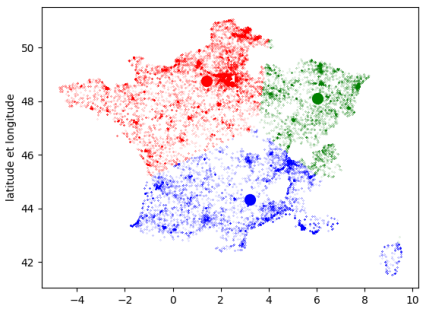
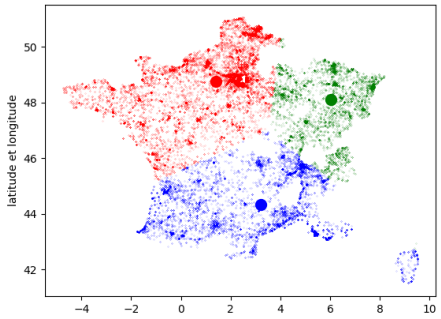
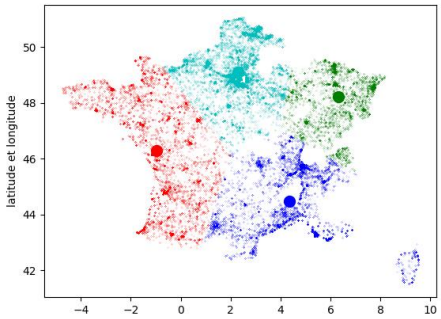
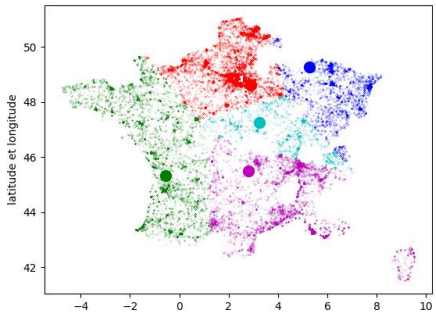
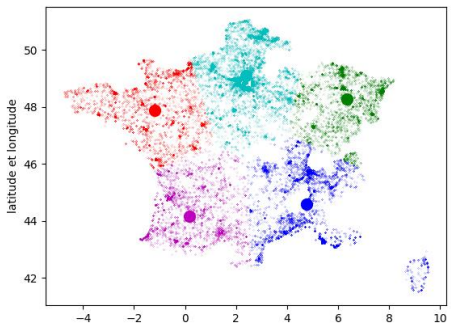
- La première consiste à attribuer aux hasards des coordonnées aux centroïdes des clusters parmi des instances de la base de données.
- La seconde étape calcul la distance, en fonction de l'argument en entrée, entre chaque instance et les coordonnées des centroïdes. On leur attribuera aux clusters ayant la distance la plus petite.
- La troisième étape va, pour chaque cluster, faire la moyenne des coordonnées des instances attribuées à celui-ci. Puis modifier ainsi les coordonnées de chaque centroïdes avec ces moyennes.
- Enfin, on alterne entre la seconde étape et la troisième étape en fonction du nombre d'itération mis en entrée de la fonction

En sortie de la fonction, on retrouve un tableau qui contient ;

- Les coordonnées des centroïdes des k clusters
- Un tableau contenant l'indice du cluster attribué à chaque instance.

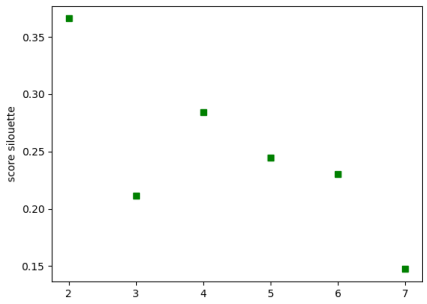
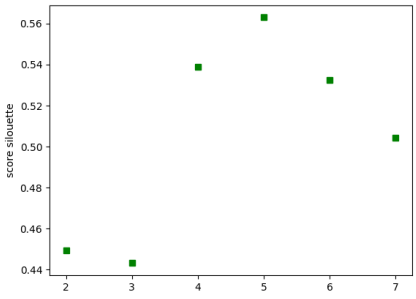
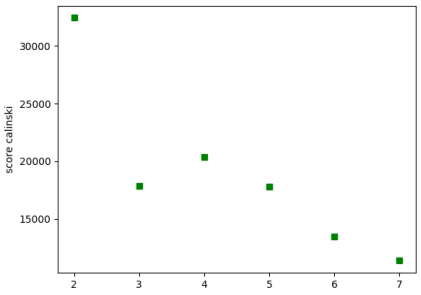
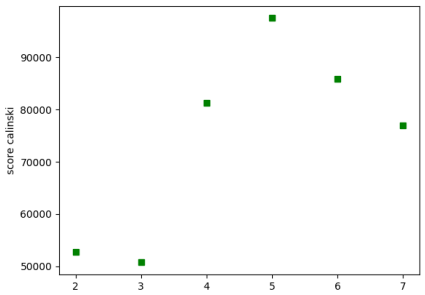
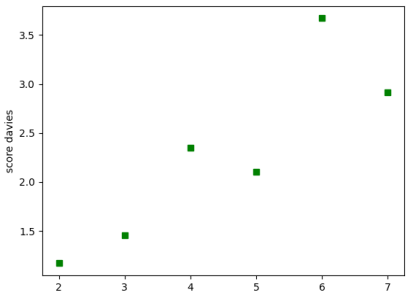
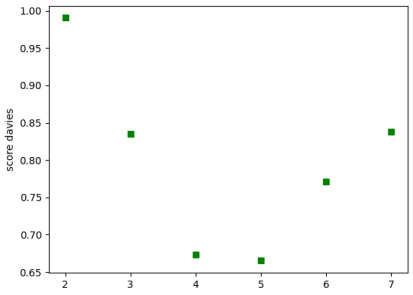
Nous allons maintenant comparer ces deux fonctions à l'aide de cartes, de tableaux et de fonctions d'évaluation quantitative, le tout en variant k, le nombre de clusters.

A noté que pour l'affichage sur les cartes, nous nous sommes contentés de la France métropole.

k	From scratch	Sklearn
2		
3		
4		
5		

Évaluation quantitative des résultats « non supervisé »

Nous avons réalisé des cartes avec des k encore plus élevé mais il ne nous semble pas intéressant d'en présenter davantage. Sinon, nous voyons bien que notre fonction fonctionne bien, chaque accident est attribué à un cluster qui lui ai proche. Il est difficile de juger la performance de notre algorithme graphiquement, nous allons donc maintenant utiliser des fonctions d'évaluations.

Méthode	From scratch	Sklearn
Silhouette score		
Calinski-Harabasz		
Davies-Bouldin		

Comme on peut le voir, les fonctions d'évaluations s'accorde pour dire que le meilleur k, dans le cas de la fonction sklearn, est 5. En effet, les coefficients de silhouette et calinski-harabasz sont les plus élevé pour k = 5. De plus, le score de Davies est le plus bas, donc là où la classification est la meilleur pour k = 5 également.

En revanche, pour notre fonction « from scratch », le coefficient de silhouette est le plus élevé pour k = 2. Les méthodes de Davies et Calinski s'accorde pour dire que le k le plus adapté est k = 2. Nous expliquons ces différences par le fait que notre algorithme créer les premières centroïdes de manière aléatoire, ce qui influe sur les performances, même si plus le nombre d'itérations est grand,

plus on se rapprochera du modèle de sklearn, mais à un cout en temps très élevé. Par conséquent, le hasard est moindre lorsque k = 2 vu que c'est le minimum de cluster possible.

D'un point de vue performance sur le temps, notre fonction prend environ 20 secondes à s'exécuter avec un nombre d'itérations égale à 10. Tandis que la fonction kmeans de sklearn prend moins d'une seconde. Nous en concluons qu'une méthode « brut force » n'est pas forcément optimal. Au niveau des différentes méthodes de calcul de distance, nous ne distinguons pas de modification de temps de d'exécution ou de modification performance significatif.

Apprentissage supervisé

Répartition des données

Pour la répartition des données nous avons utilisé deux méthodes : le holdout et le leave-one-out. La méthode holdout consiste à prendre 80% de la base de données en base d'apprentissage et les 20% restant en base de test. On fait ce test cinq fois. La méthode leave-one-out consiste à enlever une seule instance de la base de données pour le test et à utiliser le reste en base d'apprentissage. On effectue ce test en retirant chacune des instances de la base de données une à une (n fois).

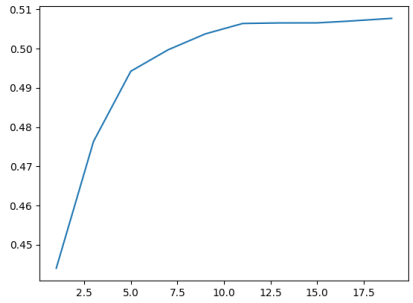
Classification avec KNN

La méthode KNN (ou k-nearest-neighbors) consiste à chercher dans la base de données les données les plus proches d'une nouvelle donnée qu'on donne. Pour cela, on calcule la "distance" entre cette donnée et chacune des lignes de la base de données. Cette distance peut être calculée avec différentes méthodes, en considérant les valeurs de la donnée et des lignes de la base de données comme des coordonnées de vecteur. Lorsque les distances ont été calculées, on cherche les k plus petites distances. k n'est pas une donnée universelle, il faut trouver le k optimal.

La méthode KNN a été codée "from scratch". Nous allons comparer cette version avec celle de scikit learn.

Tout d'abord, d'un point de vue temps d'exécution, la méthode "from scratch" prend 5.34 secondes tandis que celle de sklearn prend 0.62 secondes. Notre fonction est donc 8.6 fois plus longue. Nous voulions faire un test comparant différentes valeurs de k (3, 7, 11, 15) avec chacune des méthodes de calcul de distance (euclidien, manhattan, minkowski, hamming), en testant à chaque fois sur 20% de la base de données afin de trouver les meilleurs paramètres. Cependant, c'est impossible : cela prendrait plus de 2 semaines.

Pour ce qui est de l'accuracy, on peut cependant noter que le modèle "from scratch" est plus efficace : il a un taux de réussite de 65% contre 51% pour la fonction de sklearn.



Pour ce qui est de la valeur de k, nous avons fait des tests (avec la fonction de sklearn avec k variant de 1 à 19 et de 2 en 2 en observant l'accuracy (graphique ci-contre). Nous avons fait varier sur des valeurs impaires car c'est ce qui est conseillé pour knn. Nous en avons conclu que la meilleure valeure pour k est 11, car au-delà il n'y a plus vraiment de variations.

Classification avec trois algorithmes de « haut niveau »

Le premier des 3 algorithmes que nous allons utiliser est le Support Vector Machine (ou SVM). Il retranscrit les exemples d'entraînement sur des points dans un espace afin de maximiser la largeur

de l'écart entre deux groupes de points. De nouveaux exemples sont ensuite cartographiés dans ce même espace et prédits comme appartenant à une catégorie en fonction de quel côté de l'écart ils se situent.

Le 2ème algorithme de classification est Random Forest. Cette méthode repose sur les arbres de décisions : une série de test pour catégoriser chaque donnée.

Le dernier algorithme est le Multilayer Perceptron (ou MLP), se basant sur Perceptron un algorithme de classification binaire. Il est considéré comme le réseau de neurone le plus simple. Le perceptron est composé d'une succession de décision binaire telle que OU ou AND logiques. Le Multilayer Perceptron utilise plusieurs couches de décision et combiner les décisions binaires. On peut donc résoudre des problèmes plus complexes.

Avec chacun des algorithmes on cherchera les meilleurs paramètres à utiliser pour des résultats optimaux. Pour cela on utilisera la fonction GridSearchCV de la librairie SKlearn. En indiquant les paramètres et les différentes valeurs pour ces derniers, GridSearch va tester toutes les combinaisons et retourner la plus efficiente.

SVM :

- C : 10
- Kernel : 'rbf'
- Gamma : 'scale'

Random Forest :

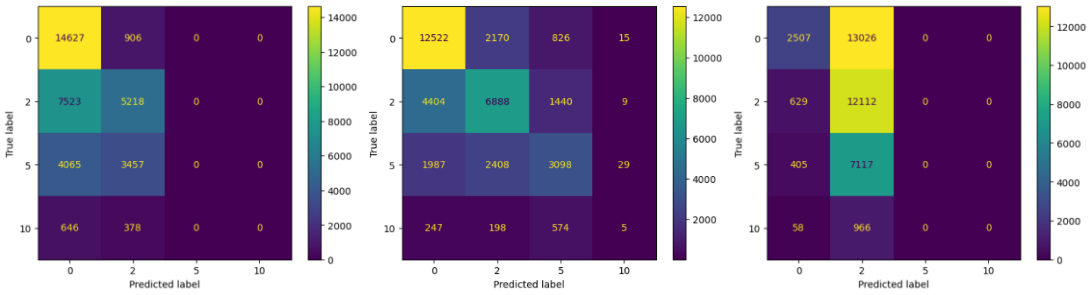
- max_features : 'sqrt'
- max_depth : None
- n_estimatorsn : 1000

MLP :

- hidden_layer_sizes : 50

Évaluation quantitative des résultats « supervisé »

Enfin on calculera le score d'accuracy, de précision et de rappel puis on affichera la matrice de confusion pour chacun des algorithmes afin de les départager.



SVM :

- Précision : 0.54
- Rappel : 0.27
- Accuracy : 0.34

Random Forest :

- Précision : 0.56
- Rappel : 0.44
- Accuracy : 0.41

MLP :

- Précision : 0.45
- Rappel : 0.39
- Accuracy : 0.39

Création de scripts utilisables en ligne de commande

Nous avons créé 3 scripts pour préparer le backend de la partie WEB. Les syntaxes sont comme suit :

```
python.exe kmeans.py lat lon "[[lat_centroid1, lon_centroid1], [lat_centroid1, lon_centroid1], ..., [lat_centroidn, lon_centroidn]]"
python.exe knn.py "[lat, lon, age, weight, hour, athmo, surf, lum, agglo]" path/to/accidents_database.csv
python.exe high_level.py "[lat, lon, age, weight, hour, athmo, surf, lum, agglo]" <random_forest|MLP|SVC>
```