Rapport Apprentissage Automatique

Projet Kaggle, prédiction de la gravité d'un accident

Amaury Bodin

Le but de ce projet est de réussir à créer un modèle qui serait capable de prédire la probabilité qu'un accident ayant eu lieu en France soit grave ou non. Pour ce faire, ils nous ont fourni les données de 2012 à 2023 sur les accidents de la route en France. Toutes ses données sont associées à un numéro d'accident (nommé Acc_num dans les datasets), et chaque accident à un nombre de variable pour exprimer ainsi bien les conditions de la route, que la météo mais aussi le type de véhicule, l'âge de la personne etc.

Pour débuter le projet j'ai d'abord commencé par merge ensemble toutes les données des datasets de 2012 à 2023 sans faire aucun tri ni modification, cela va nous donner un dataframe pandas extrêmement grand, avec beaucoup de lignes et de variables. Avant même de commencer à faire des modifications sur ce grand jeu de donnée complet on remarque qu'il y a des duplicates de Num_acc, cela est dû aux faites qu'il peut y avoir plusieurs personnes différentes impliqué dans un accident, on a donc plusieurs fois le même accident mais avec différente données selon la personne, ceci est un élément très important à prendre en compte car il faudra avant d'entraîner notre modèle enlevé ces duplicates, mais nous nous occuperons de cela plus tard.

Nous allons d'abord commencer à effectuer du features engeneering avant même de parler de sélection de modèle ou d'entraînement étant donné que cette partie est commune aux trois soumissions. Il y a aussi un grand nombre de données, elles ne sont pas toutes exploitables dans leurs états initiaux et certaines peuvent même être considérées comme inutiles dans les prédictions de la gravité d'un accident.

La première chose que nous allons faire est pour la variable de l'heure, au lieu d'avoir toutes les heures disponibles nous allons découper les heures en 4 catégories : entre 00h et 6h ce sera 0, entre 6h et 12h ce sera 1, 12h et 18h 2, et le reste 3. Cela nous permettra d'avoir beaucoup mains de valeur unique le modèle aura donc moins de soucis et aussi de voir si le moment de la journée à vraiment un impact significatif sur la gravité de l'accident.

Ensuite une modification très importante nous allons transformer la variable grave qui a de base 4 valeur possible, en seulement deux car il nous ai demandé de prédire si un accident était grave, c'est donc binaire. Pour cela 1 et 4 deviendront 0 donc pas grave et 2 et 3 deviendront 1 donc grave.

Ensuite on fait le choix, assez arbitraire, des colonnes que l'on ne juge pas assez influentes sur la gravité d'un accident. Ces colonnes représentent souvent la situation géographique, la date, mais aussi des composantes mécaniques de la voiture ou elles peuvent tout simplement avoir une trop grande quantité de données manquantes. Voici donc la liste :

```
cols_to_drop = ['mois','jour', 'com','dep', 'adr', 'gps', 'lat', 'long', 'voie', 'v1', 'v2', 'pr', 'pr1', 'num_veh_x', 'id_vehicule_x','id_vehicule_y','num_veh_y', 'id_usager','motor','secu2','secu3']
```

On va ensuite calculer l'âge du conducteur au moment de l'accident en soustrayant l'année de l'accident à son année de naissance. On va ensuite venir créer des catégories plutôt que de laisser un grand nombre de données unique avec les moins de 20 ans, 40 ans, 65 ans et les plus de 65 ans ainsi que à -1 les valeurs manquantes. Cela nous permettra par la suite de faire une meilleure sélection des duplicates à garder.

Pour la prochaine transformation nous avons fusionné les colonnes secu et secu1 qui sont globalement la même chose mais secu1 apparaît après les années 2020 alors que secu est présente depuis le début. Après la fusion nous avons de nouveau créer 4 catégories différentes, pour mieux différencier les types de protection mais aussi pour limiter le nombre de valeurs unique et manquante.

Une autre modification importante que nous allons faire avant d'entraîner quelconque modèle, est sur la catégorie du véhicule. Dans le dataset de base il y a presque 90 catégories différentes de véhicule ce qui fait un bien trop grand nombre de valeurs unique pour une catégorisation, ce qui peut donc être très dur pour notre modèle de réussir à faire des regroupements significatifs qu'il lui permette de comprendre les raisons d'un accident grave. Nous allons donc transformer toutes ces catégories en seulement 6 catégories de véhicules : les vélos et deux roues, les voitures, les véhicules lourds (types camions etc.), les transports en commun et pour finir les engins spéciaux et autres.

Ensuite nous allons essayer de traiter la variable VMA, elle indique la vitesse maximale à laquelle un véhicule peut rouler sur cette route. Lorsque l'on regarde les données dans le dataset de base, il y a beaucoup de valeur manquante, ainsi que d'outliers, pour régler ce problème nous allons d'abord regarder le type de route (qui est une autre variable fournie et qui a très peu de valeur manquante, nous allons donc pouvoir affecter une vitesse selon le type de route pour remplacer les valeurs manquantes. Et pour finir nous allons vérifier les outliers en revérifiant la catégorie de leur route, et si rien de correspond on met arbitrairement 50 car c'est le type de route le plus présent en France.

Les deux dernières modifications de variable que nous allons faire sont sur occutc et actp pour enlever les valeurs aberrantes, les valeurs non numériques ainsi que les valeurs manquantes.

Nous allons ensuite remplir les variables ou il y a des valeurs manquantes par -1, voici la liste de ces variables :
['atm', 'col', 'circ', 'vosp', 'prof', 'plan', 'surf', 'infra', 'situ','senc', 'occutc', 'obs', 'obsm', 'choc', 'manv', 'trajet', 'locp', 'etatp', 'actp'].

Maintenant que notre features engineering est fini nous allons nous occuper des drop nos duplicates. Pour cela nous utilisons une façon spécifique de le faire. En effet ici le but est de garder la ligne des duplicates qui maximise l'accident, il va donc falloir garder la meilleure valeur de chaque variable. Pour ce faire nous allons regarder chaque variable et décider de la valeur qui maximise l'accident. Ici tout est arbitraire et donc par exemple pour la catégorie de véhicule nous allons garder la valeur 1 qui correspond aux vélos et véhicules à deux roues. Suivant cette logique je repartis en trois les variables : celles dont il faut maximiser la valeur, celle où nous allons prendre la valeur la plus présente et celle dont nous voulons

choisir 1 s'il est présent, on obtient donc :
max_cols = ['hrmn', 'vma', 'age_conducteur', 'occutc', 'catu','nbv','grav']

mode_cols = ['lum', 'agg', 'int', 'atm', 'col', 'circ', 'vosp', 'prof', 'plan', 'surf', 'infra', 'situ', 'obs', 'obsm', 'manv', 'trajet', 'locp', 'etatp', 'actp']

target_cols = ['catv', 'col','vosp','senc', 'choc', 'secu_combined', 'sexe']

avec la fonction suivante :

```python
def process_duplicates(df, target_cols, max_cols, mode_cols):
    grouped = df.groupby('Num_Acc')
    processed_data = []

    for _, group in grouped:
        processed_row = {}
        for col in df.columns:
            if col in max_cols:
                processed_row[col] = group[col].max()
            elif col in mode_cols:
                mode_values = group[col].mode()
                processed_row[col] = mode_values[0] if not mode_values.empty else -1
            elif col in target_cols:
                if 1 in group[col].values:
                    processed_row[col] = 1
                else:
                    mode_values = group[col].mode()
                    processed_row[col] = mode_values[0] if not mode_values.empty else -1
            else:
                processed_row[col] = group[col].iloc[0]

        processed_data.append(processed_row)

    processed_df = pd.DataFrame(processed_data)
    return processed_df
```

Tous les duplicates ayant maintenant la même valeur il n'est plus important de savoir qui sera drop en appelant la fonction drop_duplicates

L'étape suivante ne correspond qu'au fichier nommé Projet_v4_dummies. Ou nous allons transformer toutes nos variables, qui peuvent l'être, en dummies.

```python
columns_to_convert = ['hrmn','agg','lum','int','atm','col','catr','circ','vosp','prof','plan','surf','infra','situ',
                      'senc','catv','obs','obsm','choc','manv','catu','sexe','trajet','vma','locp','actp','etatp',
                      'age_conducteur','secu_combined']

df_dummies = pd.get_dummies(data_1, columns=columns_to_convert, drop_first=True)
```

Nous allons ensuite passer pour toutes les soumissions à la partie sélections des variables importantes. Cette partie nous permet de garder les variables qui ont le plus d'impact, surtout lors de l'utilisation de fichier avec les dummies. Pour ce faire nous utilisons un Lasso, qui va nous permettre de mettre les coefficients des variables moins importantes et donc de garder que les variables avec un coefficient différent de zéro. On applique notre Lasso sur un sample de nos datas de 60 000 lignes et on trouve qu'un lasso avec un coefficient de 0.002 nous donne le meilleur résultat c'est donc ce que l'on va utiliser dans nos trois fichiers.

Une fois nos variables sélectionnées nous allons pouvoir passer créer notre sample de données de 70 000 données (soit un peu plus de 10% de toutes les données disponibles) avec les bonnes variables et nous allons pouvoir commencer votre entraînement.

Pour les fichiers Projet_v4 ainsi que Projet_v4_dummies nous allons utiliser un SVM du nom de Gradient Boosting couplé à une cross validation. Cela va nous permettre d'obtenir des

modèles prédictifs précis tout en réduisant le risque de surajustement et en optimisant les hyperparamètres du modèle.

Pour le fichier Projet_v4_random_forest nous utilisons une forêt de classification aléatoire avec une cross validation. Cela nous permet d'avoir une méthode d'apprentissage automatique qui construit plusieurs arbres de décision sur des sous-ensembles aléatoires des données d'entraînement, puis combine leurs prédictions pour améliorer la performance prédictive et réduire le surajustement. En utilisant la validation croisée, chaque sous-ensemble de données est divisé en segments pour entraîner et évaluer le modèle, assurant ainsi une estimation robuste des performances et une meilleure généralisation du modèle sur de nouvelles données.

Ayant divisé nos données en train et test, nous pouvons maintenant, faire l'entrainement et le fit puis faire des prédictions pour évaluer notre modèle. Pour avoir les prédictions nous utilisons predict_proba qui nous donnera la probabilité que la variable grave valle 1. Nous allons ensuite évaluer notre modèle en regardant L'AUC ainsi que ça courbe. Lorsque nous avons un bon résultat et pas trop d'overfitting on remarque que l'AUC que l'on obtient avec notre test est très similaire voire égale à celle donnée lorsque l'on dépose une prédiction sur Kaggle.

Pour déposer notre prédiction sur Kaggle, il nous suffit d'importer les données de Test et d'utiliser les mêmes fonctions que sur le train pour effectuer le features engeneering, le drop de duplicates et la création des dummies si besoin. Nous avons après plus qu'à utiliser predict_proba ainsi que à les sauvegarder un fichier csv au bon format.