



Rapport de projet de Fouille de données - F3

Jeu de données “santé”

Saunier Paul-Emeric

2020-2021

1- Analyse et visualisation avec R

a) Exploration des données

Le jeu de données utilisé, *Cholesterol_R.csv* provient d'une étude sur l'impact de la consommation de margarine sur la cholestérolémie¹, dans le cadre d'un régime pauvre en graisses et en cholestérol, sur 18 patients. Chaque patient a consommé une quantité équivalente à 2.31g d'esters de stanols (ingrédient actif) par jour. Des mesures de la cholestérolémie ont été réalisées avant le régime, puis après 4 et 8 semaines de régime.

Le dataset contient les colonnes suivantes :

- **ID** : identifiant pour désigner chaque patient.
- **Before** : cholestérolémie avant le régime
- **After4weeks** : cholestérolémie après 4 semaines de régime
- **After8weeks** : cholestérolémie après 8 semaines de régime
- **Margarine** : type de margarine consommé (A ou B) par le patient

L'intérêt de ce jeu de données est de voir l'évolution de la cholestérolémie pendant un régime pauvre en graisses et en cholestérol, en fonction du type de margarine consommée. Un taux trop élevé de cholestérol dans le sang est considéré comme un facteur de risque cardio-vasculaire.[ref1]

Le dataset comprend deux populations : les patients qui ont consommé de la margarine de type A et ceux qui ont consommé de la margarine de type B. Il n'y a pas de groupe témoin (qui n'aurait pas consommé de Margarine pendant le régime).

La cholestérolémie moyenne avant le régime de tous les patients du jeu de données est de 6.408 mmol/L (unité non-précisée dans les données mais semble cohérente). Elle est nettement supérieure à la valeur moyenne mondiale de 4.64 mmol/L^[ref. 2]

Une exploration manuelle du dataset montre que toutes les valeurs de cholestérolémie fournies sont plausibles. (Max 8.43 mmol/L, Min 3.66 mmol/L).

¹ Cholestérolémie : concentration sanguine en cholestérol

b) Feature engineering

Après import des données dans R, il est possible de calculer la modification de la cholestérolémie pour chaque patient après 4 et 8 semaines de régime avec les formules :

$$\begin{aligned} \text{Variation après 4 semaines de régime} = \\ \text{Cholestérolémie après 4 semaines} - \text{Cholestérolémie avant le régime} \end{aligned}$$

$$\begin{aligned} \text{Variation après 8 semaines de régime} = \\ \text{Cholestérolémie après 8 semaines} - \text{Cholestérolémie avant le régime} \end{aligned}$$

La variation moyenne de cholestérolémie obtenue pour tous les patients après 4 semaines de régime est de -0.5661 mmol/L, et de -0.6289 mmol/L après 8 semaines.

c) Visualisation

Trois visualisations ont été sélectionnées pour cet exercice. La première est un diagramme en barres superposées présentant la variation du taux de cholestérol pour chaque patient du jeu de données :

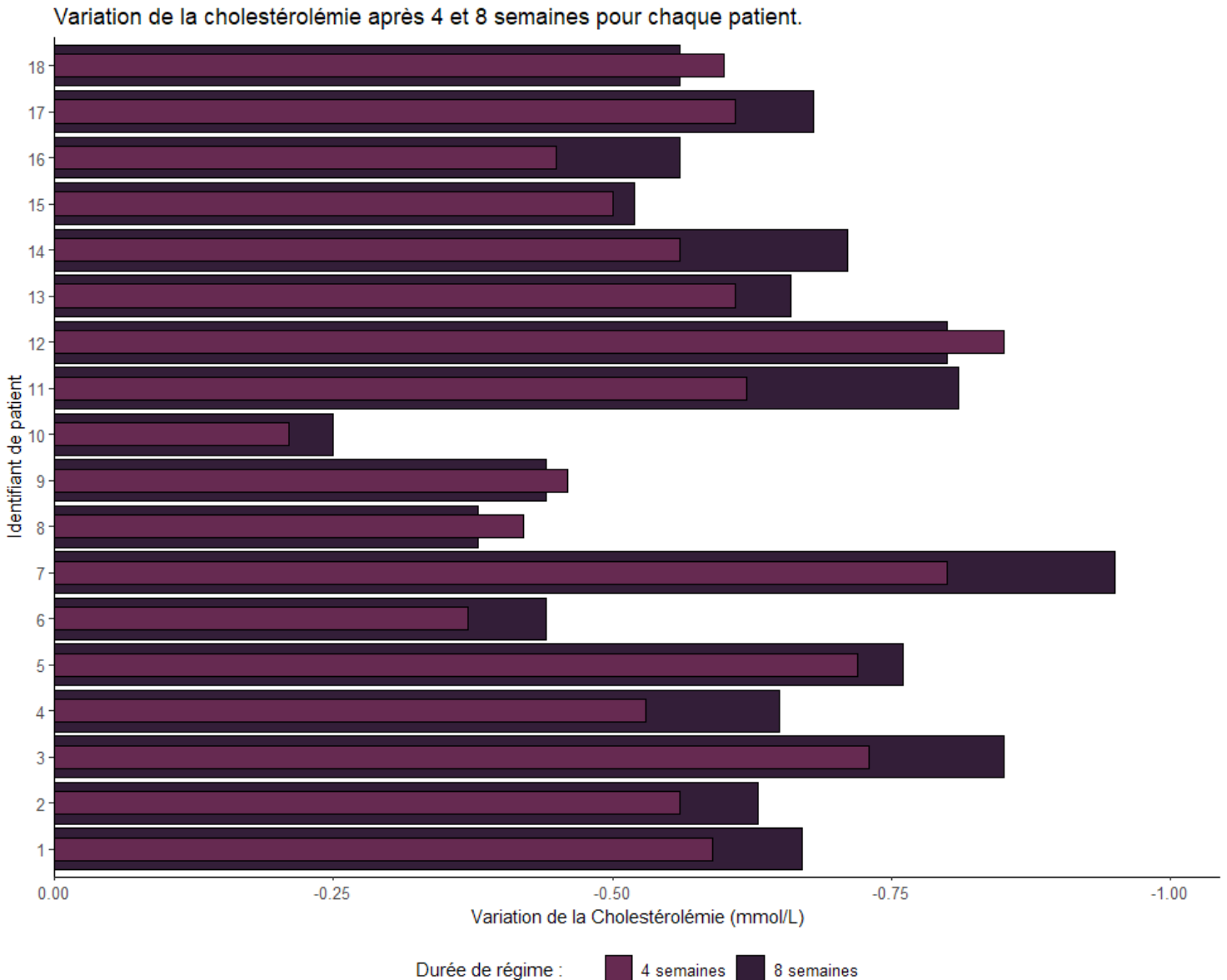


Fig. 1 : Variation de la cholestérolémie pour chaque patient en fonction de la durée de régime

Toutes les mesures du dataset montrent une diminution du taux de cholestérol chez les patients après le régime.

Les deux visualisations suivantes sont des Boxplots :

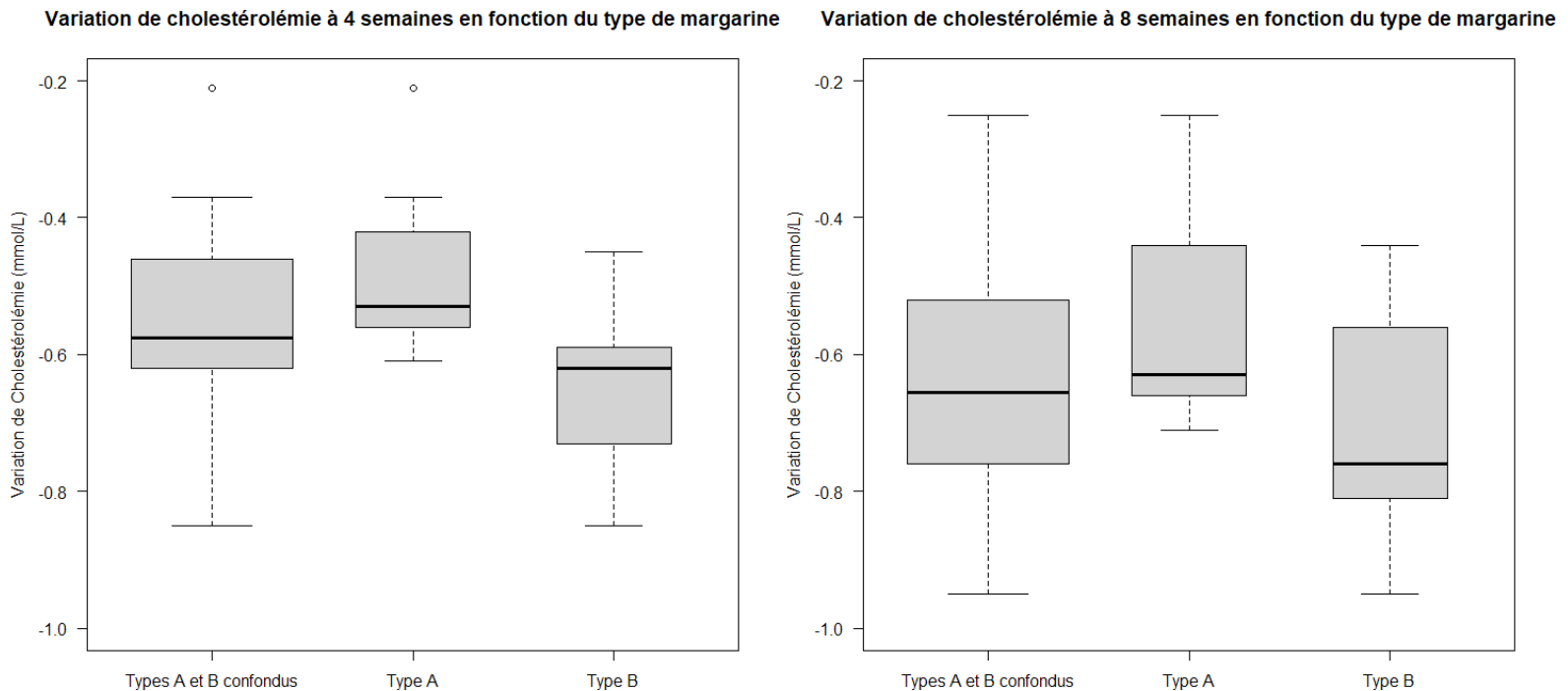


Fig. 2 et 3 : Boxplots de la variation de cholestérolémie après 4 et 8 semaines de régime

Les boxplots représentent pour chaque groupe :

- les valeurs minimales et maximales
- La médiane (trait noir)
- Les 1^{ers} et 3^e quartiles (bords de la boîte)
- Les points anormaux (cercles sur la Fig. 2)

Deux groupes de patients sont identifiables dans le dataset. Le premier est composé des patients qui ont consommé la margarine de type A (*Margarine = A*), le deuxième de ceux qui ont consommé la margarine de type B (*Margarine = B*).

Ces représentations permettent de visualiser les différences entre les groupes. (Les figures 2 et 3 ont la même échelle). À priori, les patients ayant consommé de la margarine de type B ont vu leur taux de cholestérol plus baisser que ceux ayant consommé de la margarine de type A.

d) Analyse et conclusion

Tous les patients du jeu de données ont vu leur cholestérolémie baisser après 4 et 8 semaines de régime.

Deux tests ANOVA (ANalysis Of VAriance) ont été réalisés pour observer si la variation de cholestérolémie après 4 et 8 semaines change significativement en fonction du type de margarine consommé. Les deux tests montrent une différence entre les deux groupes à 4 et 8 semaines (*p-values* < 0.05, respectivement 0.0228 et 0.0469 à 4 et 8 semaines). La baisse de cholestérol semble être influencée par le type de margarine consommée.

La baisse moyenne observée est de -0.629 mmol/L pour une valeur moyenne initiale de 6.408 mmol/L, soit environ 11%, après 8 semaines. Pour comparaison, les statines, traitement de référence de l'hypercholestérolémie chez l'homme, doivent diminuer le cholestérol d'au moins 20% pour être considérées "d'intensité basse".^[ref. 3]

Il n'y a pas assez d'informations pour pouvoir affirmer que la baisse cholestérolémie observée est causée par la consommation de Margarine. Pour pouvoir tirer des conclusions plus robustes, il aurait fallu un groupe témoin suivant le même régime bas en graisse et bas en cholestérol sans consommer de margarine, et un nombre de patients plus important.

2- Classification avec Python

a) Exploration et nettoyage des données

Le jeu de données utilisé, `cardio_train.csv` comprend environ 70 000 lignes avec plusieurs variables :

- **Age (*age*)** : age des patients
- **Height (*height*)** : taille des patients
- **Weight (*weight*)** : poids des patients
- **Gender (*gender*)**: genre des patients
- **Systolic blood pressure (*ap_hi*)** : Pression artérielle systolique
- **Diastolic blood pressure (*ap_lo*)** : Pression artérielle diastolique
- **Cholesterol (*cholesterol*)** : Variable catégorielle indiquant la présence d'hypercholestérolémie
- **Glucose (*gluc*)** : Variable catégorielle indiquant la présence d'hyperglycémie
- **Smoking (*smoke*)** : Variable indiquant les patients fumeurs
- **Alcohol intake (*alco*)** : Variable indiquant si les patients consomment de l'alcool
- **Physical activity (*active*)** : Variable indiquant si les patients pratiquent une activité sportive
- **Presence or absence of cardiovascular disease (*cardio*)** : Variable "cible" indiquant si le patient est atteint ou non de maladies cardio-vasculaires.

Beaucoup de ces variables sont des facteurs connus de risque cardio-vasculaire. L'enjeu est d'essayer de créer un modèle capable de prédire si un patient est atteint de maladies cardio-vasculaires en utilisant les informations disponibles.

Toutes les valeurs dans le jeu de données sont renseignées (*vérification avec `DataFrame.isna().any()`*).

Après import des données dans pandas et exploration (avec `DataFrame.describe()` par exemple), des valeurs aberrantes ressortent.

Certaines valeurs de pressions artérielles systolique et diastolique sont négatives (ce qui est impossible). D'autres sont supérieures à 300. Il y a aussi des cas où la pression artérielle systolique est inférieure à la pression artérielle diastolique. Il a été choisi de retirer du dataset les lignes dans les cas suivants :

- Pression artérielle systolique (`ap_hi`) < 30 ou > 250
- Pression artérielle diastolique (`ap_lo`) < 10 ou > 250
- Pression artérielle systolique < Pression artérielle diastolique

1299 lignes incohérentes sont exclues de cette manière. Les limites supérieures et inférieures de pression artérielle ont été choisies en se basant sur les limites physiologiques, de manière conservatrice. Pour référence, une crise hypertensive, potentielle urgence vitale, est définie chez l'homme dès que la pression artérielle dépasse 180.

b) Feature engineering

La taille et le poids sont disponibles pour chaque patient. Il est possible de calculer l'IMC pour l'ensemble du dataset avec la formule :

$$\text{IMC} = \text{Poids en kg} / (\text{Taille en mètre})^2$$

Avec une `DataFrame` pandas il est possible d'utiliser la formule suivante :

```
data["BMI"] = data.apply(lambda x: x["weight"] / ((x["height"] / 100)**2),  
                        axis=1)
```

L'IMC prend en compte la taille et le poids d'une personne et permet d'estimer le niveau de risque d'obésité/maigreur. Un IMC inférieur à 16 indique une maigreur extrême et un IMC supérieur à 40 une obésité morbide.

Après le calcul, j'ai pris la décision de supprimer les lignes avec un IMC inférieur à 10 ou supérieur à 70 (39 au total).

Les variables catégorielles (`gluc`) et (`cholesterol`) sont aussi transformées en variables binaires avec la fonction `pd.get_dummies()`.

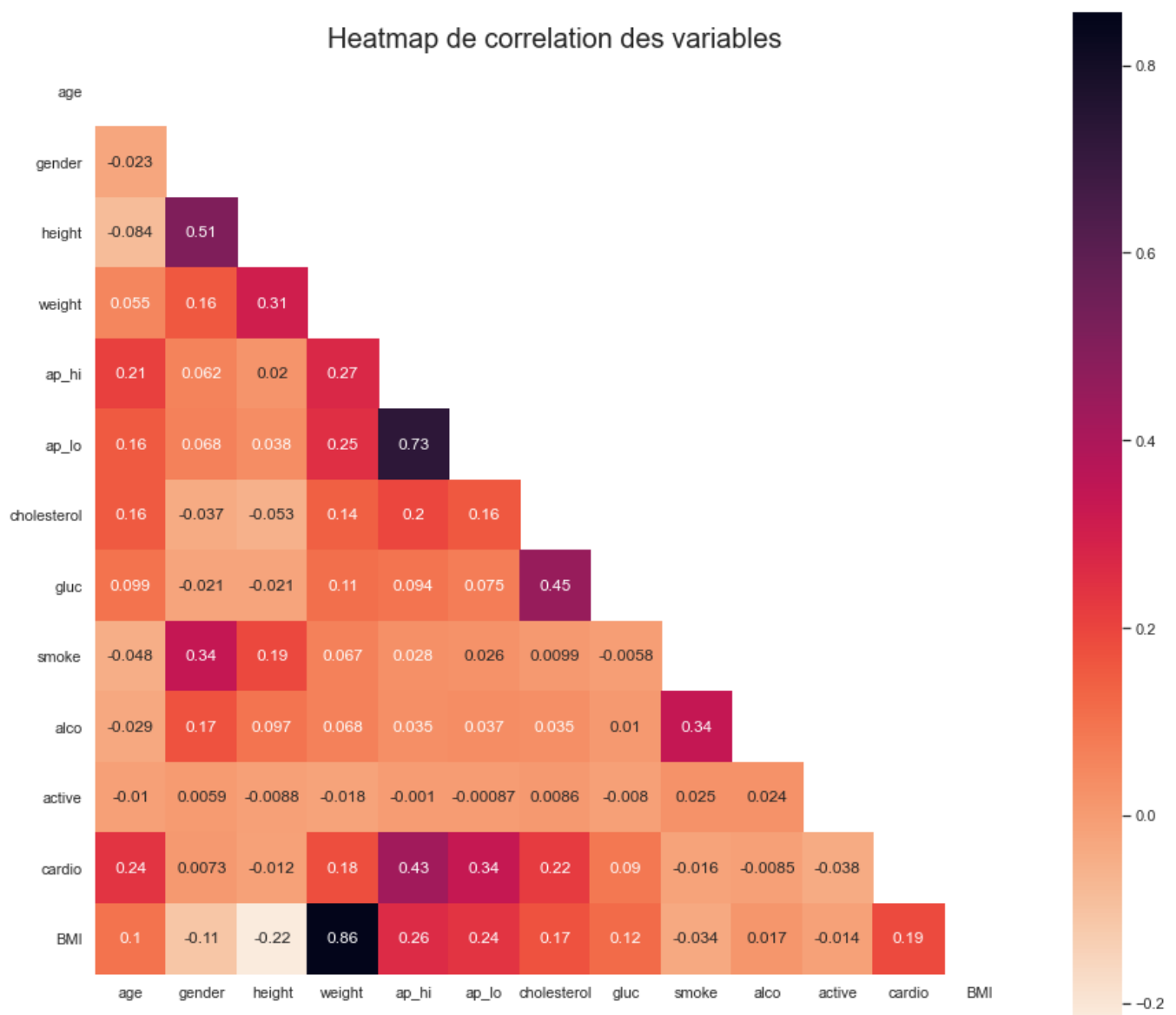


Fig. 4 : Heatmap des corrélations entre les variables

Pour explorer les données, il est intéressant de tracer une heatmap des corrélations entre les variables. La figure ci-dessus permet de mettre en évidence plusieurs relations intéressantes :

- La variable cible (*cardio*) est assez fortement corrélée aux pressions artérielles *ap_hi* et *ap_lo* qui sont elles-mêmes corrélées ensemble.
- La taille des patients (*height*) du dataset est corrélée au genre (*gender*)
- La glycémie (*gluc*) est corrélée à la cholestérolémie (*cholesterol*)
- La consommation d'alcool (*alcohol*) est corrélée avec le tabagisme (*smoke*)

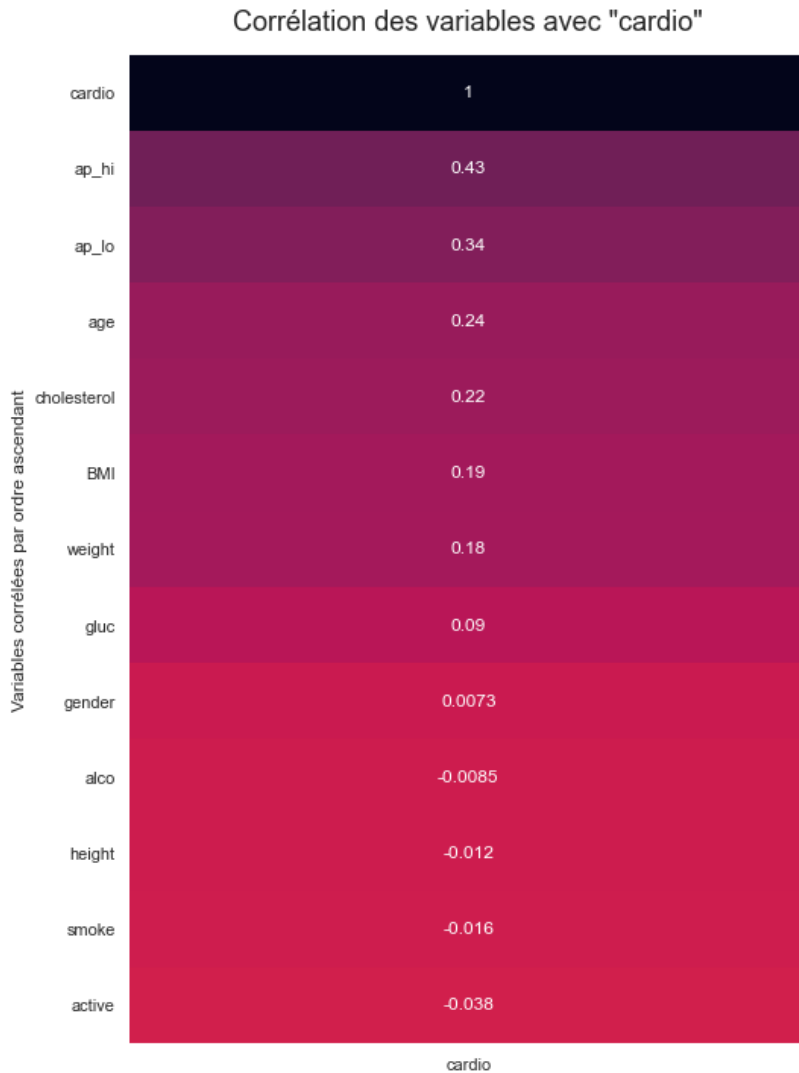


Fig. 5 : Heatmap des corrélations avec la variable cible (cardio)

La figure 5 montre la corrélation des variables avec cardio par ordre croissant. Il est intéressant de remarquer que les variables décrites comme subjectives dans le dataset (tabagisme *smoke*, consommation d'alcool *alco*, et activité physique *active*) sont assez peu corrélées avec la variable cible.

Les trois sont pourtant reconnues comme impactant le risque cardio vasculaire d'un individu. L'activité physique est un facteur protecteur, alors que la consommation d'alcool et le tabagisme sont des facteurs de risques.^[ref.4]

Deux hypothèses peuvent permettre d'expliquer cette différence :

- Il est possible que les patients reportent mal ces variables (biais de déclaration)
- Il est possible que la sélection des patients dans le dataset efface ces relations (biais de sélection)

c) Exploration des modèles

Après exploration et nettoyage du dataset, ce dernier a été séparé en 3 sous-groupes :

- **Train** : 70% du dataset (*48 063 lignes*) qui servira pour entraîner les modèles.
- **Validation** : 20% du dataset (*13 801 lignes*) qui permet de valider l'entraînement du modèle (et en particulier d'estimer l'overfitting)
- **Test** : 10% du dataset (*6 798 lignes*), permet de tester le modèle pour essayer d'estimer la capacité de généralisation du modèle.

J'ai pris la décision d'enlever la taille et le poids du jeu de données pour l'entraînement, sachant qu'il sont utilisés pour calculer l'IMC. Les variables restantes sont : l'âge, l'IMC, le genre, les pressions artérielles systoliques et diastoliques, le cholestérol, la glycémie, le tabagisme, la consommation d'alcool, et l'activité physique des patients.

Plusieurs algorithmes de classification de la librairie *scikit-learn* ont été testés :

- Random forests
- Support-vector machine
- Gradient boosting
- k-nearest neighbors
- Gaussian Naive Bayes
- Logistic regression

Chacun a été entraîné avec le même sous-ensemble *Train* du dataset. Ils ont ensuite été évalués en fonction de leur capacité à prédire la présence ou l'absence de la variable cible (*cardio*).

Un diagramme en barres a été créé pour comparer les résultats de classification obtenus :

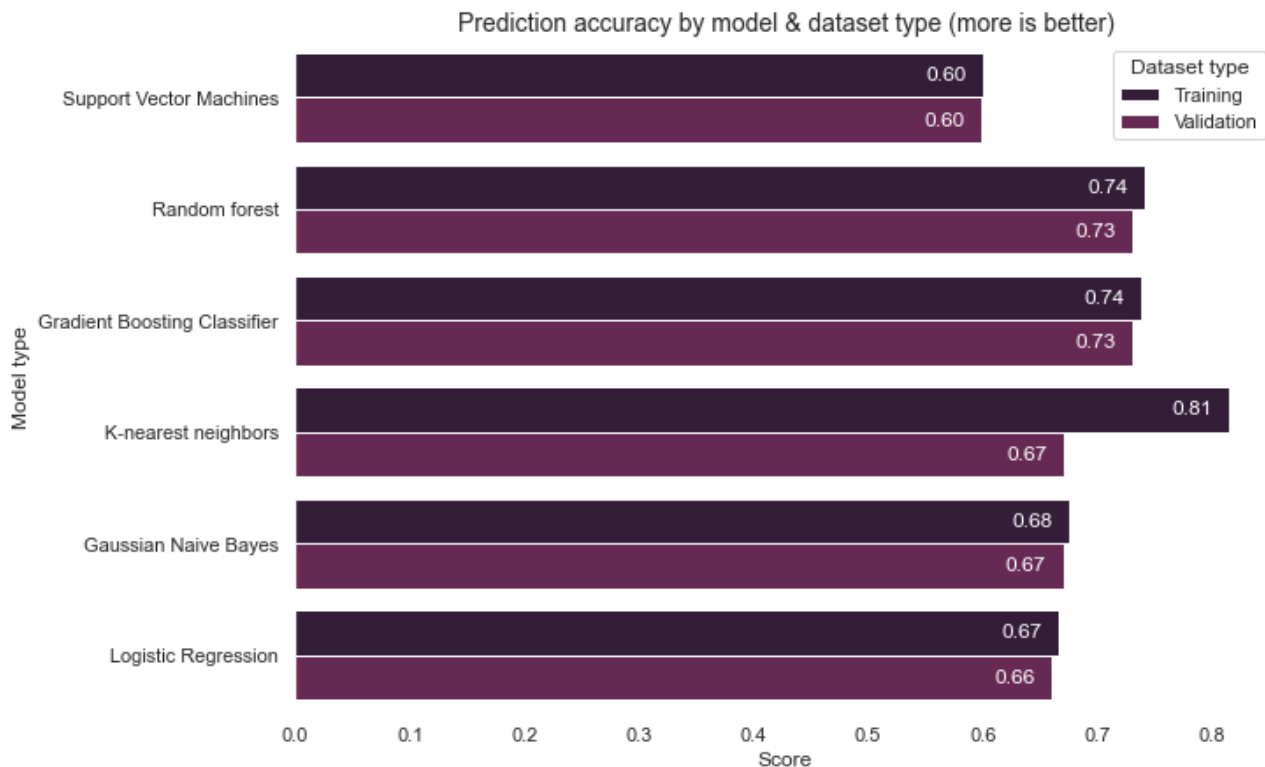


Fig. 6 : Performance pour chaque modèle en fonction du type de dataset.

Pour chaque modèle, la barre la plus foncée représente la précision de prédiction obtenue sur le dataset d'entraînement, et la barre la plus claire la précision sur le dataset de validation. Il est important de vérifier la différence entre les deux pour s'assurer, entre autres, que le modèle n'est pas "overfitté". Un score de 0.74 indique que dans 74% des cas, le modèle est capable de prédire correctement la présence de la variable cible.

Le modèle k-nearest neighbors a une bonne précision sur le dataset d'entraînement mais est beaucoup moins précis sur le jeu de données de validation.

Les deux modèles les plus intéressants semblent être le Random Forest et le Gradient Boosting classifier. Leurs précisions d'entraînement et de validation sont proches (respectivement 0.74 et 0.73 pour les deux). Les deux sont des algorithmes d'apprentissage supervisé et sont qualifiés comme "ensemble-based".

J'ai choisi d'utiliser le modèle de Random Forest car je le connaissais mieux que le Gradient Boosting classifier.

d) Amélioration du modèle par Grid Search

Une fois l'algorithme de classification sélectionné, il est possible d'optimiser les hyper-paramètres du modèle avec la fonction `GridSearchCV()` de *scikit-learn*. `GridSearchCV()` prend en argument un dictionnaire de paramètres pour un modèle de classification et teste toutes les combinaisons possibles pour trouver la plus optimale.

Cette approche est simple mais demande beaucoup de temps de calcul si la taille du dictionnaire est importante. J'ai choisi d'utiliser le dictionnaire de recherche suivant :

```
param_1 = {  
    "criterion": ["gini", "entropy"],  
    "max_depth": [ 5, 7, 9, 12, 15],  
    "min_samples_split": [2, 5, 10, 15, 100],  
    "min_samples_leaf": [2, 5, 7, 9, 12, 15, 20, 500],  
    "max_features": ['auto', 'sqrt', 0.5, 0.4, 0.3] }
```

Voici les paramètres optimisés obtenus par la recherche :

```
Best_params_ = {'criterion': 'entropy',  
                'max_depth': 12,  
                'max_features': 'auto',  
                'min_samples_leaf': 5,  
                'min_samples_split': 2}
```

Il est important de préciser que `GridSearchCV()` réalise une validation croisée en blocs pour chaque set de paramètres testés, pour éviter l'overfitting et obtenir des mesures de performance plus représentatives.

e) Validation du modèle

Après avoir optimisé les paramètres, il faut valider le modèle. *scikit-learn* inclut des fonctions de validation croisée en blocs. Cette dernière consiste à entraîner le modèle plusieurs fois sur des sous-ensembles du jeu de données, en utilisant à chaque fois un sous-ensemble exclu de l'entraînement pour la validation

Les scores obtenus par le modèle optimisé lors de la cross-validation en 10 blocs sont les suivants :

Score moyen : 0.73664

Ecart-type : 0.00617

Un rapport de classification sur le dataset de test a été réalisé :

	precision	recall	f1-score	support
0	0.71	0.79	0.75	3439
1	0.76	0.68	0.72	3359
accuracy			0.73	6798
macro avg	0.74	0.73	0.73	6798
weighted avg	0.74	0.73	0.73	6798

Fig. 7 : Rapport de classification du modèle

Les lignes 0 et 1 de ce rapport donnent les informations sur le modèle pour les patients sans problèmes cardio-vasculaires (*cardio* = 0) et les patients qui en ont (*cardio* = 1). Le modèle prédit moins bien les cas positifs que les cas négatifs. Le recall pour les patients *cardio* 1 est de 0.68 contre 0.79 pour les autres.

Pour aller plus loin, il aurait été intéressant d'essayer de rechercher les modèles avec le moins de faux négatifs possible (maximiser le recall pour la classe *cardio* = 1) tout en conservant une précision correcte.

Cette approche aurait permis de faire un test plus utile pour le dépistage des patients en permettant de détecter les problèmes cardio-vasculaires avant qu'ils ne se produisent.

f) Création d'un modèle réduit

Le modèle actuel utilise toutes les variables disponibles dans le jeu de données, à l'exception de la taille et du poids qui sont combinées dans l'IMC calculé précédemment.

En utilisant les mesures d'importance des variables pour le modèle, il est possible de choisir celles qui sont les plus utiles pour les prédictions :

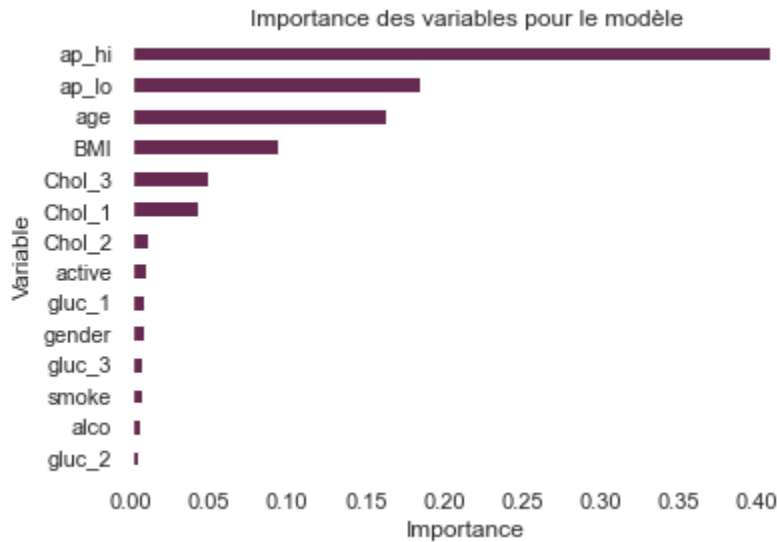


Fig. 8 : Diagramme en barres de l'importance des variables pour le modèle.

Les pressions artérielles, l'âge, l'IMC et, dans une moindre mesure, la cholestérolémie sont les variables les plus importantes pour le modèle de Random Forest.

Un nouveau modèle de Random Forest a été entraîné en utilisant moins de variables. Il n'a besoin que des pressions artérielles, de l'âge et de l'IMC du patient. Le taux de cholestérol (*cholesterol*) n'a pas été inclus dans le jeu de données utilisé : pour l'obtenir il faut obligatoirement une prise de sang, contrairement à toutes les autres variables.

L'intérêt est d'essayer d'obtenir un modèle qui a besoin de moins d'informations complexes à renseigner, permettant une utilisation plus facile (*pas besoin de prise de sang*).

Après cross-validation en 10 blocs, le modèle réduit obtient une précision moyenne de 0.72353 et un écart-type de 0.00469. Ses performances sont un peu moins bonnes que le modèle précédent, mais restent correctes. Il peut être intéressant dans certains cas d'utilisation (*diagnostic rapide à destination du public, application de santé, campagne de prévention ?*).

3- Topic modeling avec Python

a) Exploration et préparation des données

Le dataset suivant, *covid19_tweets.csv* est une collection de 179 108 tweets recueillis à partir de juillet 2020 avec le hashtag #covid19. L'objectif est d'utiliser du topic modeling pour obtenir des informations intéressantes.

Le dataset contient les colonnes suivantes :

- **user_name** : Username de l'auteur du tweet
- **user_location** : Location de l'auteur du tweet (*renseignée manuellement par l'utilisateur*)
- **user_description** : Description du profil de l'auteur du tweet
- **user_created** : Date de création du compte de l'auteur du tweet
- **user_followers** : Nombre de followers de l'auteur du tweet
- **user_friends** : Nombre d'amis twitter de l'auteur du tweet
- **user_favourites** : Nombre de tweets favoris de l'auteur du tweet
- **user_verified** : True si l'auteur du tweet est vérifié
- **date** : Date de publication du tweet
- **text** : Texte du tweet
- **hashtags** : Hashtags contenus dans le tweet
- **source** : Plateforme utilisée pour envoyer le tweet
- **is_retweet** : True si la ligne est un retweet (copie du tweet d'un autre utilisateur republiée)

En regardant rapidement les données, il semble que la colonne *is_retweet* est toujours *False*, indiquant que les retweets ont été exclus de la collection ou qu'ils ont été supprimés du csv.

Cette observation est confirmable en utilisant la fonction `DataFrame.is_retweet.unique()`, qui, alors qu'elle retourne toutes les valeurs présentes dans la colonne *is_retweet*, retourne uniquement *False*.

La colonne *text* contient le texte de tous les tweets. Vu qu'il n'y a pas de retweets, ils devraient tous être différents. Il est possible de vérifier en comparant le nombre d'éléments de `DataFrame.value_counts.text` et le comparant avec le nombre d'éléments total restant de la `DataFrame`. 425 tweets sont dupliqués dans le jeu de données. Ils peuvent être supprimés avec la fonction `DataFrame.drop_duplicates('text', inplace=True)`.

La colonne *user_location* paraît à première vue être intéressante pour lier des tendances à des emplacements géographiques. Son intérêt est cependant limité par le fait qu'elle est remplie manuellement par l'utilisateur sur son profil (*au lieu de correspondre à une géolocalisation*). La colonne est donc remplie de valeurs qui sont, soit inutilisables car fictives/humoristiques, soit invérifiables.

b) Pré-processing des tweets

Avant de pouvoir faire du topic modeling, il faut préparer le texte qui va être utilisé. Les tweets contiennent des hashtags, des caractères spéciaux, des numéros et des émoticônes qui seront gênants pour constituer un corpus.

L'algorithme de pré-processing créé est le suivant :

1. Remplacer les emojis des tweets par leur noms en chaîne de caractère : utilisation de la fonction `demojize()` du module *emoji*.² Les emojis ont du sens. Les supprimer peut faire perdre une partie utile du texte.
Ex: `demojize("I ate 🍕") = "I ate pizza"`.
2. Supprimer tous les caractères unicodes spéciaux avec `unidecode()` du module *unidecode*.³
Ex: `unidecode("This is a Tweet") = "This is a Tweet"`
3. Passe tout le texte en minuscule avec `String.lower()`
Ex: `"INSERT TWEET HERE".lower() = "insert tweet here"`
4. Utilise des motifs regex avec la librairie standard *re* pour :
 - 4.1. Supprimer les urls : `"https?:\/\/\S+|www\.\S+"`
 - 4.2. Supprimer tous les nombres : `"\d+"`
 - 4.3. Supprimer tous les caractères spéciaux : `"[\^\\w\\s]"`
 - 4.4. Remplacer les espaces répétés par un seul : `"\s\s+"`*Ex: `"2 0 2 0 is awful!!!!" = " is awful"`*

La fonction peut être appliquée à l'ensemble des 178 683 tweets du jeu de données avec `map` :

```
df["c_text"] = DataFrame.text.progress_map(preprocessor)
```

Pour les opérations potentiellement longues (*environ une minute trente ici*), il est intéressant d'utiliser la fonction `progress_map()` du module *tqdm*⁴ au lieu de `map()` pour avoir une barre de progression lors de l'exécution.

² <https://pypi.org/project/emoji/>

³ <https://pypi.org/project/Unidecode/>

⁴ <https://pypi.org/project/tqdm/>

c) Vérification du langage des tweets

Avant de continuer, j'ai choisi de vérifier le langage des tweets du dataset avec la fonction `detect()` du module *langdetect*.⁵

La fonction peut-être exécutée de la manière suivante :

```
df["lang"] = df.c_text.progress_map(detect)
```

Un total de 2461 tweets ont été détectés comme non-anglais sur le jeu de données. Ils ont été exportés dans un fichier CSV pour vérification. La plupart contiennent des mots incompréhensibles et semblent présenter peu d'intérêt. Ils ont été exclus du dataset pour les opérations suivantes. Ci-dessous est la liste des 10 langues les plus détectés avec leur nombre de tweets respectifs :

Langue	Nombre de tweets
En (English)	176222
It (Italian)	526
Es (Spanish)	324
Id (Indonesian)	244
Ca (Catalan)	209
Ro (Romanian)	175
Fr (French)	175
Et (Estonian)	124
Nl (Dutch)	115
Da (Danish)	100

Fig. 9 : Nombre de tweets des 10 langues les plus détectés dans le dataset

Cette étape de vérification du langage n'était pas obligatoire, mais elle permet de mieux vérifier la qualité des données.

⁵ <https://pypi.org/project/langdetect/>

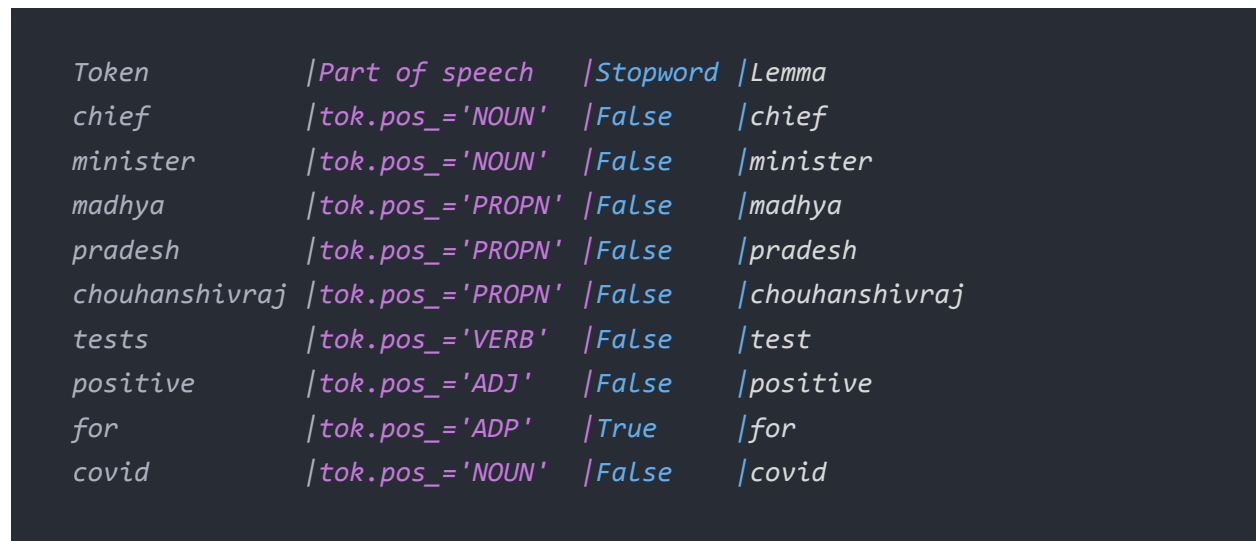
d) Tokenisation, Lemmatisation & part of speech tagging

L' étape suivante est de séparer le texte des tweets en tokens avant de regrouper les mots identiques. Pour cela, il a été choisi d'utiliser spaCy.⁶ spaCy est une librairie open source de Natural Language Processing (NLP).

La librairie permet d'utiliser des modèles pré-entraînés capables de :

- Tokeniser des phrases en ignorant la ponctuation
- Assigner des *"part-of-speech tags"* à chaque mot. Le modèle sait séparer les verbes, les noms, les adjectifs, les noms propres...
- Détecter les entités nommées.

Le module complémentaire *Lemminflect*⁷ permet d'utiliser le *"part-of-speech tag"* de chaque mot pour lemmatiser de manière plus intelligente le texte.



Token	Part of speech	Stopword	Lemma
chief	tok.pos_='NOUN'	False	chief
minister	tok.pos_='NOUN'	False	minister
madhya	tok.pos_='PROPN'	False	madhya
pradesh	tok.pos_='PROPN'	False	pradesh
chouhanshivraj	tok.pos_='PROPN'	False	chouhanshivraj
tests	tok.pos_='VERB'	False	test
positive	tok.pos_='ADJ'	False	positive
for	tok.pos_='ADP'	True	for
covid	tok.pos_='NOUN'	False	covid

Fig. 10 : Exemple des résultats obtenus avec spaCy sur un tweet du dataset

Le modèle est capable de reconnaître les noms propres sans majuscules (elles ont été enlevées lors du preprocessing) et n'essaye pas de les lemmatiser !

⁶ <https://spacy.io/>

⁷ <https://pypi.org/project/lemminflect/>

e) Sélection des tokens et création de dictionnaires

Après avoir utilisé spaCy pour générer des tokens, il faut procéder à une sélection des mots d'intérêt. Cette dernière a été réalisée de la manière suivante :

1. Ne conserver que les mots (*exclut les espaces*).
2. Exclure les mots dans la stop list de spacy (*mots à exclure car non utiles, par exemple, "is"*)
3. Ne garder que les mots d'au moins 3 caractères.

Les tokens sélectionnés sont ensuite répartis dans trois dictionnaires différents en fonction de leur *"part-of-speech tag"* (*chaque token peut être dans plusieurs dictionnaires*) :

- Le premier accepte les noms communs, les verbes, adjectifs et les noms propres.
- Le deuxième accepte les noms communs, verbes et adjectifs.
- Le troisième n'accepte que les noms propres.

Les dictionnaires sont créés avec `gensim.corpora.Dictionary()` du module *gensim*⁸

Les valeurs extrêmes sont enlevées des dictionnaires avec la fonction :

```
Dictionary.filter_extremes(no_below=X, no_above=Y, keep_n=Z)
```

Les mots qui apparaissent dans moins de X documents ou dans plus de Y documents sont filtrés, et seuls les Z plus fréquents mots sont conservés.

Des bag of words (*BOW*) sont créés avec la fonction `.doc2bow()` pour chaque dictionnaire.

Les BOWs sont convertis en matrices TF-IDF avec :

```
gensim.models.tfidfmodel.TfidfModel()
```

TF-IDF (*Term Frequency-Inverse Document Frequency*) permet d'évaluer l'importance des mots en fonction de leur nombre d'occurrences et de leur fréquence. Par rapport à un bag of word, TF-IDF prend en compte la rareté des termes dans le corpus entier.

⁸ <https://radimrehurek.com/gensim/index.html>

f) Création d'un modèle LDA

Les modèles LDA (Latent Dirichlet Allocation) sont une forme de modèles d'apprentissage non supervisés. En partant d'un nombre de sujet fixé k , ils assignent un sujet à chaque mot d'un document en prenant en compte les sujets des autres mots et les sujets assignés précédemment au mot en cours d'évaluation.

Les modèles sont créés avec la fonction : `gensim.models.LdaMulticore()`

Il est possible d'évaluer la performance des modèles LDA en mesurant leur cohérence avec :
`gensim.models.CoherenceModel()`

C_v ⁹ a été choisie comme mesure de cohérence. Un modèle est toujours évalué en fonction d'un corpus de documents. Il est important de noter qu'une cohérence élevée ne garantit pas forcément la qualité d'un modèle. Un modèle avec un score élevé peut fournir une classification moins pertinente qu'un autre avec un score plus bas.

L'évaluation de la qualité d'un modèle LDA est complexe. Pour ce travail, un des objectifs est de réaliser des visualisations intéressantes.

Le premier modèle entraîné sur le corpus avec les noms communs, verbes, adjectifs et noms propres obtient un score de 0.27408 avec 6 sujets.

Il est possible d'optimiser les hyperparamètres du modèle pour maximiser la cohérence de la même manière que pour le random forest de la partie 2.

L'algorithme de l'article *Evaluate Topic Models: Latent Dirichlet Allocation*^[ref. 5] a été adapté pour ce travail. Son fonctionnement est similaire à `GridSearchCV()`. Toutes les combinaisons de paramètres spécifiées sont testées et évaluées de manière itérative pour maximiser la cohérence du modèle.

Après optimisation, le modèle précédent obtient un score de 0.30360 avec 13 sujets.

⁹ <https://radimrehurek.com/gensim/models/coherencemodel.html>

g) Visualisations du topic modeling

Il est possible de générer des nuages de mots avec le module *wordcloud*.¹⁰



Fig. 11 : Nuages de mots générés avec les mots des 4 premiers topics du modèle optimisé

Ci-dessus est un exemple de wordcloud généré avec les mots les plus fréquents des 4 premiers sujets du modèle LDA optimisé précédemment. Des thèmes presque cohérents semblent ressortir.

¹⁰ <https://pypi.org/project/wordcloud/>

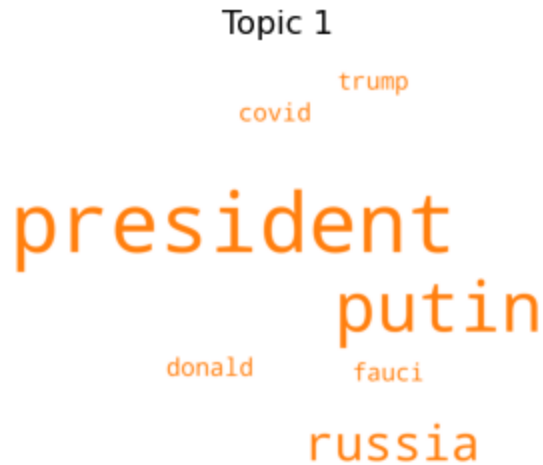


Fig. 12 : Wordcloud généré avec un des sujets du modèle entraîné uniquement avec les noms propres

Le modèle entraîné sur les noms propres ne donne pas des résultats aussi intéressants que le précédent. Ses paramètres optimaux sont probablement complètement différents de ceux du modèle avec les autres mots. Avec 13 sujets et sans changer les réglages, le modèle obtient pourtant un score de cohérence de 0.53327.

Le modèle entraîné avec les noms communs, verbes et adjectifs en excluant les noms propres donne des résultats assez similaires au premier. Il obtient une cohérence légèrement supérieure de 0.31069.

h) Analyse de sentiments

VADER-Sentiment-Analysis¹¹ permet de réaliser une analyse de sentiment sur le texte des tweets. Elle ne correspond pas exactement à du topic modeling mais peut donner des informations intéressantes. Après initialisation de l'analyseur, il peut être appelé avec tous les tweets du dataset :

```
df["sentiment"] = df.c_text.progress_map(analyser.polarity_scores)
```

L' algorithme est capable d'utiliser les majuscules, la ponctuation et les emojis. Seuls ces derniers ont été conservés lors du prétraitement du texte. Pour obtenir des résultats plus précis, il aurait fallu procéder à un nettoyage particulier du dataset pour éviter la perte d'information.

Le score de sentiment composite moyen obtenu par le dataset est de 0.06587 avec une déviation standard de 0.45548, -1 étant le score le plus négatif et 1 le plus positif.

[illegible]

Fig. 13 : Tweets avec les scores de sentiments les plus extrêmes. La sélection aurait été différente si les majuscules et la ponctuation avaient été prises en compte.

¹¹ <https://pypi.org/project/vader-sentiment/>

Références

1. Kannel WB. Cholesterol and risk of coronary heart disease and mortality in men. *Clin Chem*. 1988;34(8B):B53-9. PMID: 3042200.
2. Farzadfar F, Finucane MM, Danaei G, Pelizzari PM, Cowan MJ, Paciorek CJ, Singh GM, Lin JK, Stevens GA, Riley LM, Ezzati M; Global Burden of Metabolic Risk Factors of Chronic Diseases Collaborating Group (Cholesterol). National, regional, and global trends in serum total cholesterol since 1980: systematic analysis of health examination surveys and epidemiological studies with 321 country-years and 3·0 million participants. *Lancet*. 2011 Feb 12;377(9765):578-86. doi: 10.1016/S0140-6736(10)62038-7. Epub 2011 Feb 3. PMID: 21295847.
3. <https://pharmacomedicale.org/medicaments/par-specialites/item/statines>
4. https://www.cdc.gov/heartdisease/risk_factors.htm
5. <https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0>