Antoine LE BOULCH
Tom LELIEVRE
Nathan SIMON

ISEN ALL IS DIGITAL! | yncréa

Rapport projet IA A3

Professeures responsables :
Nesma Settouti
Abdallah Saab Nadine
IA

Sommaire

Librairie utilisé :

Scikit-learn
Numpy
Panda
Pickle
Json
folium
matplotlib
XGBoost
seaborn
Plotly

Sources :
Photo :
https://solutions.lesechos.fr/tech/c/ia-et-gestion-des-risques-garder-lhumain-au-coeur-des-processus-42449/

Clustering

Préparation des données

Pour la préparation des données on réalise un import du fichier csv obtenu à la fin du projet de Big Data du projet de la semaine précédente.On choisit de garder ce fichier plutôt que d'utiliser fourni pour cette semaine car il contient plus de données. Pour pouvoir répondre aux besoins du client on sélectionne les colonnes haut_tot, X et Y pour l'affichage sur la carte et le clustering et les colonnes tronc_diam, age_estim et haut_tronc pour la détection d'anomalies.

Clustering

Pour l'apprentissage des données on a testé plusieurs méthodes de clustering que sont, KMeans, AgglomerativeClustering, SpecttralClustering et GfaussianMixture.

KMeans fonctionne de la manière suivante, on initialise un certain nombre de clusters de manière aléatoire puis on leur attribue les points les plus proches d'eux, le centre du cluster prend la position moyenne des points qui lui sont attribués on répète jusqu'à convergence.

AgglomerativeClustering considère d'abord chaque point comme un cluster distinct puis on calcule la distance entre chaque cluster et on fusionne les plus proches puis la matrice est mise à jour. On répète cette opération jusqu'à atteindre le nombre de clusters voulu.

SpectralClustering utilise une matrice de similarité qui représentent les relations entre les points. Les clusters sont déterminés en utilisant KMeans sur un espace créé a partir des vecteurs propres du Laplacien de cette matrice.

DBSCAN utilise la densité de points autour de chaque point pour définir si il s'agit de bruit, d'un point de bordure de cluster ou d'un point coeur du cluster en fonction d'une distance max entre deux point pour qu'il y ait liaison et d'un nombre minimum pour former un point coeur.

GMM trouve une approximation des données en utilisant une combinaison de distributions gaussiennes et attribue chaque point à une gaussienne de manière probabiliste, chaque gaussienne représente un cluster.

Métriques

Puisque nous avons effectué un apprentissage non supervisé il nous faut des manières d'évaluer nos modèles qui ne nécessitent pas de Y réels alors on a utilisé le Silhouette score, le score de Davies-Bouldin et le score de Calinski-Harabasz.

Pour 2 clusters

model	KMeans	Agglomerative	Spectral	GMM
silhouette	0.664	0.647	0.661	0.604
Davies-Bouldin	0.493	0.489	0.491	0.546
Calinski-Harabasz	27578.817	24068.899	26765.863	20546.308

Pour 3 clusters

model	KMeans	Agglomerative	Spectral	GMM
silhouette	0.643	0.629	0.637	0.641
Davies-Bouldin	0.501	0.454	0.434	0.487
Calinski-Harabasz	32721.596	23134.145	15635.039	30911.422

Dans toutes mes expérimentations quelques soient les paramètres utilisés pour les algorithmes de clustering c'est K Means qui est le plus performant

Le silhouette score prend en compte la distance moyenne des points du clusters entre eux et la distance moyenne des points aux points du cluster le plus proche qui n'est pas le sien plus le score est proche de 1, plus les points sont assignés au bon cluster si il est proche de 0 les points sont à la frontière entre deux clusters enfin si le score est négatif les points sont attribués aux mauvais clusters.

Le score Davies-Bouldin mesure la moyenne des rapports entre la distance intra-cluster et la distance inter-cluster pour chaque cluster, un score bas indique que les clusters sont compacts et bien séparés.

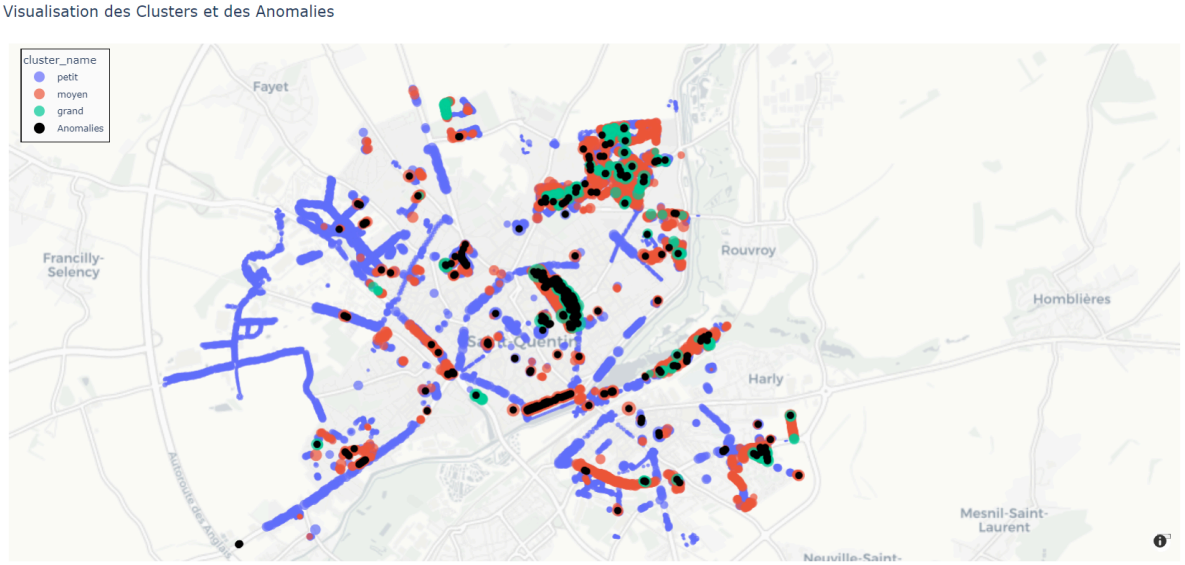
Le score de Calinski-Harabasz évalue la dispersion inter-cluster par rapport à la dispersion intra-cluster. Un score élevé indique que les clusters sont bien séparés les uns des autres et que les points à l'intérieur des clusters sont compacts

Détection des anomalies

Pour la détection des anomalies on utilise IsolationTree sur les colonnes tronc_diam age_estim et haut_tronc, on obtient de cette manière 549 anomalies.

IsolationTree fonctionne avec des arbres de décisions appelés arbre d'isolation, ces arbres contiennent des sous-ensembles aléatoire de notre jeu de donnés. Ces arbres partitionnent les données de manières récursives jusqu'à ce que chaque point soit isolé. Les anomalies sont les points qui ont les chemins d'isolations les plus courts

Visualisation sur une carte

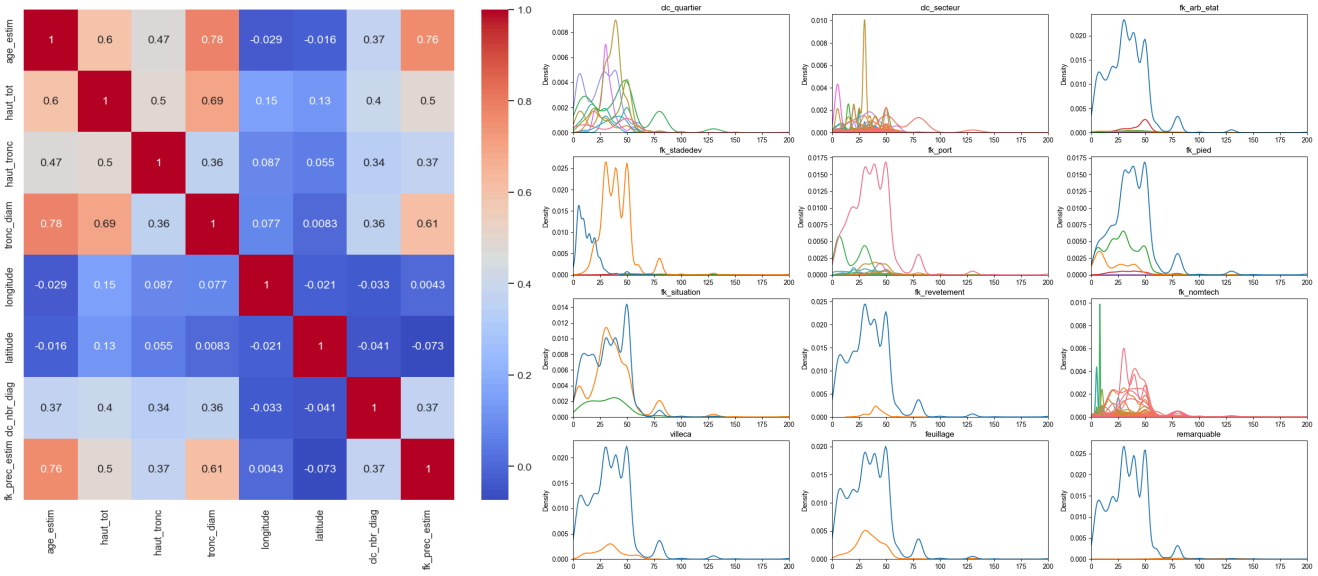


Sur une carte créée grâce à la librairie Plotly.graphe on visualise nos clusters avec une couleur différente pour chacun ensuite on affiche les anomalies en ajoutant des points noirs en surimpression sur les arbres qui ont été détectés comme étant des anomalies. Dans les besoins client il est spécifié qu'il faut afficher seulement deux ou trois catégories qui sont petits et grands ou petits, moyens et grands donc si l'utilisateur rentre un nombre supérieur de clusters à générer les clusters sont regroupés dans ces trois catégories grâce à la moyenne des hauteurs totales des arbres au sein des clusters.

Prédiction de l’âge

Les besoins du client numéro 2 était de prédire les âges estimés de certains arbres à partir de notre dataset en utilisant différents modèles (chaque modèle à été grid search pour trouver les best_param). Le but final étant de réaliser un script python permettant de prédire l’âge et de la renvoyer au format JSON à partir de données de test en format .JSON en utilisant le meilleur modèle.

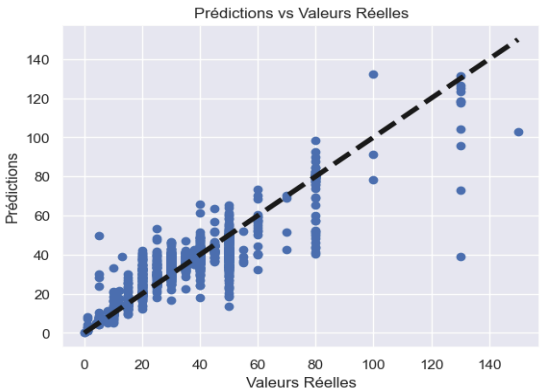
Pour tout les modèle voici les features sélectionnées
`'haut_tronc','tronc_diam','fk_stadedev','clc_nbr_diag','fk_nomtech','haut_tot'`



Ces features ont été déterminés par le biais de matrices de corrélation et de pearson et des histogrammes de densité. (fk_prec_estim et remarquable étant ignorés car eux même sont déterminés grâce à l’âge).
On va également appliquer un encoder (Ordinal, Label) au feature tel que fk_nomtech et fk_stadedev, en plus d’un scaler sur toute les features pour standardiser (evite le probleme des outlier, améliore la convergence)

Random Forest

Un ensemble de d'arbres de décision. Chaque arbre est construit à partir d'un échantillon aléatoire des données d'entraînement (bootstrapping). A chaque division (ou nœud), un sous-ensemble aléatoire des caractéristiques est choisi. Cela assure que les arbres sont diversifiés et réduit la corrélation entre eux. Pour faire une prédiction, Random Forest agrège les prédictions de tous les arbres. Pour la régression, la moyenne des prédictions des arbres est utilisée. Évalue l'importance des différentes caractéristiques dans la prédiction. Cela se fait en observant combien la précision de la prédiction diminue lorsque les données pour une caractéristique sont aléatoirement mélangées. Plus la baisse est importante, plus la caractéristique est considérée comme importante.

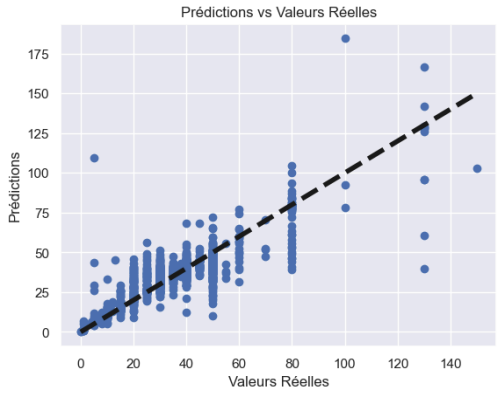


Résultat avec scikit

Mean Absolute Error (MAE): 0.24500230701503742
Root Mean Squared Error (RMSE): 0.42593542645308874

R-squared (R^2): 0.8182772441708803
Cross Validation Score: 0.7424395916910986
OOB_score: 0.8220250708338028

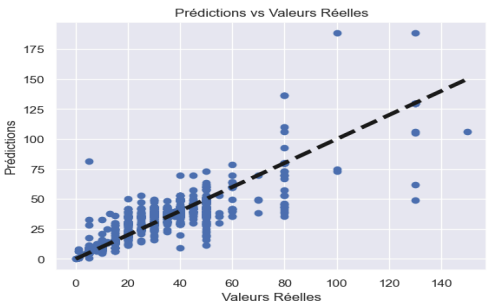
Résultat from Scratch :



R^2: 0.781358967348436
MSE: 0.21827795793866195
OOB Score: 0.8222008225196206

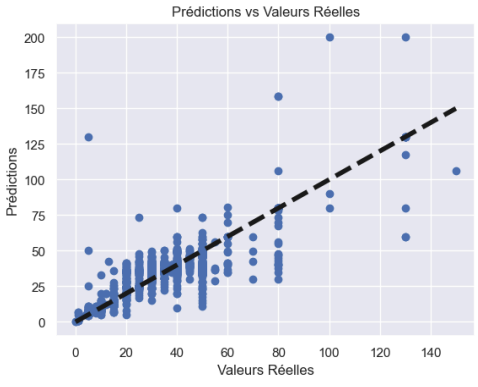
Classification and regression tree

L'algorithme commence avec l'ensemble de données entier et cherche la meilleure caractéristique et le meilleur point de division (seuil) pour diviser les données en sous-ensembles. Le "meilleur" est défini comme la division qui maximise la pureté des nœuds enfants ou minimise une fonction de coût (comme l'erreur quadratique moyenne pour la régression).Cette étape est répétée de manière récursive pour chaque sous-ensemble résultant jusqu'à ce qu'un critère d'arrêt soit atteint. Les critères d'arrêt peuvent inclure un nombre minimum d'observations dans un nœud, une profondeur maximale de l'arbre. Une fois le critère d'arrêt atteint, l'algorithme crée un nœud feuille. Pour la régression, le nœud feuille représente la moyenne des valeurs cibles des observations.L'élagage consiste à retirer des parties de l'arbre (en transformant certains nœuds internes en nœuds feuilles) pour améliorer la capacité de généralisation du modèle sur des données non vues. Elle réduit la complexité de l'arbre et évite le surajustement. Pour la régression, l'algorithme parcourt l'arbre jusqu'à un nœud feuille, et la prédiction est la valeur moyenne des observations dans ce nœud.
Résultat from Scikit learn



Mean Absolute Error (MAE): 0.30149990787479575
Root Mean Squared Error (RMSE): 0.5084996857392401
R-squared (R^2): 0.7409979715286941
Cross Validation Score: 0.6556251672631337

Résultat From Scratch

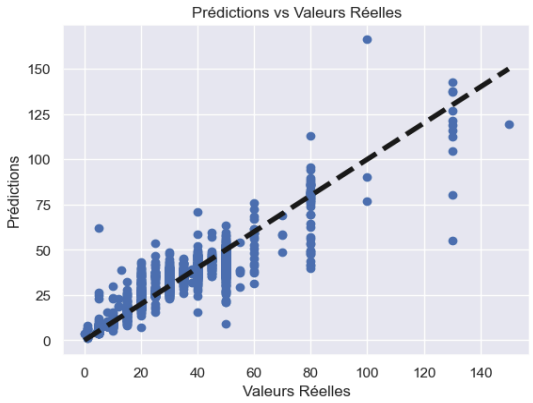


R^2: 0.6986828945215631
MSE: 0.30081673910055573
RMSE: 0.5484676281245373
mae: 0.2949840830214086

Algorithme de boosting (XGBoost, Gradient Boosting)

Le modèle est initialisé avec une prédiction constante qui peut être la moyenne des cibles pour la régression. Pour chaque instance dans le jeu de données, le résidu (l'erreur de prédiction) est calculé comme la différence entre la valeur cible réelle et la prédiction actuelle du modèle. Un arbre de décision est ajusté sur ces résidus, c'est-à-dire qu'il apprend à prédire le résidu pour chaque instance. Un taux d'apprentissage (learning rate) est appliqué pour réduire l'impact de chaque arbre et rendre le modèle plus robuste. La prédiction de l'arbre sur les résidus est ajoutée à la prédiction actuelle du modèle pour chaque instance, après avoir été multipliée par le taux d'apprentissage. Cela met à jour le modèle avec les nouvelles informations.Pour faire une prédiction, les contributions de tous les arbres sont ajoutées aux prédictions initiales.

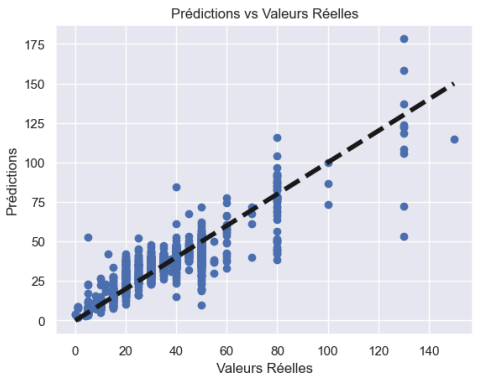
Gradient Boosting :



Mean Absolute Error (MAE): 0.2704448495050102
Root Mean Squared Error (RMSE): 0.4324043833542429
R-squared (R^2): 0.8127154450259364
Cross Validation Score: 0.7255052298199345

XGBoost a été conçue pour être très efficace et rapide. Il utilise une structure de données optimisée et des techniques de calcul pour accélérer l'apprentissage, ce qui le rend significativement plus rapide que l'implémentation standard du Gradient Boosting sur de grands ensembles de données.
XGBoost peut gérer automatiquement les valeurs manquantes, ce qui n'est pas toujours le cas dans les implémentations du Gradient Boosting.
Contrairement au Gradient Boosting traditionnel qui arrête de construire un arbre lorsque toutes les feuilles sont pures ou lorsqu'il atteint une profondeur maximale prédéfinie, XGBoost utilise le "pruning" (élagage) d'arbre, où la construction de l'arbre continue jusqu'à une profondeur maximale spécifiée, puis les branches avec le gain le plus faible sont élaguées.

XGBoost :



Mean Absolute Error (MAE): 0.28883691011283663
Root Mean Squared Error (RMSE): 0.446949637808049
R-squared (R^2): 0.0.7999037419533639
Cross Validation Score: 0.739092603410952

Comparaison classement :

Les graphiques montre une comparaison entre les valeurs prédites et les valeurs réelles on remarque qu'en général pour tous les modèles ont plus de mal à déterminer l'âges des vieux arbres (cela est due principalement durant la phase d'entraînement il y a plus d'arbres jeunes qu'il y a d'arbre vieux).
Pour CART la version scikit-learn présente de meilleur résultat que la version from Scratch de même pour le random Forest comme le démontre les metrics et le graphique.

On remarque également que RandomForest est le meilleur modèle parmis tout ceux tester, il est suivis de près par Gradient Boosting et XGBoost. Cela est expliqué par plusieurs facteurs :

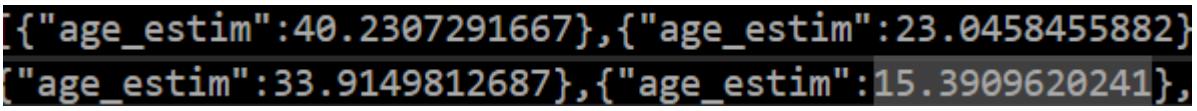
- Random Forest est plus robuste au bruit car les arbres individuels peuvent être influencés différemment par les anomalies (outliers, bruit), et leur effet est atténué lorsqu'on prend la moyenne des prédictions. Contrairement à GB où due à son fonctionnement séquentielle le bruit peut être de plus en plus accentué si les hyperparamètre sont mal réglé -> prédiction moins stable
- En général RandomForest est moins sensible au calibrage des hyperparamètre contrairement à GB (comme le learning rate par exemple)

C'est donc pour cela que le RandomForest est le modèle que nous avions décidé de prendre pour réaliser le script de prédiction

Le Script

Pour la réalisation du script nous avons enregistré une base de test en format .json ainsi qu'un dictionnaire au format .pkl (avec le modèle, les encodeurs et les scalers). Le script va encoder et scale les features de la base de test puis on va ensuite prédire l'âge à partir du data_test et l'enregistrer au format json.

Exemple de résultat :



Système d'alerte pour les tempêtes

Pour ce besoin client, il nous a été demandé de créer un système d'alerte en cas de tempête, pour prédire quels arbres seraient susceptibles d'être déracinés si le vent est trop fort.

Pour cela, il nous a été demandé de prédire la feature `fk_arb_etat`.

Il nous a été indiqué que les arbres catégorisés comme `Essouché` devaient être considérés comme les arbres déracinés.

fk_arb_etat	count
EN PLACE	6645
SUPPRIMÉ	394
Essouché	165
REMPLACÉ	137
ABATTU	41
Non essouché	27

Premièrement, on tente de prédire toutes les catégories (`EN PLACE`, `SUPPRIMÉ`, `Essouché`, `REMPLACE`, `ABATTU`, `Non essouché`).

Cependant, on remarque que les `Essouché` ne représentent que 165 individus sur les 7409, soit à peine 2,2% et les résultats n'étaient pas concluants (même après SMOTE). Il a donc été décidé de ne garder que les lignes contenant `Essouché` et `Non essouché`.

Un OneHotEncoder a été utilisé pour encoder les variables catégorielles. Cela permet de ne pas influencer le modèle en ne hiérarchisant pas les différentes classes. Cela va créer une colonne booléenne par possibilité.

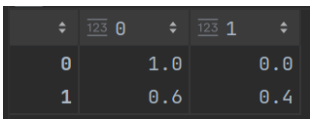
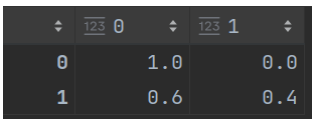
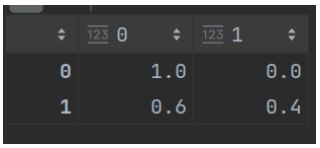
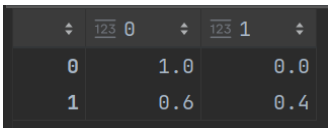
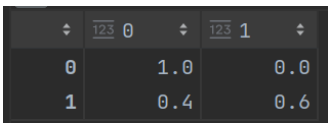
Afin de choisir les colonnes les plus pertinentes pour le modèle, une feature selection a été effectuée. Parmi les 165 colonnes (après OHE), les n plus pertinentes ont été sauvegardées, n=20 dans notre cas. Le feature importance va permuter des variables et examiner l'effet sur la sortie : plus la différence est grande, plus la variable est considérée comme importante.

	Feature	Importance
4	tronc_diam	0.064776
2	haut_tot	0.063204
1	latitude	0.053898
6	age_estim	0.047848
31	clc_secteur_E…	0.042228
0	longitude	0.040946
87	clc_secteur_S…	0.040637
3	haut_tronc	0.036857
159	villeca_VILLE	0.036351
13	clc_quartier_…	0.032509

Extrait trié par ordre d'importance.

Un SMOTE est appliqué pour tenter de faire remonter la part des éléments minoritaires.

Ci-dessous les matrices de confusion test pour les différents essais, avec des mise en forme et paramétrages différents :

RF + OHE fichier : rf_0.1.ipynb	
RF + OHE + grid search rf_0.2.ipynb	
RF + OHE + label selection rf_0.3.ipynb	
RF + OHE + label selection + grid search rf_0.4.ipynb	
RF + OHE + SMOTE + label selection + grid search rf_1.0.ipynb	

Le modèle RF fitté ainsi que ses encodeurs sont placés dans un dictionnaire et exportés avec la librairie pickle.

Le modèle d'apprentissage choisi est le RandomForestClassifier, pour les raisons suivantes:

- Classifieur pouvant faire du multi label ou binaire
- Algorithme très puissant
- Excellente accuracy
- Overfit maitrisé avec le bagging/bootstraping aléatoire (les échantillons)
- Résiste bien aux données bruitées grâce à son aléatoire (comme celles introduites par un SMOTE)
- Supporte bien les données déséquilibrées

On peut noter que d'autres modèles tels que MLPClassifier ou XGBClassifier de la lib xgboost ont été testés, mais n'ont pas apporté de résultats supérieurs au RandomForest.

Le modèle va faire une prédiction, qui peut être accompagnée de la probabilité associée.

```
gscv.classes_
Executed at 2024.06.20 15:49:51 in 45ms
array(['Essouché', 'Non essouché'], dtype=object)
```

0	1
0.985	0.015
0.980	0.020
0.810	0.190
0.850	0.150
0.875	0.125
0.870	0.130
0.765	0.235

On voit ici que `Essouché` est à l'indice 0 et `Non essouché` l'indice 1, et les retrouver dans leur colonne correspondante du `predict_proba`.

Concernant le script, il permet de prendre en argument : une date au format `aaaa-mm-jj`, un fichier pkl pour le modèle et un fichier json pour le dataset.

Il prédit les arbres potentiellement à risque d'après le modèle et créé une carte au format html.

Pour cela, la lib folium (leaflet pour python) a été utilisée.

Pour obtenir les données météo, c'est l'API de Visualcrossing qui a été utilisée en raison de sa capacité à fournir des données du passé et des prévisions à deux semaines. Elle est utilisée via des requêtes HTML et un retour au format JSON (en passant par la lib `requests` de python).

Les arbres affichés comme à risque d'être déracinés sont sélectionnés en fonction de la vitesse du vent et de leur probabilité True pour la prédiction de `fk_arb_etat`.
paliers inspirés de l'échelle de Beaufort :

```
62 a 74km/h       --> aficher ceux dont la probabilité True >0.9
75 a 88           --> afficher >0.8
89 a 102          --> afficher >0.7 arbres parfois déracinés
103 et plus       --> afficher >0.6
< 61km/h          --> rien, vent considéré comme trop faible
```

On peut cependant se questionner sur la pertinence du modèle dans le sens où aucune documentation à propos de la base de données n'est disponible, et la signification de `Essouché` reste floue. Par définition, cela correspond à l'action d' "*Enlever les souches qui sont restées dans un terrain, après l'abattage des arbres, soit par traction mécanique, soit par explosif", (Larousse, 2024)* ; ou encore "*Débarrasser (un terrain) des souches restées dans le sol après avoir abattu les arbres.*", (Le Robert, 2024).
Réaliser un modèle pour différencier les `Essouché` et les `Non essouché` (qui ont des caractéristiques très similaires voire identiques, mais dans un état d'avancement des travaux (leur abattage) différent) est donc une idée qui nous semble peu judicieuse.

Diagramme de Gantt