Rapport IA

Concevoir et développer une application d'étude du patrimoine arboré

Alexandre Guerrier (CSI3) - Carla Bossuyt (BIOST3) - Elea Lecointe (BIOST3)

Rendu le : 21/06/2024

Référents Big Data: Mme SETTOUTI Nesma – Mme ABDALLAH SAAB Nadine

Table des matières

Diagramme de Gantt

Groupe 3		Lundi M	Lundi A	Mardi M	Mardi A	Mercredi M	Mercredi A	Jeudi M	Vendredi M	Vendredi A
Gantt	Les 3									
Rapport	Les 3									
Diaporama	Les 3									
Soutenance	Les 3									
Besoin client 1 : Visualisation sur carte										
1. Préparation des Données :	carla									
2. Apprentissage Non Supervisé :	carla									
3. Métriques pour l'Apprentissage Non Supervisé :	carla									
4. Visualisation sur Carte :	carla									
5. Préparation du Script :	carla									
Fonctionnalité supplémentaire										
Besoin client 2 : Modèle de prédiction de l'âge										
1. Préparation des Données :	elea									
2. Apprentissage Supervisé pour la régression :	elea									
3. Métriques pour la classification :	elea									
4. Préparation du Script :	elea									
Fonctionnalité suplémentaire	carla									
Besoin client 3 : Système d'alerte pour les tempêtes										
1. Préparation des Données :	alexandre									
2. Apprentissage Supervisé pour la classification :	alexandre									
3. Métriques pour la classification :	alexandre									
4. Préparation du Script :	alexandre									
5. Fonctionnalite suplementaire	alexandre									

Introduction

Notre projet vise à concevoir et développer une application d'étude du patrimoine arboré de la ville de Saint-Quentin. Effectivement, les arbres jouent un rôle important dans les villes, que cela soit pour le bien-être des habitants ou encore pour la nature. Dans le but d'obtenir une gestion efficace de l'ensemble de ces arbres, il est nécessaire d'avoir une bonne compréhension de nombreux facteurs ; comme la taille des arbres ou bien leur âge. Ainsi, les technologies de l'intelligence artificielle vont permettre de faire des prédictions à l'aide de modèles, qui utilisent différentes méthodes d'apprentissage. Ces méthodes sont plus ou moins efficaces en fonction du jeu de données auquel ils ont à faire. Cela peut permettre de prédire les âges, les tailles à l'aide d'une visualisation sur une carte, mais aussi la mise en place d'un système d'alerte pour prédire les arbres susceptibles d'être déracinés en cas de tempête.

Besoin Client 1 - Visualisation sur carte

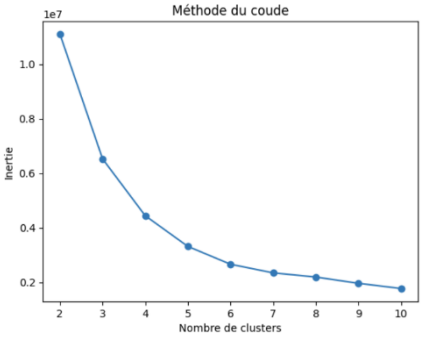
Le Besoin Client 1 concerne la création d'un visuel sur une carte qui sépare les arbres selon leur taille. En effet, l'utilisateur pourra choisir le nombre de catégories (par exemple : 2 catégories (petits, grands), 3 catégories (petits, moyens, grands), ...).

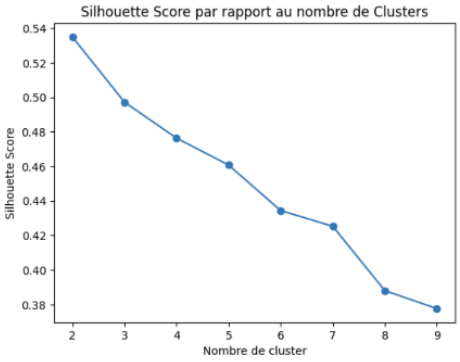
Premièrement, il faut préparer les données. Pour cela, nous extrayons les données d'intérêt. Nous réalisons une matrice de corrélation [1]entre la hauteur totale et les autres variables, dans le but de connaître les colonnes pertinentes qui nous serviront de base d'entraînement pour notre modèle. Les variables sont : la hauteur du tronc, le diamètre du tronc, le stade de développement et l'âge estimé. Pour réaliser notre matrice de corrélation ainsi que notre base d'entrainement, nous devons nous assurer que toutes les données sélectionnées soient numériques. Pour cela, nous utilisons le package « sklearn.prepocessing » et utilisons « OrdinalEncoder ». Cet estimateur transforme chaque caractéristique en donnée numérique de 0 à n_catégories-1. Par exemple pour le stade de développement, nous obtenons donc : 0 (Jeune), 1 (Adulte), 2 (vieux) et 3 (sénescent).

Dans un second temps, nous réalisons un apprentissage non supervisé sur nos données d'intérêt, en utilisant la méthode de clustering K-Means pour séparer les arbres en différents groupes, basés sur leur taille. Cette méthode prend en paramètre le nombre de clusters souhaité et se base sur la distance entre les points de données pour former ces groupes. Le processus commence par la sélection aléatoire de k points initiaux, appelés centroïdes. Ensuite, chaque observation est attribuée au centroïde le plus proche en utilisant la distance euclidienne. Les centres des clusters sont alors recalculés comme la moyenne de toutes les observations assignées à chaque cluster. Ces étapes d'attribution et de mise à jour sont répétées jusqu'à ce qu'un nombre maximal d'itérations soit atteint, ou que les centroïdes ne changent plus de manière significative. Dans notre cas, le résultat final est un ensemble de clusters bien définis qui regroupent les arbres en fonction de leur taille (2 clusters : petits et grands).

[1] *Statistique avec python*. (s. d.), à l'adresse https://asardell.github.io/statistique-python/

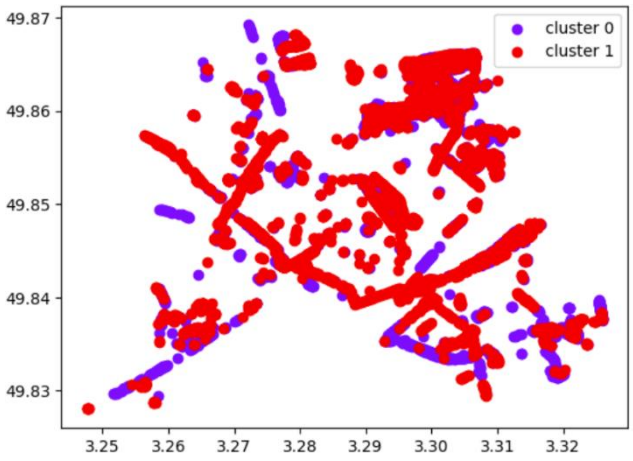
Ensuite, nous déterminons le nombre de clusters à choisir en utilisant la méthode du coude. Cette méthode est basée sur le principe que la somme des variances au sein des clusters peut être réduite en augmentant le nombre de clusters. Un nombre plus élevé de clusters permet de mieux distinguer des groupes de données plus homogènes. Pour identifier le nombre optimal de clusters, nous traçons la courbe de la somme des variances intra-clusters et cherchons le point de retournement, appelé "coude". Ce point indique le nombre de clusters à partir duquel l'ajout d'un cluster supplémentaire n'entraîne plus de réduction significative de la variance intra-cluster. Dans notre analyse, la courbe obtenue avec la méthode du coude ne présente pas de point de retournement distinct entre 2 et 10 clusters, ce qui signifie qu'aucun nombre spécifique de clusters ne se distingue nettement des autres





Une autre méthode d'évaluation est le « silhouette score », qui mesure la cohésion des échantillons au sein de leurs clusters respectifs, par rapport aux autres clusters. Ce score varie de -1 à 1 : un score proche de 1 indique une bonne séparation des clusters, tandis qu'un score proche de -1 suggère que les échantillons pourraient être mieux regroupés dans d'autres clusters. Dans notre analyse, le nombre optimal de clusters obtenu est n=2, avec une silhouette score de 0,535. Ce résultat indique que les échantillons sont le mieux regroupés dans 2 clusters distincts.

Carte qui sépare les arbres selon leur taille avec la bibliothèque Matplotlib : petits (cluster 1) et grands (cluster 0)

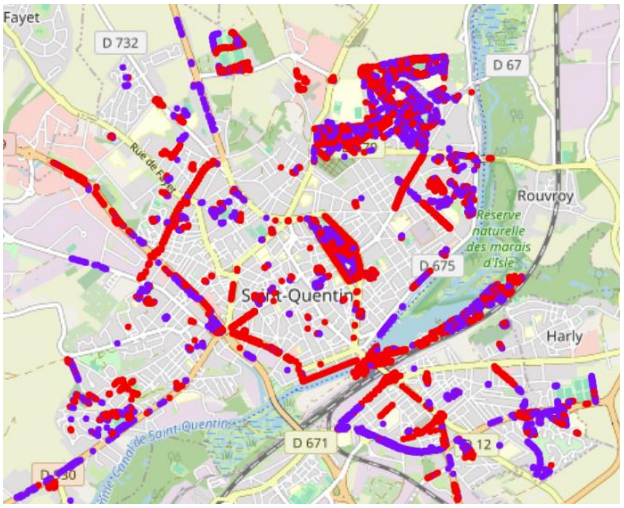


Puis, nous avons cartographié les arbres en utilisant des couleurs distinctes pour chaque cluster. Cette carte utilise les bibliothèques Matplotlib et numpy pour générer une carte de dispersion. Pour chaque cluster, nous avons généré une couleur distincte à partir de la palette de couleurs rainbow de Matplotlib[2]. Nous traçons un nuage de points (grâce à scatter) où chaque point représente la position géographique (longitude et latitude) d'un arbre, coloré en fonction de son cluster assigné. Cette visualisation permet de facilement identifier la répartition géographique des différents clusters d'arbres.

Pour connaître à quelle hauteur correspond chaque cluster, nous réalisons un boxplot sur la hauteur totale et déduisons que le cluster 0 (violet) correspond au cluster des arbres « grands » et le cluster 1 (rouge) aux arbres « petits ».

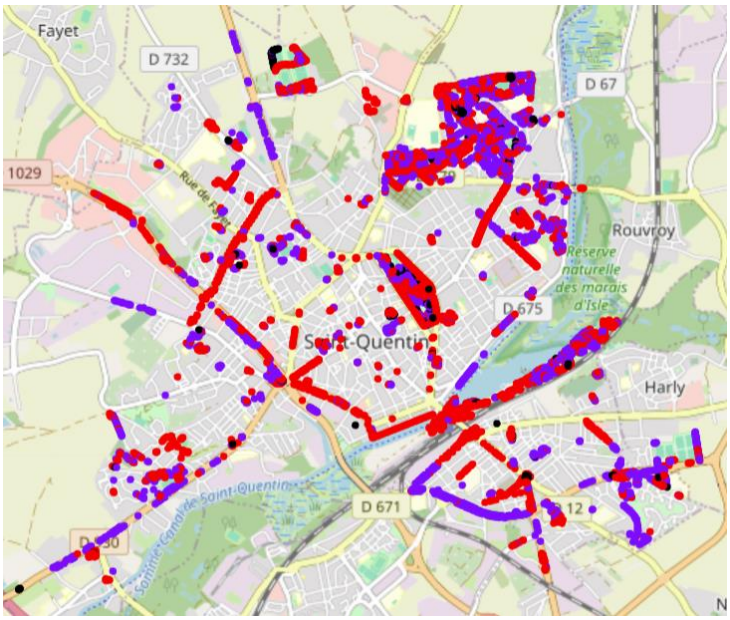
[2] Bergman, L. D., Rogowitz, B. E., & Treinish, L. A. (1995). A rule-based tool for assisting colormap selection. *Proceedings Visualization '95*, 118-125, https://doi.org/10.1109/VISUAL.1995.480803

Carte qui sépare les arbres selon leur taille avec la bibliothèque Folium : petits (rouge) et grands (violet)



Pour visualiser les clusters d'arbres de manière interactive, nous avons utilisé la bibliothèque Folium[3]. En centrant la carte sur les coordonnées moyennes des arbres, nous avons ajouté des marqueurs circulaires pour chaque arbre en utilisant leurs positions géographiques. Chaque marqueur est coloré en fonction du cluster auquel l'arbre est assigné, permettant ainsi de visualiser la répartition spatiale de ces derniers. Enfin, la carte est sauvegardée en format HTML pour une consultation interactive.

Carte qui sépare les arbres selon leur taille avec la bibliothèque Folium : petits (rouge) et grands (violet), ainsi que les anomalies (noir)



Enfin, nous avons intégré une fonctionnalité supplémentaire pour détecter les anomalies par le biais de l'algorithme « Isolation Forest »[4], une technique d'apprentissage automatique pour la détection d'anomalies, lui-même fondé sur l'algorithme « Decision Tree ». En spécifiant une proportion attendue de 5% d'anomalies, nous avons entraîné le modèle sur le DataFrame des données pertinentes. Les prédictions du modèle ont été ajoutées dans une nouvelle colonne du DataFrame, où une valeur de -1 indique la détection d'une anomalie. Ensuite, nous avons filtré ces anomalies et extrait les lignes correspondantes pour une analyse plus approfondie. Cette approche permet d'identifier les arbres qui diffèrent significativement des autres, signalant potentiellement des problèmes ou des caractéristiques uniques concernant la taille. Au total, nous avons identifié 371 anomalies. Chaque anomalie est représentée sur la carte par un marqueur circulaire noir, facilitant leur visualisation et leur analyse spatiale.

	feuillage	remarquable	cluster	anomalie
92	Feuillu	Non	0	-1
95	Feuillu	Non	0	-1
...				
97	Feuillu	Non	0	-1
98	Feuillu	Non	0	-1

Pour finir, nous créons la fonction principale qui prend en entrée le choix de l'utilisateur (nombre de catégories) et qui retourne une carte avec les positions des arbres colorées selon le résultat du clustering.

---

[3] *Using Leaflet and Folium to make interactive maps in Python*. (2016, août 1). Earth Data Science - Earth Lab. https://www.earthdatascience.org/tutorials/introduction-to-leaflet-animated-maps/

[4] *IsolationForest*. (s. d.). Scikit-Learn. Consulté 21 juin 2024, à l'adresse https://scikit-learn/stable/modules/generated/sklearn.ensemble.IsolationForest.html

Besoin Client 2 – Modèle de prédiction de l'âge

Le besoin du Client 2 consiste à développer un modèle d'intelligence artificielle capable de prédire l'âge des arbres. De ce fait, il faut trouver la meilleure méthode d'apprentissage supervisée, ainsi que de trouver les paramètres optimaux à placer dans cette dernière.

Pour cela, nous avons commencé par sélectionner uniquement les colonnes souhaitées parmi notre data "Data_Arbre.csv". Nous avons fait le choix de prendre les colonnes "haut_tot", "haut_tronc", "tronc_diam", "fk_stadedev", "feuillage", "fk_nomtech"; car d'après la matrice de corrélation, ce sont les variables qui semblent pouvoir impacter la prédiction de "age_estim", les autres variables ne semblent pas pertinentes selon ce besoin.

Dans le but de pouvoir réaliser des régressions, nous avons dû encoder nos valeurs qui n'étaient pas numériques, c'est à dire les colonnes "feuillage" ,"fk_stadedev" et "fk_nomtech".

Par la suite nous avons normalisé nos données, tout en discernant nos entrées (colonnes_age_estim), de notre sortie ("age_estim").

Après avoir découpé notre base de données en bases d'apprentissage (80%) et de test (20%), nous avons pu réaliser des GridSearchs sur nos différents modèles d'apprentissage. Le but était de trouver les meilleurs hyperparamètres pour obtenir les meilleures métriques possibles.

Dans un premier temps, nous avons réalisé des GridSearchs sur nos différents modèles. Nous avons tâtonné dans le but de trouver les meilleurs hyperparamètres. Ainsi nous avons obtenu ces résultats :

	Résultats hyperparamètres obtenus par GridSearch	Meilleur score GridSearch
Random Forest Regressor	max_depth': 20, 'max_leaf_nodes': 150, 'max_samples': 0.8, 'min_samples_split': 2, 'n_estimators': 300	0.80
Decision Tree Regressor	max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2	0.75
SVR	{'C': 10, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'rbf'}	0.77
MLP	activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'adaptive', 'solver': 'adam'	0.78

Les hyperparamètres qui nous sont retournés, sont alors placés en paramètres des modèles. Nous avons effectué l'apprentissage sur la base Train, puis la prédiction sur la base Test. Alors, nous avons pu évaluer nos modèles grâce à différents scores comme le R^2 (coefficient de détermination), le RMSE (Root Mean Squared Error), le MAE (Mean Absolute Error), le MSE (Mean Squared Error), ou encore grâce au score de validation croisée (CROSS VAL).

	R2	RMSE	MAE	MSE	CROSS VAL	TEMPS EXE (s)
Random Forest Regressor	0.78	0.47	0.30	0.22	0.73	1.80
Decision Tree Regressor	0.72	0.53	0.34	0.28	0.66	0.01
SVR	0.76	0.49	0.33	0.24	0.69	2.36
MLP	0.76	0.49	0.32	0.24	0.69	5.62

Pour donner l'exemple du RandomForestRegressor, nous obtenons un temps d'exécution qui est rapide (1,80 seconde)

Le R2 est tout à fait correct (supérieur à 0,5 et se rapprochant de 1), ceci indique que le modèle a une bonne capacité prédictive. Ainsi, 78% de la variance des données cibles (`age_estim`) est expliquée par le modèle.

Le RMSE de 0.47 indique l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles. Le RMSE est relativement faible, ce qui suggère que les prédictions du modèle sont assez proches des valeurs réelles.

Le MAE de 0.30 montre l'erreur absolue moyenne entre les prédictions et les valeurs réelles. Le MAE est assez bon, car une valeur faible indique des prédictions plus précises, ce qui confirme que le modèle fait des prédictions assez précises.

Le MSE de 0.22 quant à lui, est une autre mesure de l'erreur de prédiction ; elle est calculée en prenant la moyenne des carrés des erreurs. Le MSE est assez bas, il est donc cohérent avec les autres métriques d'erreur, indiquant ainsi des prédictions assez précises.

Le CROSS VAL de 0.73 est relativement élevé. Cela signifie que le modèle a une performance assez stable lorsqu'il est évalué sur différentes parties des données. Le score de CROSS VAL est assez proche de celui du R2, ce qui indique que le modèle n'est pas trop sur-ajusté (overfitting) et apprend bien sur des données qui n'ont pas été vues précédemment.

Grâce aux différents scores des modèles que l'on a pu obtenir, le modèle RandomForestRegressor semble être le meilleur dans l'ensemble. C'est donc ce dernier que l'on va retenir pour prédire l'âge des arbres. Une des principales qualités du RandomForest est la réduction de l'overfitting par rapport à un arbre de décision simple. En combinant les prédictions de plusieurs arbres, le modèle diminue la variance et donne des prédictions plus généralisables. Il est également relativement stable par rapport aux variations des données d'entraînement. C'est à dire que de petites modifications dans les données d'entraînement n'entraînent pas de grands changements dans les prédictions, contrairement aux arbres de décision individuels.

Par la suite, nous avons créé un fichier Pickle dans le but de récupérer les variables dont nous avons besoin, en format binaire, pour réaliser notre prédiction dans un fichier Json. Nous avons également créé un fichier Json contenant les valeurs de X_test (df2).

Grâce à la fonction principale "func_besoin2", nous avons pu lire le fichier Json et en retirer la colonne "age_estim", car pour réaliser notre prédiction de l'âge, nous ne devons pas l'utiliser. Par la suite, nous avons pu lire le fichier Pickle, prédire les données X_test, et enfin enlever la normalisation des données "age_estim". Finalement, nous avons transformé notre fichier Pickle en DataFrame, qui a enfin été transformé en format Json. Nous avons alors obtenu un fichier Json contenant les âges prédits pour chaque arbre.

Fonctionnalité supplémentaire :

L'arbre de décision (CART)[5] permet d'expliquer une variable cible à partir d'autres variables. L'arbre CART est composé d'un nœud racine qui contient les données. Il y a également des nœuds internes qui sont des points de divisions où l'arbre se divise en branches binaires selon certaines conditions. En bas de l'arbre, se trouvent des feuilles, qui sont des nœuds terminaux fournissant la prédiction finale. Pour choisir la variable à diviser, ainsi que le seuil de division, l'impureté (indice de Gini ou Entropie) est calculée pour chaque variable et seuil de la classification. Pour les variables de régression, la MSE (erreur quadrique statique) est calculée pour chaque seuil. La variable et le seuil obtenant le seuil le plus bas est choisi pour former un sous-arbre gauche et un sous-arbre droit. Pour éviter le surapprentissage (overfitting), CART contient des critères d'arrêts : la profondeur maximale de l'arbre, le nombre minimum d'échantillons par nœud, et le gain d'impuretés requis pour justifier une division. A la fin de notre arbre, les feuilles représentent les prédictions. Pour la classification, la valeur prédite est la classe majoritaire des échantillons de cette feuille. Concernant la régression, la valeur prédite est la moyenne des valeurs des échantillons dans cette feuille.

Besoin Client 3 – Système d'alerte pour les tempêtes

Les demandes du Client 3 consistaient à mettre en place un système d'alerte pour prédire les arbres qui sont susceptibles d'être déracinés en cas de tempête.
Nous avons donc utilisé plusieurs approches pour savoir quel type de modèle serait le plus adapté au besoin de ce client.
L'idée est de faire une classification des arbres qui sont arrachés par la tempête ou non. Comme base d'apprentissage, les arbres dont l'état est 'essouché' sont considérés comme arrachés par une tempête. Les autres arbres sont considérés comme non atteint. Nous pouvons aussi choisir « essouché » / « non essouché », cependant cela réduit énormément le jeu de données.
Un autre problème se pose, le jeu de données est déséquilibré. Près de 98% des arbres sont en place et le reste est essouché. Il a donc fallu mettre en place une méthode pour retrouver un équilibre sur ces données, ainsi que des modèles adaptés à cette dernière.

Procédé mis en place :

1. Préparation des données

Récupération des données fournies via la librairie Pandas et utilisation de toutes les colonnes de la base de données.

Séparations des données : catégorielles, numériques, booléennes.

Pour les données catégorielles, utilisation du OneHotEncoder de Sklearn. L'encodeur va créer une colonne par type de données différentes. Par exemple, pour les quartiers, l'encodeur fera une colonne par quartier et affectera un 1 à la colonne correspondant au quartier de l'arbre. Cela transforme les données catégorielles, qui ne sont mal interprétées par le modèle, en un vecteur de zéro / un, et permet de mettre au même niveau d'importance les valeurs catégorielles.

[5] *Cart/cart.py at master · zziz/cart*. (s. d.). GitHub. Consulté 21 juin 2024, à l'adresse https://github.com/zziz/cart/blob/master/cart.py

Pour les données numériques, un StandardScaler est utilisé pour mettre sur la même échelle toutes les données.

Les valeurs booléennes sont remplacées par des 0 ou 1 en fonction de ce que l'on souhaite.

On divise le jeu de données en base d'apprentissage Train, et en base de Test.

On utilise SMOTE pour équilibrer la minorité dans la base de Train. Cela apporte plus de diversité pour l'apprentissage. SMOTE va dupliquer les valeurs minoritaires et ajouter un certain degré de bruit pour qu'elles ne soient pas exactement identiques.

2. Apprentissage du modèle

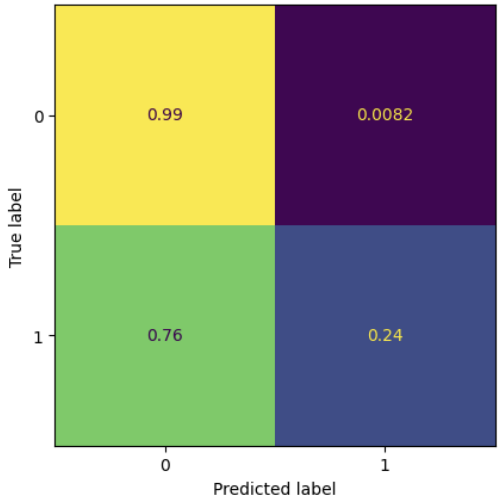
Nous avons commencé par l'apprentissage d'un premier modèle de RandomForestClassifier, puis par la suite par l'extraction des meilleures features avec un feature selection. Pour déterminer l'importance des features, le classifieur va en enlever et/ou changer l'ordre de celles-ci, et ainsi calculer la différence de score des différents arrangements. Plus le score est différent, et plus la feature est importante. Un classement à la fin a pu être refait.

Par la suite, nous avons pu créer un nouveau modèle avec les features retenues.

Nous avons trouvé les hyperparamètres du modèle à l'aide un GridSearchCV. On a réalisé un dictionnaire avec une liste de paramètres à tester, ainsi qu'une méthode de scoring (ici le score 'F1'), pour avoir un score sur la matrice confusion.

3. Métrique pour la classification

Nous avons fait de choix d'utiliser le score F1 de la matrice de confusion, car sur un scoring accuracy, les résultats auraient été biaisés sur la base de test. En effet, en raison du déséquilibre des données, même si le modèle classifiait au hasard l'état de l'arbre testé, le score resterait élevé. C'est pour cela que nous regardons la matrice de confusion afin de constater si les arbres sont bien classifiés ou non. Il serait possible d'utiliser d'autres métriques de la matrice, comme le rappel ou la précision.



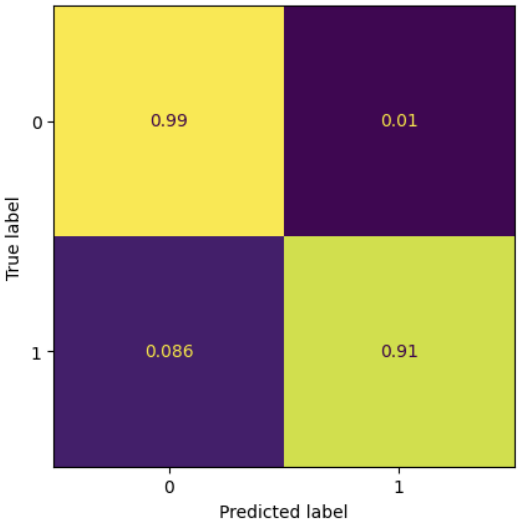
Matrice de confusion base de Test

Voici la matrice de confusion du modèle final, sur la base de test. On observe que le modèle classifie bien les positifs, avec un vrai négatif à 0.99. Cependant, on peut remarquer qu'il a plus de mal à classifier les négatifs. On se retrouve avec un taux de 0.76 de faux positifs.

Malgré plusieurs méthodes et paramétrages différents, les résultats restent assez mauvais sur la matrice de test.

Cependant la matrice pour la validation montre que le modèle a très bien appris sur la base de données.

On a donc un modèle validé, mais des résultats incohérents pour les prédictions.



Matrice de confusion cross val base de test

Grâce aux résultats que nous avons obtenus précédemment, nous avons testé plusieurs autres modèles pour élargir nos résultats (toujours sur des données smote, mais avec la méthode essouché/non essouché comme apprentissage) :

<u>Modèle DummyClassifier :</u>

Matrice de validation :
```
[1., 0.]
[1., 0.]
```
Ici le modèle ne passe pas la validation.

<u>Modèle MLP (réseau de neurones) :</u>

Matrice de validation :
```
[0., 1.]
[0., 1.]
```
Le modèle ne passe pas non plus le stade de validation

<u>Modèle AdaBoost :</u>

Matrice de validation :
```
[0.98473282, 0.01526718]
[0.        , 1.        ]
```
Ici la validation passe avec une bonne matrice de confusion.

Matrice de test :
```
[0.2       , 0.8       ]
[0.23529412, 0.76470588]
```
Sur la base de test le modèle devient incohérent sur la classification

Le AdaBoost est un modèle qui combine plusieurs Weak Learner combiné avec une méthode de boosting entre les Weak Learner utilisé.

On a pu observer durant la préparation du modèle, que la base de données n'était pas adaptée à ce que l'on souhaitait faire. En effet, le déséquilibre n'affecte pas plus que ça le résultat du modèle RandomForest. Ce dernier ajoute de la diversité par lui-même dans le cas de données déséquilibrées.

Le modèle ne peut pas faire de lien entre les features car la base de données n'est pas adaptée à nos besoins. Un apprentissage plus précis tel que 'essouché'/'non essouché' nous donne un modèle over-fit qui ne peut pas classifier d'autre arbre que sa base d'entrainement. Cela est du au manque de données. Cet apprentissage nous limite à ~160 cas différents.

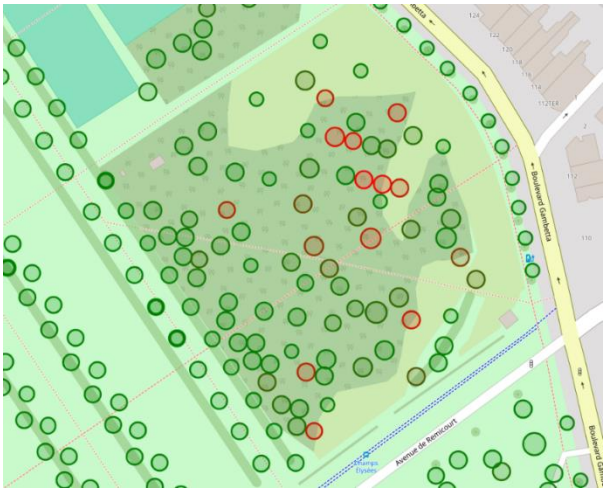
4. Préparation du script

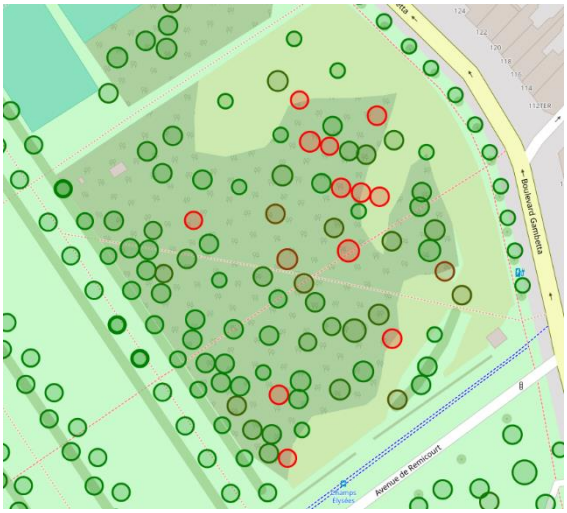
La demande du client était de réaliser une prédiction des arbres qui seraient probablement déracinés par les prochains évènements météorologiques. Nous avons utilisé la librairie Folium de python (Leaflet sur R) pour illustrer sous forme de carte, les arbres qui sont susceptibles d'être essouchés. Le script va donc récupérer le modèle ainsi que les encodages utilisés pour faire une prédiction sur la base de données utilisée. La prédiction retourne une probabilité que l'arbre soit essouché.

Nous avons également implémenté des données météo (ici la vitesse du vent), sur un jour donné en paramètre, qui va déterminer en fonction de la probabilité associé à chaque arbre, si celui-ci sera affecté ou non. L'implémentation est faite à l'aide d'une API qui nous communique des informations souhaitées.

Carte disponible ICI

La carte sans prise en compte de la météo. Un gradient de couleur va de vert à rouge si la probabilité est plus proche de 0 ou 1 (1 = arbre probablement touché)





La carte avec prise en compte de la météo. On observe que certains arbres sont passés à une probabilité de 1, par rapport à la carte précédente.

Conclusion

Le projet d'étude du patrimoine arboré de la ville de Saint-Quentin a permis d'exploiter les données à l'aide des prédictions d'intelligence artificielle pour répondre à des besoins concernant la gestion et la préservation des arbres urbains.

Pour le Besoin Client 1, la visualisation sur carte des arbres selon leur taille a été réalisée en utilisant des techniques de clustering. Le modèle K-Means a permis de séparer les arbres en différentes catégories de taille, offrant une représentation claire de leur répartition géographique. L'usage des méthodes du coude et du silhouette score a permis de déterminer le nombre optimal de clusters, tandis que l'algorithme Isolation Forest a détecté les anomalies.

Le Besoin Client 2 a été demandé dans le but de développer un modèle d'intelligence artificielle pour prédire l'âge des arbres. Le RandomForestRegressor s'est avéré être le modèle le plus performant, en fournissant des prédictions précises grâce à des métriques comme le R^2, le RMSE, le MAE, le MSE, ou encore le score de validation croisée. La création de fichiers Pickle et Json a permis de faciliter l'intégration et l'utilisation des prédictions, ainsi que leur visualisation finale.

Pour le Besoin Client 3, plusieurs approches ont été testées pour prédire les arbres susceptibles d'être déracinés par des tempêtes. Bien que le modèle RandomForestClassifier ait montré de bons résultats lors de la validation, il n'a pas réussi à bien prédire sur la base de test en raison des données déséquilibrées, ou manquantes. D'autres modèles, tels que l'AdaBoost, ont également été testés, mais avec des succès limités. Cette partie du projet nous a montré la nécessité d'un jeu de données mieux adapté pour des prédictions plus fiables.

En conclusion, ce projet démontre la capacité de l'intelligence artificielle à fournir des outils efficaces pour la gestion du patrimoine des arbres de Saint-Quentin. Les visualisations et l'adaptation des modèles d'intelligence artificielle permettent des prédictions, afin de gérer les arbres urbains. La création d'une application en développement Web associé aux prédictions des modèles d'intelligence artificielle pourra devenir un outil précieux pour la ville de Saint-Quentin.