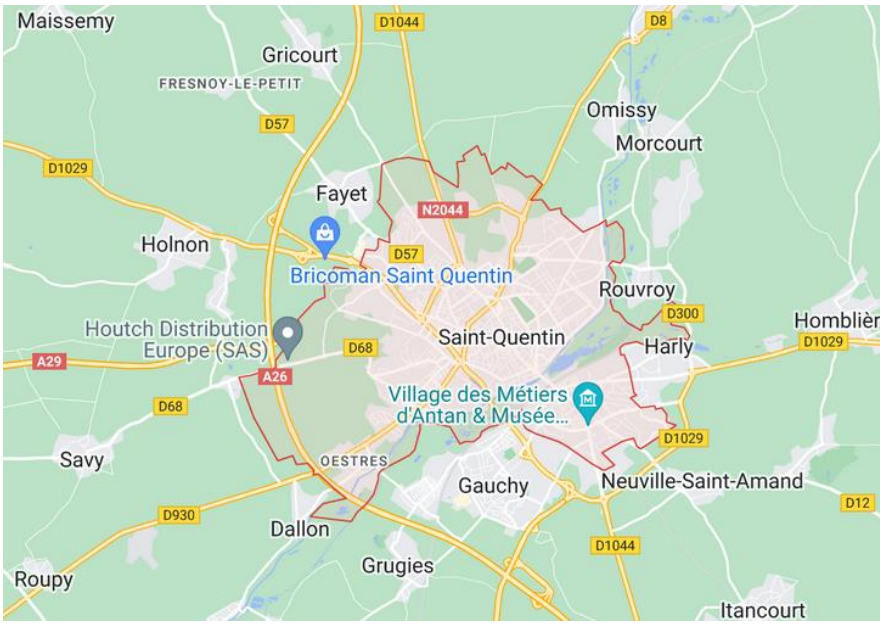
Rapport
Projet de
Big Data
3ème année



Groupe 4

Arthur Grossmann--Le Mauguen

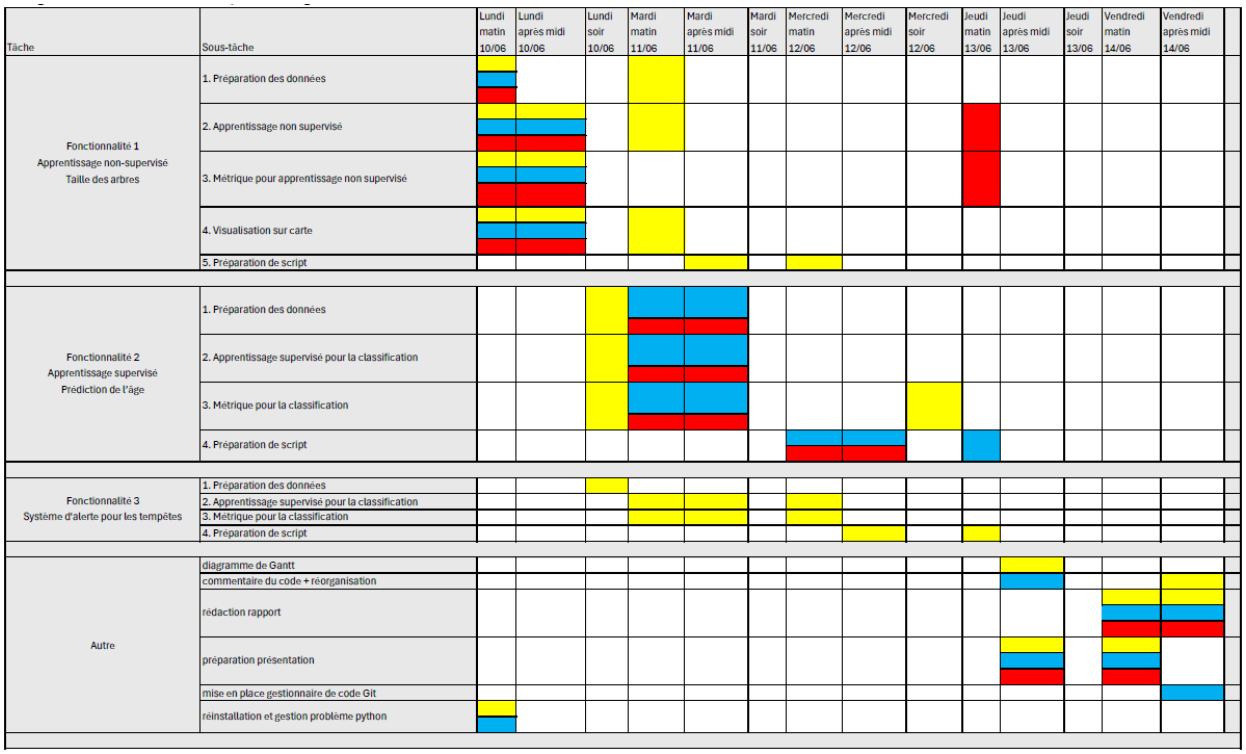
Enzo Guillard

Lucas Bercegeay

CSI 3 Nantes

Table des matières

Diagramme de Gant :

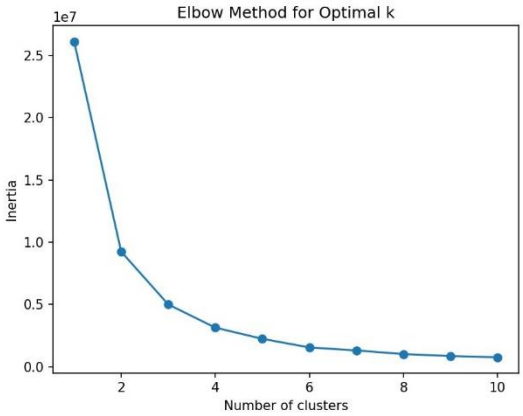


Membres	couleur correspondante
Arthur GROSSMANN--LE MAUGUEN	
Enzo GUILLARD	
Lucas BERCEGEAY	

Fonctionnalité 1 : Apprentissage non supervisée : Taille des arbres

Dans cette partie, l'objectif principal était de déterminer la Taille des arbres du jeu de données. Dans un premier temps, le but était de préparer les données. D'abord, nous avons extrait du fichier csv les colonnes qui nous semblaient les plus pertinentes pour l'étude de la fonctionnalité. Nous avons choisi de garder les 3 colonnes suivantes : 'tronc_diam', 'haut_tot' et 'haut_tronc', l'encodage des données dans ce cas-là n'était pas nécessaires car il s'agissait seulement de données numériques.

Dans un second temps, nous devions choisir un algorithme de clustering pour séparer les arbres en différents groupes basés sur leur taille. Nous avions à disposition la bibliothèque Sklearn, qui nous proposais plusieurs algorithmes, mais nous avons choisis la méthode Kmeans, que nous avons déjà vu en cours donc qui était plus facile à manipuler. Afin de déterminer le nombre de clusters idéal pour la suite, nous avons utilisé la "elbow method" (méthode du coude). Nous avons donc exécuté le modèle Kmeans pour différents nombres de Clusters. Pour notre part, nous avons décidé de choisir un nombre de Clusters entre 2 et 10. Puis nous avons calculé l'inertie et nous avons tracé la courbe de l'inertie en fonction du nombre de Clusters. Nous obtenons le résultat suivant :



Étapes de la Méthode du Coude:

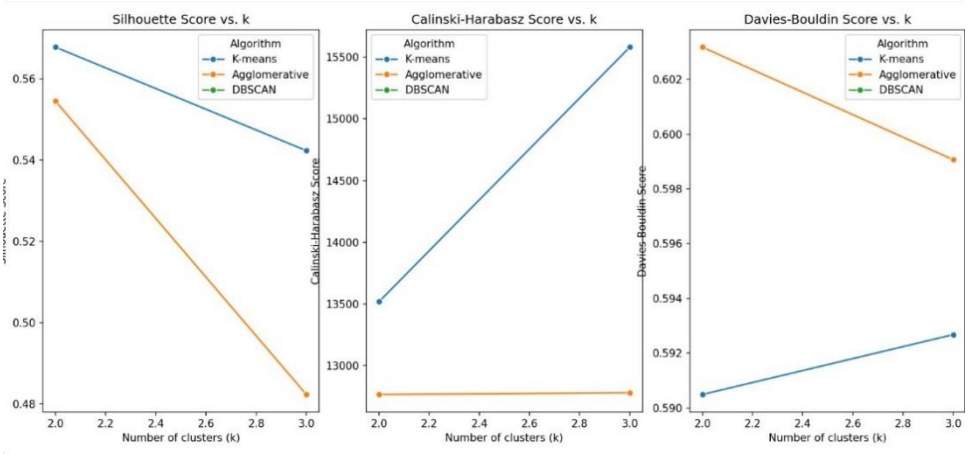
1. Exécuter K-means pour différents nombres de clusters (K)
2. Calculer l'inertie (SSD)
3. Tracer l'inertie en fonction de K
4. Identifier le "coude" dans le graphique

Rq: peut-être subjectif sur le choix du K

On identifie maintenant le "coude" sur la courbe qui est l'endroit où l'on peut observer une cassure. Ici, on observe que le nombre de clusters idéale est 2 ou 3. Pour notre part, nous avons choisis 2 clusters. Nous avons ensuite évalué notre modèle Kmeans à l'aide de 3 métriques : Silhouette Coefficient, Calinski-Harabasz Index et Davies-Bouldin Index. Pour le premier métrique, le mieux était d'être proche de 1, pour le deuxième il fallait avoir le score le plus élevé possible et pour le troisième, il fallait un score proche de 0. Nous avons comparé notre méthode Kmeans avec 2 autres modèles : Agglomerative Clustering et DBSCAN. Le fait de comparer avec ces 2 modèles nous a permis d'observer que Kmeans est le plus efficace. Nous obtenons les résultats suivants :

méthode	Données normalisées?	Silhouette Score	Calinski-Harabasz Index	Davies-Bouldin Index
KMeans	NON	0,57	13 500	0,59
KMeans	OUI	0,48	9 300	0,77
Agglomerative Clustering	NON	0,52	9 900	0,61
Agglomerative Clustering	OUI	0,47	8 400	0,77
DBSCAN	NON	0,34	19,4	2,93
DBSCAN	OUI	0,51	123	0,53

Nous avons décidé de ne pas normaliser les données puisque lorsque l'on normalisait les données nous remarquions que les scores étaient moins bon. Nous avons donc tracé des courbes pour comparer les 3 méthodes en fonction des 3 métriques :



On obtient donc les 3 courbes suivantes. On peut voir que la courbe de la méthode DBSCAN n'apparaît pas car les valeurs sont très éloignées des scores idéals de chaque métrique. En comparant les méthodes Kmeans et Agglomerative Clustering, on voit clairement que pour chaque métrique la méthode Kmeans est largement plus efficace.

Pour terminer, nous avons affiché les arbres sur une carte que nous avons chargé en html. On identifie donc bien 2 clusters différents :

Fonctionnalité 2 : Apprentissage supervisé : Prédiction de l'âge

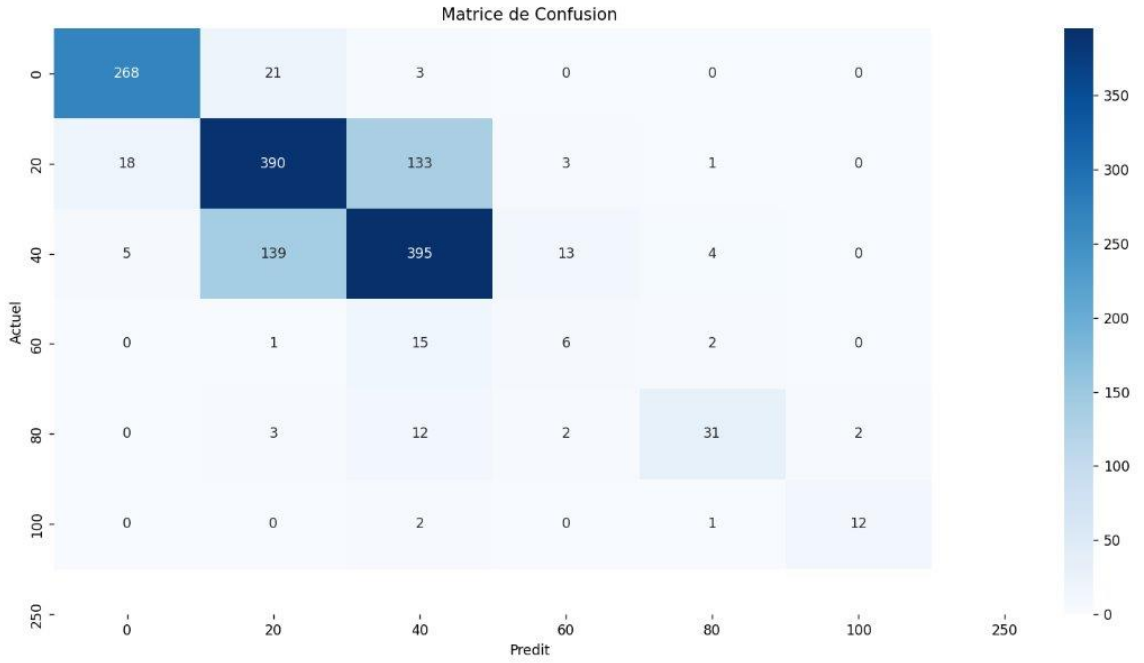
Le but de cette fonctionnalité était de prédire l'âge d'un arbre suivant les caractéristiques choisis qui sont : 'haut_tot', 'haut_tronc', 'tronc_diam' et 'fk_stadedev'. Pour commencer, nous avons préparé les données. Pour ce faire, nous avons pris les colonnes de la base de données qui nous intéressaient, nous les avons encodées, principalement utile pour la colonne catégorielle 'fk_stadedev'. Ensuite, nous avons normalisé les données et nous les avons séparés en différent jeux de données : 80% en base d'apprentissage et 20% en base de test. Une fois cette étape terminée, nous avons utilisé trois méthodes différentes : RandomForestClassifier, DecisionTreeClassifier et KneighborClassifier. On a ensuite effectué un GridSearch pour retourner les meilleurs paramètres et le meilleur modèle de chaque méthode. Pour ce faire, nous avions ultérieurement choisit les paramètres pour effectuer cette étape spécifique à chaque modèle. Voici les meilleurs paramètres que nous obtenons pour chaque méthode :

RandomForestClassifier(max_features=3, n_estimators=200, random_state=42)

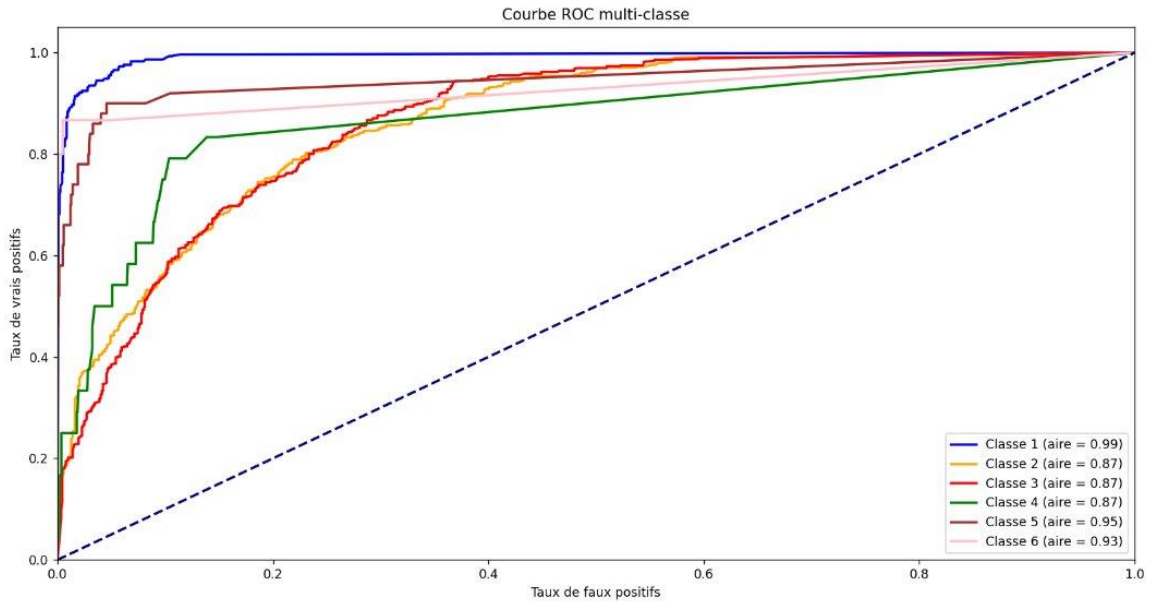
DecisionTreeClassifier(min_samples_leaf=8, min_samples_split=20)

KNeighborsClassifier(leaf_size=50, n_neighbors=9)

Parallèlement, nous avons effectué une prédiction pour savoir dans quelles classes d'âges l'arbre est affecté et nous avons tracé les matrices de confusions et les courbes de ROC pour chaque méthode. Voici un exemple de matrice de confusion pour la méthode RandomForest :



Et voici un exemple de courbe de ROC pour RandomForestClassifier :



Pour expliquer brièvement, la classe 1 correspond à la tranche 0-20 ans, la classe 2 à la tranche 20-40 ans et ainsi de suite. On remarque que pour le modèle prédit plutôt bien la bonne classe d'âge dans la matrice de confusion même s'il se trompe à quelques endroits. Pour la matrice de confusion, l'aire sous la courbe est proche de 1 ce qui signifie que le modèle est bon.

Après avoir tracer graphiquement les matrices de confusion et les courbes ROC, nous avons calculé des métriques pour évaluer le modèle. Nous avons calculé la précision et l'exactitude pour chaque méthode et nous avons obtenus les résultats suivants :

Classifiction_Report	RandomForest	DecisionTree	Kneighbor
Exactitude	0.7435	0.7435	0.7510
Précision	0.7445	0.7419	0.7509

On remarque que toutes les méthodes ont une exactitude et une précision similaire, il n'y a pas un modèle qui surpasse les autres. Pour appuyer sur les métriques, nous avons décidé de le faire pour chaque classe d'âge, comme on peut le voir ci-dessous avec l'exemple de la méthode RandomForestClassifier :

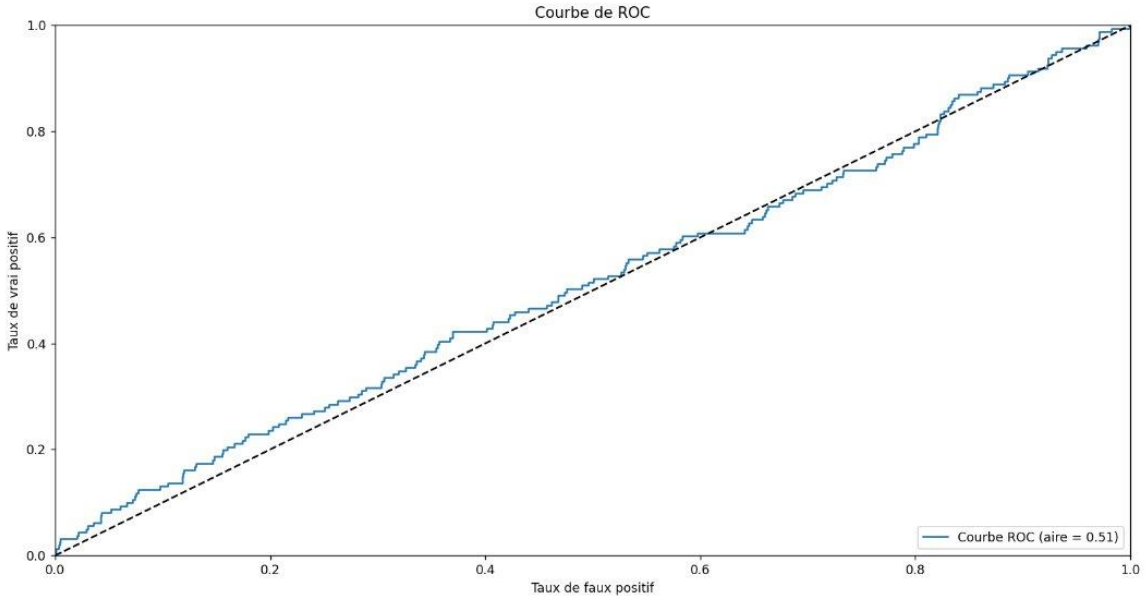
	precision	recall	F1-score	Support
1	0.92	0.92	0.92	292
2	0.7	0.72	0.71	545
3	0.71	0.71	0.71	556
4	0.25	0.25	0.25	24
5	0.79	0.62	0.7	50
6	0.86	0.8	0.83	15
accuracy	-	-	0.74	1482
Macro avg	0.71	0.67	0.69	1482
Weighted avg	0.74	0.74	0.74	1482

On remarque ici que les classes n'ont pas la même précision pour chaque classe puisque pour cette méthode, on choisit aléatoirement ce qui fait qu'on peut avoir une classe avec moins d'arbres que d'autres, mais en globalité, on peut remarquer qu'on a des métriques assez bonnes pour chaque classe d'âge.

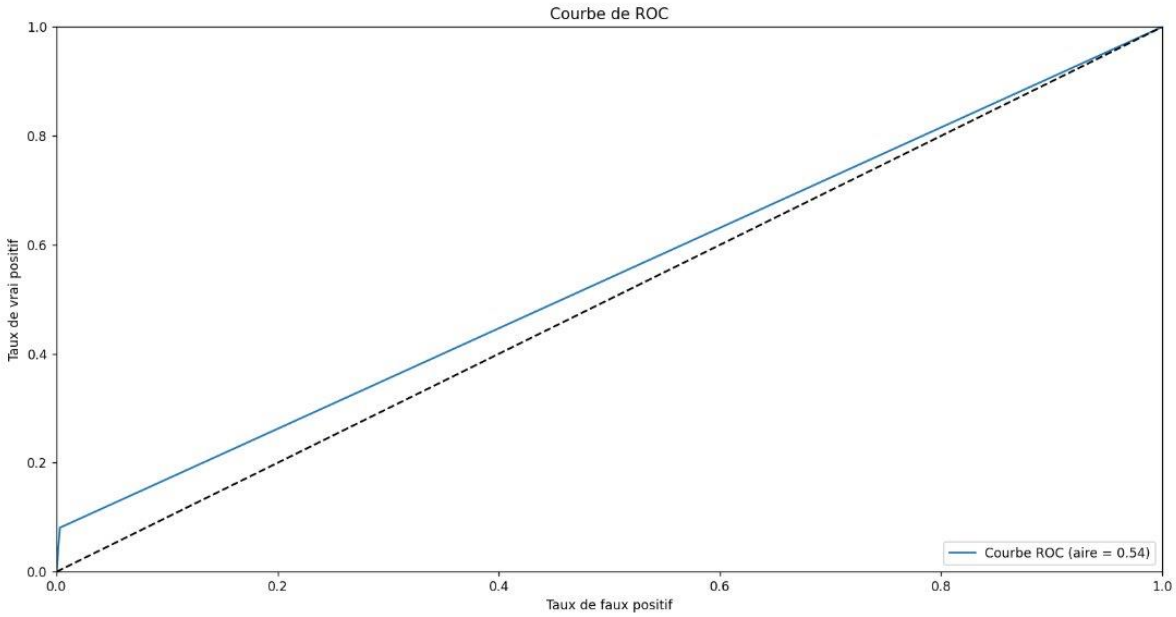
Fonctionnalité 3 : Apprentissage supervisée : Système d'alerte pour les tempêtes

Pour cette fonctionnalité, nous devions prédire si l'arbre en entrée est susceptible d'être déracinés lors d'une tempête future, est-il toujours debout ou abattu. Pour commencer, nous avons préparé les données du CSV. Pour ce faire, nous avons pris les colonnes de la base de données qui nous intéressaient, nous avons encodées les colonnes catégorielles 'fk_pied' et 'clc_secteur' et nous avons également converti les chaines de caractères en minuscules. Ensuite, nous avons normalisé les données avec ordinalencoder et nous les avons séparés en différent jeux de données : 80% en base d'apprentissage et 20% en base de test avec split. Une fois cette étape terminée, nous avons utilisé deux méthodes différentes : SGDClassifier et RandomForestClassifier. Une fois la méthode appliquée, nous avons entrainés 4 modèles : SGDClassifier normalisé et non normalisé et RandomForestClassifier normalisé et non normalisé. Nous avons ensuite effectué une validation croisé pour retourner les meilleurs hyperparamètres et le meilleur modèle de chaque méthode. Pour ce faire, nous avions ultérieurement choisit les paramètres pour effectuer cette étape spécifique à chaque modèle.

Parallèlement, nous avons effectué une prédiction et nous avons tracé les matrices de confusions et les courbes de ROC pour chaque méthode. Le premier exemple est la méthode SGDClassifier et la deuxième est la méthode RandomForestClassifier :

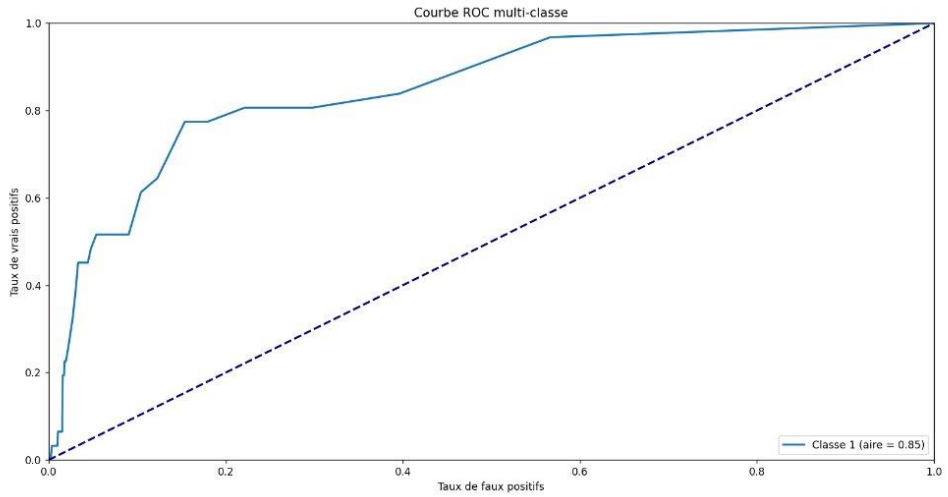


Matrice de confusion	**Négatif Réel**	**Positif Réel**
Négatif Prédiction	5 749	17
Positif Prédiction	161	0



Matrice de confusion	Négatif Réel	Positif Réel
Négatif Prédiction	1 449	2
Positif Prédiction	23	8

Grâce aux courbes ROC, on en a déduit que la meilleure méthode était la méthode RandomForest puisque son aire sous la courbe est meilleure, même si cela reste très proche. On l'a donc optimisé avec le GridSearch pour avoir la meilleure optimisation et nous avons obtenu cela : Meilleurs hyperparamètres : max_features = 5 et n_estimators = 1. En effectuant cela, nous avons de nouveau tracer la matrice de confusion et la courbe ROC et nous avons obtenu le graphique ci-dessous, avec une aire sous la courbe nettement supérieure que précédemment :



	précision	recall	F1-score	Support
0	0.98	1,00	0.99	1 451
1	0.25	0.03	0.06	31
accuracy			0.98	1482
Macro avg	0.49	0.50	0.49	1482
Weighted avg	0.96	0.98	0.97	1482

On remarque que le modèle a de très bonnes performances pour la classe majoritaire (classe 0) mais des performances très faibles pour la classe minoritaire (classe 1). Cela peut être problématique dans des applications où détecter correctement la classe minoritaire est crucial. Des ajustements supplémentaires, comme la rééchantillonnage des données ou l'utilisation de métriques spécifiques aux classes, pourraient être nécessaires pour améliorer les performances sur la classe minoritaire.

Fonctionnalité 4 : Création de scripts utilisables en ligne de commande

Pour cette fonctionnalité, nous avons créé 3 scripts différents pour chaque fonctionnalité réalisée auparavant.

Pour la première fonctionnalité, nous avons fait un script qui prend en entrée dans le terminal 'haut_tot', 'haut_tronc' et 'tronc_diam. Une fois cette ligne rentrée dans le terminal, le code va chercher le fichier 'Centroïde.csv' créer ci-dessus et il va nous retourner à l'aide de ce fichier le numéro du cluster de l'arbre qu'on va venir stocker dans un fichier .json.

Pour la fonctionnalité 2, c'est le même principe, on rentre dans le terminal la méthode souhaitée ainsi que 'haut_tot', 'haut_tronc', 'tronc_diam' et 'fk_stadedev'. On encode la donnée 'fk_stadedev' pour la mettre en numérique et on standardise les données rentrer et grâce aux fichiers .pkl qu'on a créé lors de la fonctionnalité 2, on retourne la classe de l'âge prédit qu'on vient stocker dans un fichier .json.

Pour la fonctionnalité 3, c'est encore la même chose, on rentre en entrée dans le terminal 'longitude', 'latitude', 'clc_secteur', 'haut_tronc', 'tronc_diam' et 'age_estim'. On encode les données catégorielles et on standardise le tout. Par le même processus, on utilise le fichier pkl créé dans la fonctionnalité 3 et on prédit si l'arbre va rester debout après la tempête. A nouveau, on va venir stocker le résultat dans un fichier .json. Le but de stocker ses fichiers dans un .json, c'est qu'on va pouvoir s'en servir pour l'utiliser lors du projet de WEB.

Sources

Pour nous aider dans notre étude nous avons utilisés des informations présentes sur les sites suivants :

https://scikit-learn.org/stable/index.html

Annexes

Schéma résumé fonctionnalité 1 :

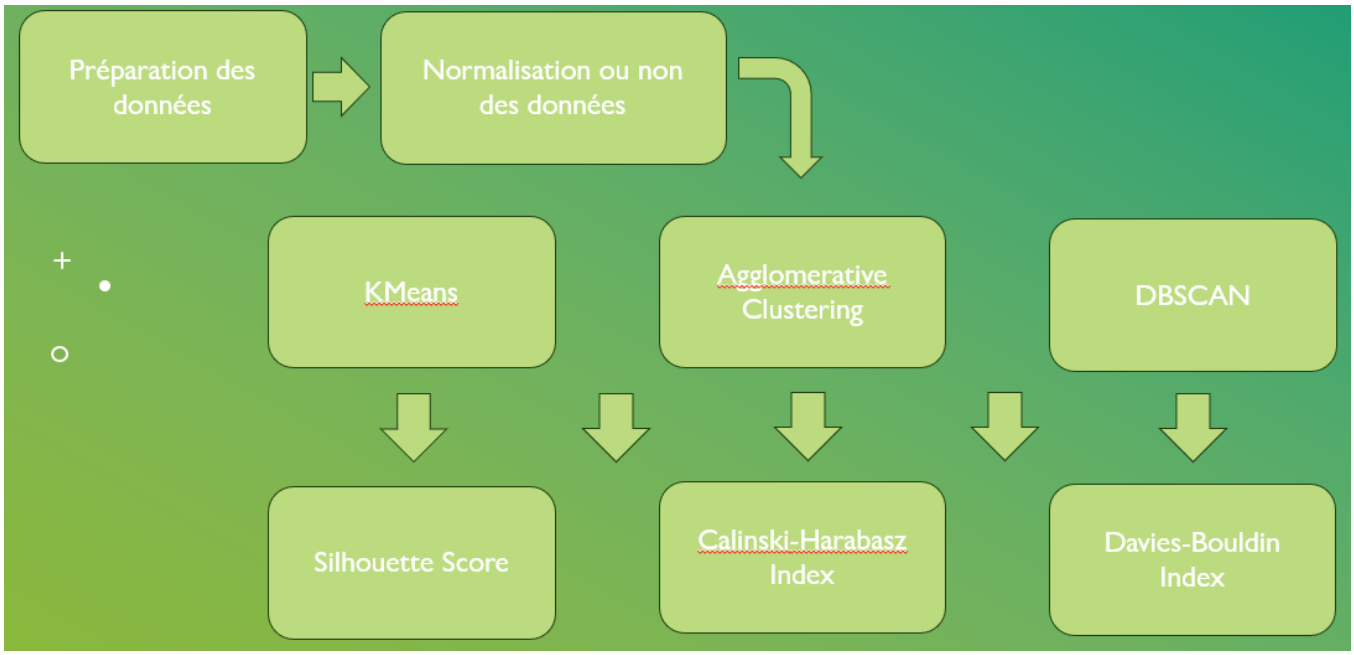


Schéma résumé fonctionnalité 2 :

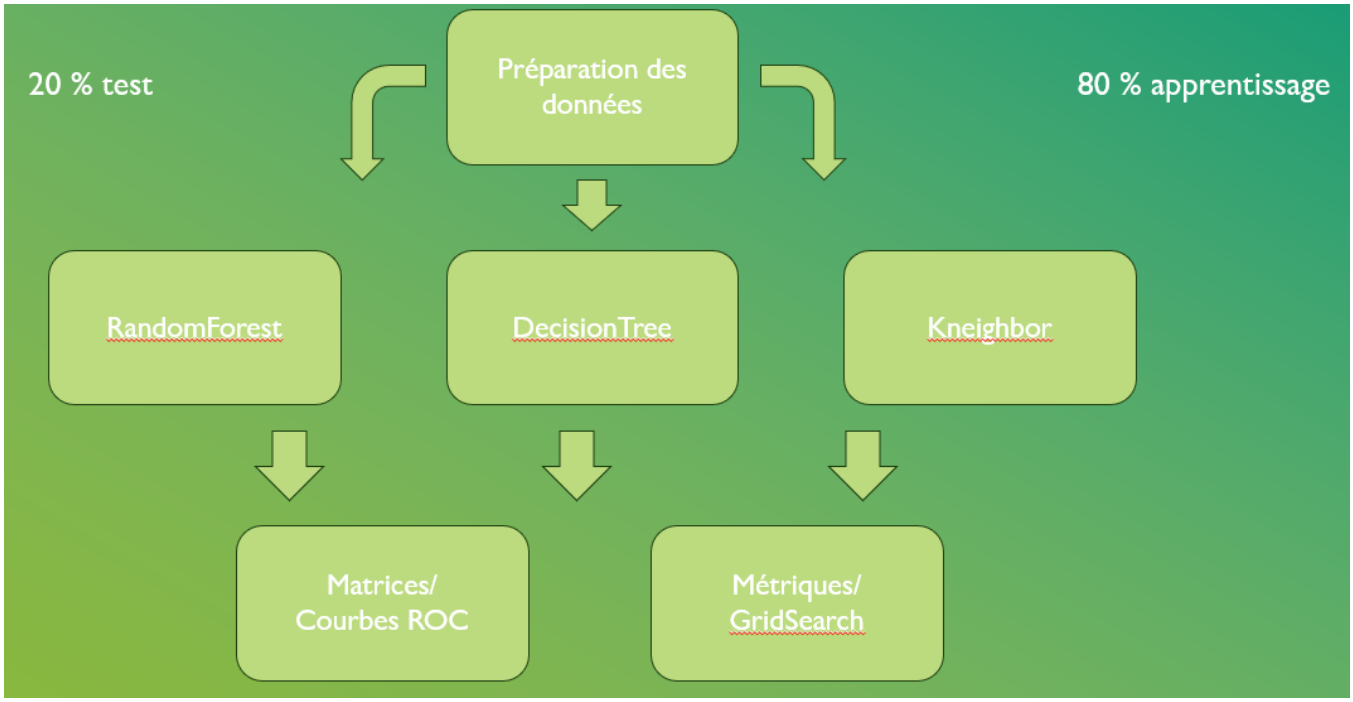


Schéma résumé fonctionnalité 3 :

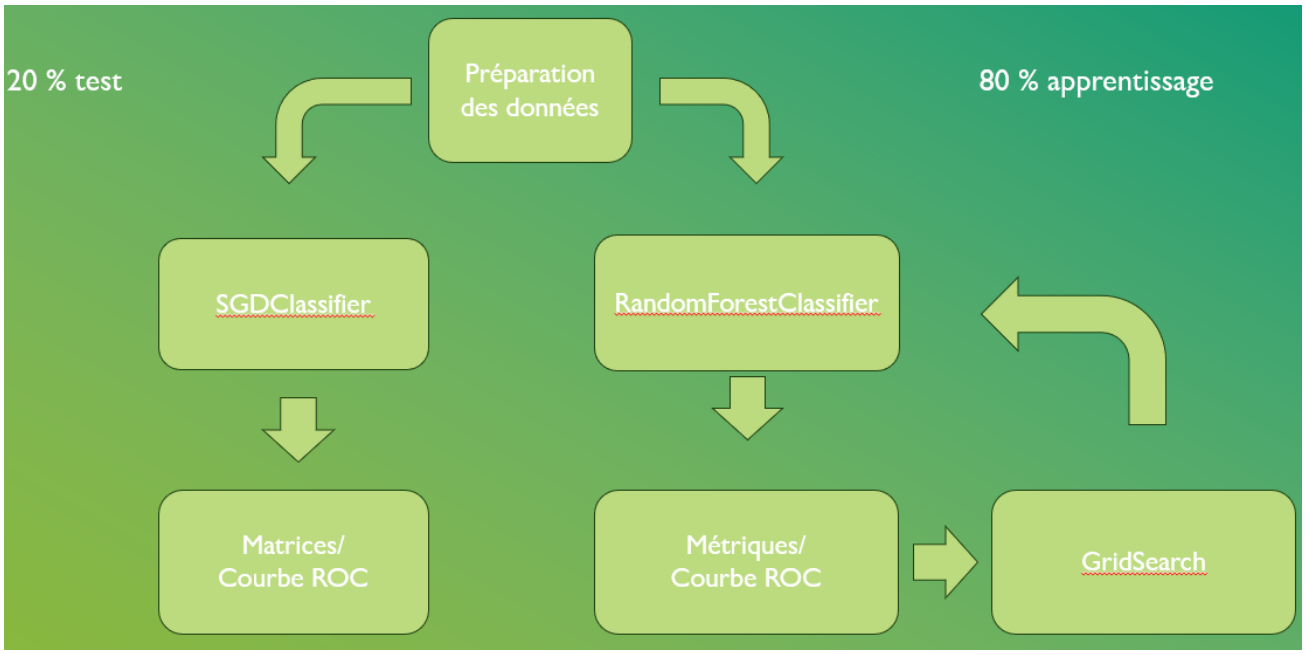


Schéma résumé fonctionnalité 4 :