Noms : MARIE, DELAUNAY, LORRET-DESPRET

Prénoms : Corentin, Robin, Noé

Classe : A3

Titre du document : Projet - IA

Date et heure d'envoi : Vendredi 20 à 18h

Nombre de mots : 3063

☑ En adressant ce document à l'enseignant, je certifie que ce travail est le mien et que j'ai pris connaissance des règles relatives au référencement et au plagiat.

Sommaire

Gantt de la semaine

Diagramme de Gantt - Projet IA

Liste des tâches à faire - Dates clés du projet - Répartition du travail

Membres de l'équipe (avec couleur attribué) :	Corentin	Noé (Scrum master)	Robin

Début du projet : 16/06/2025 — Fin du projet : 20/06/2025

TÂCHE	ATTRIBUÉE À	DÉBUT	FIN	Demi-journées 1	2	3	4	5	6	7	8
Phase 0 : Fondamentaux du projet	Équipe entière	0	1								
1. Lecture des besoins pour chaque fonctionnalité											
2. Attribution des tâches à chaque memebre											
3. Première version du diagramme de Gantt											
Phase 1 : Besoin du client 1	Corentin	1	7								
1. élaboration d'un premier code utilisant K-mean et un nombre de cluster déterminé a la main		1	2								
2. ajout d'une boucle pour trouver les combinaisons pour des clusters de 2 à 8		2	3								
2.1 ajout des premiers algorithmes de score (silhouette, coude) pour determiner le meilleur algorithme de clustering		3	4								
2.2 ajout d'un algorithme de clustering Agglomerative puis comparatif avec K-mean selon les scores		5	5								
3 Ajout des algorithmes de score (Davies, Calinsky) pour faire un "composite" pour avoir un score plus fiable		5	6								
4 Elaboration d'un 3eme algorithme DBSCAN qui fait du clustering selon la densité		6	8								
Phase 2 : Besoin du client 2	Robin	1	7								
1. Première version : Script de prédiction sur données brutes		1	2								
2. Première version : Constat et évaluation des premiers résultats		2	2								
3. Seconde version : Super-paramétrage à partir des données brutes		3	4								
4. Seconde version : Création de la pipeline et gestion du Preprocessing		4	4								
5. Seconde version : Sélection des modèles et gestion de l'entraînement		5	6								
6. Seconde version : Constat/Évaluation des résultats + Corrections		6	7								
7. Peaufinage du script final, commentaires et vérification des besoins du cahier des charges		7	7								
Phase 3 : Besoin du client 3	Noé	1	8								
1.1 Modifications de la dataframe		1	1								
1.2 Implémentation d'un premier modèle de régression		2	2								
1.3 Comparaison à d'autres modèles		3	4								
2.1 Recherches sur les RNN		4	7								
2.2 Implémentation du LSTM		4	5								
2.3 Création d'un script pour benchmark + générer les modèles		5	7								
3.1 Écriture du script final		7	8								
3.2 Finalisation		7	8								
Phase 3 : Fin de projet : Rapport + Soutenance	Équipe entière	7	8								
1. Vérification finale des rendus des scripts		7	7								
2. Préparation de la soutenance : Support visuel + Entraînements de passage à l'oral		7	8								
3. Rédaction du rapport + Relecture		7	8								

Introduction et mise en place du projet

Pour ce second “sprint” du projet d’A3, nous allons traiter de la partie IA. Il nous faut répondre aux besoins de 3 clients. Pour cela nous aurons besoin de préparer les données, entraîner des modèles d’apprentissages supervisés ou non supervisés (conformément aux besoins selon les clients), puis nous devrons évaluer, constater et rendre compte des résultats/métriques pour finalement réaliser un script de production qui satisfera les demandes.
L’ensemble de données sur lequel nous travaillons est une base de données AIS nettoyée préalablement avec un script R réalisé durant le projet de Big Data. Notre nettoyage fait de sorte que très peu de données soient supprimées. Nous utilisons Random Forest, nous entraînons le modèle pour prédire les valeurs NA initiales, mais aussi les NA imputés quand des données sont aberrantes ou erronées (mais pas NA).

Clustering maritime

Pour cette partie il faut créer une visualisation sur carte qui regroupe les navires selon des schémas de navigation similaires. Cette approche permet d'identifier des comportements typiques, de détecter des anomalies et d'optimiser les itinéraires maritimes.
Variables utilisées pour le clustering : LAT, LON qui est position géographique, SOG qui est la vitesse en nœuds, COG qui est la route en degrés, Heading qui est cap en degré.

Algorithmes de Clustering Testés

K-Means fonctionne selon un principe simple : il divise les données en k groupes en cherchant à minimiser la variance à l'intérieur de chaque groupe. L'algorithme commence par placer k points au hasard (les centroïdes), puis attribue chaque navire au centroïde le plus proche. Ensuite, il recalcule la position des centroïdes comme étant la moyenne des points qui leur sont attribués … Et recommence jusqu'à ce que les centroïdes ne bougent plus.

C'est rapide, mais il faut connaître à l'avance le nombre de clusters qu'on veut. Donc comme on le verra dans la suite du rapport, on testera K-Means pour chaque nombre de cluster :

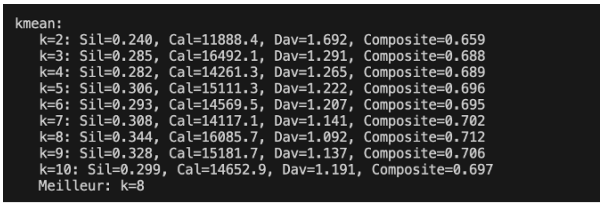


Figure 1 Visuel de l'algorithme de K-Means pour 2 à 10 cluster (les informations des scores de silhouette, Calinski-Harabasz, Davies-Bouldin et Composite seront expliqué dans la suite du rapport)

L'agglomératif fonctionne dans l'autre sens : au début, chaque navire forme son propre petit groupe. Puis on fusionne progressivement les groupes les plus proches entre eux, jusqu'à avoir le nombre de clusters voulu.
Cette approche m'a rappelé un projet récent sur la CAH que j'ai présenté lors d'une soutenance il y a quelques semaines. Le principe reste le même : on part de n clusters (un par point) et on descend progressivement vers k clusters en fusionnant à chaque étape les deux groupes les plus similaires. La matrice des distances est recalculée après chaque fusion pour tenir compte de la nouvelle configuration.
L'avantage ? On n'a pas besoin de décider du nombre de groupes à l'avance, et ça peut détecter des formes bizarres. Le dendrogramme (arbre hiérarchique) nous donne même une vision complète de toutes les étapes de regroupement. Par contre, c'est beaucoup plus lent sur de gros jeux de données - c'est pourquoi on a dû réduire notre échantillon à 5000 points pour éviter les problèmes de mémoire, contrairement au projet précédent où on travaillait sur des données plus petites.

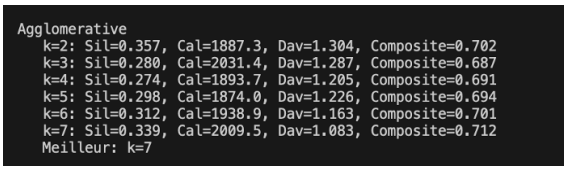


Figure 2 visuel de l'algorithme de l'Agglomerative pour 2 à 10 cluster (les informations des scores de silhouette, Calinski-Harabasz, Davies-Bouldin et Composite seront expliqué dans la suite du rapport)

DBSCAN a une approche complètement différente : il regarde la densité. Si un point a suffisamment de voisins dans un certain rayon, il devient un “point central" et forme un cluster avec ses voisins. Les points isolés sont considérés comme du bruit.
C'est super pour détecter des groupes de formes bizarres et il trouve automatiquement le nombre de clusters... mais il faut bien régler les paramètres eps (le rayon) et min_samples (nombre minimum de voisins). Dans notre cas, on a testé pour un rayon de 0.5, 0.7, 1 et pour un nombre minimum de voisin de 10 et 15.

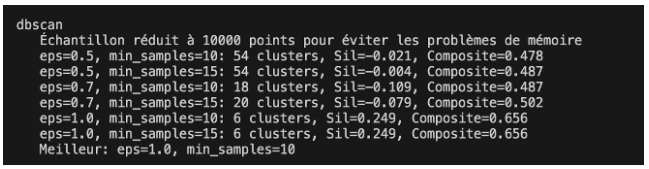


Figure 3 Visuel de l'algorithme de DBSCAN pour 2 à 10 cluster (les informations des scores de silhouette, Calinski-Harabasz, Davies-Bouldin et Composite seront expliqué dans la suite du rapport)

Métriques d'Évaluation

La silhouette mesure à quel point chaque point est bien dans son cluster plutôt que dans un autre. Ça va de -1 à 1 : plus c'est proche de 1, mieux c'est.
Alors que l’index de Clinski-Harabasz compare la dispersion entre les clusters et l'intérieur des clusters. Plus le score est élevé, mieux les groupes sont séparés.
Et Davies-Bouldin regarde la similarité entre chaque cluster et son voisin le plus proche. Ici, plus c'est bas, mieux c'est.

Score Composite

On a créé un score qui combine les trois métriques précédentes avec la formule : ***0.4 * silhouette + 0.3 * calinski + 0.3 * davies***.
On donne plus de poids à la silhouette car elle est plus facile à interpréter.

Méthode du Coude (Elbow Method)

Spécifique à K-Means, cette méthode regarde comment l'inertie (la somme des distances au carré) diminue quand on augmente le nombre de clusters. Le « coude » sur la courbe indique souvent un bon compromis. Malheureusement dans notre cas on ne trouve pas un coude très visuel, donc pour ce jeu de donné, on a décidé de prendre le score du composite en compte.

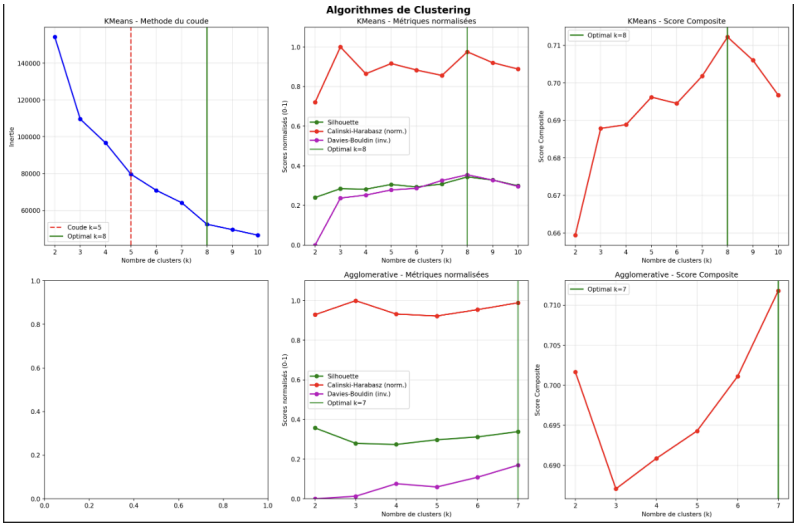


Figure 4 visuel des résultats finaux des algorithmes de clustering

Résultats Expérimentaux

K-Means	Algorithme Agglomératif
Nos tests montrent que K-Means atteint son meilleur score composite avec k=8 (0.712). Cependant, la méthode du coude suggère k=5 comme bon compromis. Mais comme dis plus tôt, on décide de prendre le score composite comme score car le coude n'est pas très visuellement représentatif.	L'agglomératif montre des résultats avec un pic à k=7, comme K-Means avec un score de 0.712 pour la Composite.
Détail des résultats : *- Silhouette optimale : k=8 (0.344)* *- Calinski-Harabasz optimal : k=3 (16492.1)* *- Davies-Bouldin optimal : k=8 (1.092)* *- Score composite optimal : k=8 (0.712)* *- Méthode du coude : k=5*	*Résultats détaillés :* *- Silhouette optimale : k=2 (0.357)* *- Calinski-Harabasz optimal : k=3 (2031.4)* *- Davies-Bouldin optimal : k=7 (1.083)* *- Score composite optimal : k=7 (0.712)*

DBSCAN a eu plus de mal sur notre dataset. Avec eps=0.5 ou 0.7, on obtient des scores de silhouette négatifs, ce qui indique une sur-segmentation (trop de petits clusters). Le meilleur résultat vient avec eps=1.0 qui donne 6 clusters et un score de 0.656.

Comparaison et Choix Final

Synthèse des Résultats

Au final, K-Means et l'agglomérative sont à égalité avec 0.712, tandis que DBSCAN reste à 0.656.

Répartition des Clusters

Voici comment se répartissent nos 419,085 navires dans les 7 clusters :

Cette répartition montre qu'il y a effectivement des patterns de navigation distincts, avec un comportement largement majoritaire et plusieurs comportements plus spécifiques.

```
Cluster 0: 45,587 navires (10.9%)
Cluster 1: 41,191 navires (9.8%)
Cluster 2: 71,524 navires (17.1%)
Cluster 3: 124,602 navires (29.7%)
Cluster 4: 25,614 navires (6.1%)
Cluster 5: 35,372 navires (8.4%)
Cluster 6: 75,195 navires (17.9%)
```

Visualisation et Script Final

Carte Interactive

Le programme génère une carte HTML interactive avec Plotly qui permet de voir géographiquement où se trouvent les différents clusters. Chaque groupe a sa couleur, ce qui rend l'analyse visuelle plus facile.

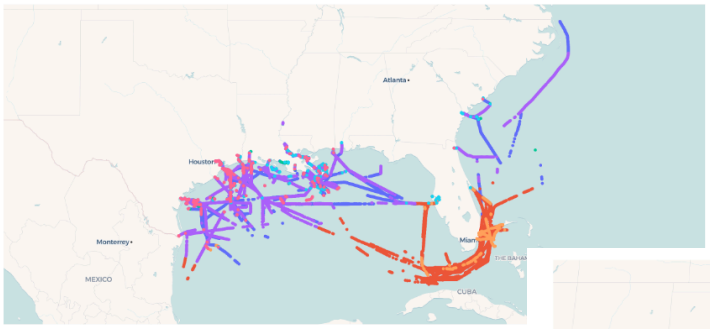


Figure 5 Affichage de la carte avec les 400 000 points

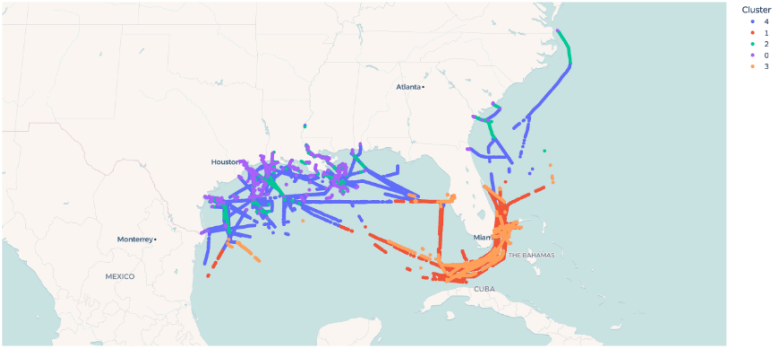


Figure 6 Voilà le résultat pour la méthode du coude de K-Means

Script de Prédiction

Le script final charge le modèle pré-entraîné et peut prédire le cluster d'un nouveau navire :
« *python3 besoin_client_1.py --lat 29.18 --lon -89.30 --sog 13.4 --cog 227.6 --heading 227* »
C'est important que le script ne refasse pas tout l'apprentissage à chaque fois donc il charge juste le modèle sauvegardé, ce qui est beaucoup plus rapide. A noter que le client doit lire le README.txt pour savoir en profondeur comment marche le script.

Prédiction du type de navire

Comme le titre l'indique, le besoin du client 2 est de prédire le VesselType en fonction des autres données disponibles. Nous a été demandé de suivre 4 principales étapes : préparation des données, apprentissage supervisé pour la classification, métrique pour la classification et préparation d'un script Python.

Première version de l'implémentation de la solution

Pour répondre au besoin de ce client, nous avons premièrement implémenter une première version de la solution. La solution suit précisément le guide donné par le cahier des charges.

Ainsi, en reprenant les demandes du cahier des charges, voici ce que nous avons mis en place et qui était demandé :

1. Préparation des données :
    - Sélection des colonnes (variables explicatives) pertinentes de la base de données en l'état : Cette partie a été étudiée et justifiée lors de la précédente semaine du projet Big Data. Nous avons établi qu'il était intéressant de supprimer d'une part les colonnes qui n'ont pas leur place dans un modèle d'apprentissage car elles ne sont pas explicatives : "id", "VesselName","IMO","CallSign". D'autre part, nous avons fait le choix (pour l'instant) de ne pas se servir des colonnes de positions géographiques (LON, LAT).
    - Encodage grâce à *LabelEncoder* et normalisation avec *StandardScaler* de la librairie *scikit-learn*.
2. Apprentissage supervisé pour la classification :
    - Choix de l'algorithme d'apprentissage : Le choix du meilleur modèle s'est fait par le test de 3 algorithmes que nous avons initialement testé car ce sont les modèles que nous avons estimé les plus pertinents pour la situation en lisant la documentation qui nous a été mis à disposition (RandomForest, SVM (SVN) et K-NearestNeighbor). Il en a

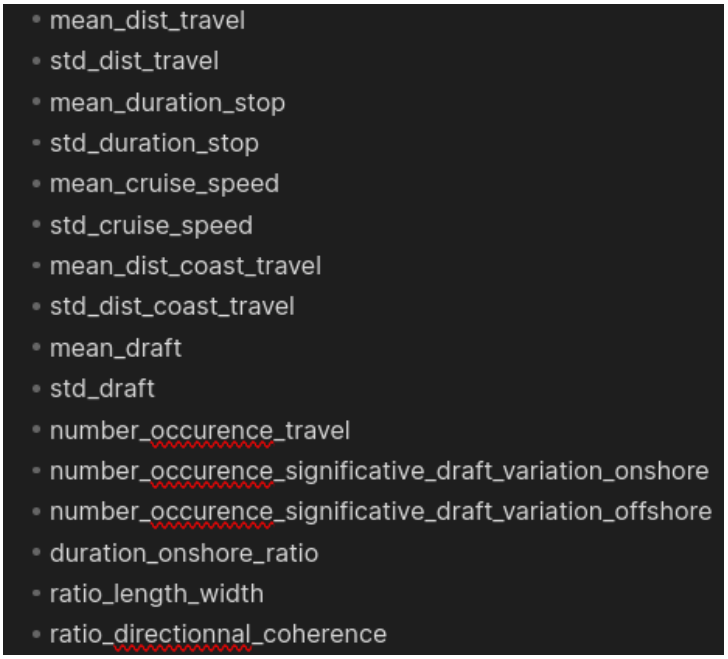
découlé que la meilleure précision a été atteinte avec RandomForest pour cette première version (86%).
    - Nous notons que pendant la séparation des données train/test, nous avons fait en sorte de grouper les MMSI, cela permettant d'éviter de retrouver les mêmes bâteaux dans les deux échantillons (certaines variables explicatives ne varie pas d'une ligne à l'autre si c'est le même bâteau -> cela pourrait faussement entraîné les modèles à "apprendre" les bateaux et avoir une précision largement surestimée).
3. Métriques pour la classification :
    - Evaluation des résultats de la classification : Nous nous sommes servi des fonctions *accuracy_score()* et *precision_recall_fscore_support()* qui nous ont donné plusieurs métriques de comparaison entre les modèles (précision sur plusieurs spectrums)
    - Nous avons également utilisé GridSearchCV pour rechercher les combinaisons optimales pour chacun des modèles testés (hyper paramètres des modèles).
4. Préparation d'un script :
    - Grâce à la librairie *args* avec ses différentes méthodes *parser()*, *add_argument()*, *parse_args()*, etc. Nous avons pu ajouter la fonction demandée pour prendre en entrée soit le MMSI d'un bateau existant dans la base de données pour prédire avec le meilleur modèle son VesselType.

Seconde version de l'implémentation de la solution

Ayant rapidement abouti à cette première réflexion, nous avons cherché à gagner encore plus de précision dans la prédiction et à améliorer notre processus d'entraînement. La réflexion qui a débouché sur cette seconde implémentation était la suivante : Nous avons une base de données séquentielle en fonction du temps, nous ne pouvons pas utiliser toutes les variables comme des singularités sur lesquelles entraîner un modèle qui soit "logiquement" performant.
Nous avions donc à "donner vie" aux variables suivantes : LON, LAT, SOG, COG, Heading, Draft (car elles varient toutes en fonction du moment du voyage d'un navire). La solution a été donc de transformer le tableau comme nous le connaissions de base, en ensemble de segments (les trajets des navires) ordonnées par MMSI et dans l'autre chronologique par lequel le bateau en question à réaliser les trajets identifiés.
Pour "segmenter", nous avons cherché à reconnaître un arrêt (SOG, Speed Over Ground, qui chute sous un seuil de 0.5 knots, de là nous avions simplement à retenir les timings et la position du début et de la fin d'un segment. Cette manière de structurer les données nous a également permis d'identifier : les durée des arrêts, les durée des trajets, le nombre d'occurrence de trajet par bâteau sur la durée de la base, la vitesse de croisière ("vrai vitesse") et même les distances des trajets.

Nous avons également pu obtenir la distance à la côte en utilisant la librairie *geopandas*. Elle nous permet de “savoir” via des objets *Shape* la présence de la terre ferme (de manière +/- précise). Cette capacité a découlé de calculer la distance moyenne à la côte des trajets (pour possiblement identifier des Passengers (VesselType entre 60-69) qui sont plus proches des littoraux en général).
Les implémentations précédents nous ont aussi permis de “déterminer” si un arrêt a été fait onshore ou offshore (avec l’établissement d’un certain seuil de distance à la côte), nous avons agrégé Draft (le tirant d’eau du navire à un temps t), plus le tirant est haut comparé à la normalité du bateau, plus il est “chargé” (plus de poids qui alourdit le bateau). Cela nous permet donc d’établir avec une bonne précision si un bateau s’est chargé/déchargé et même si cela c’est réalisé onshore ou offshore (en croisant avec la fonctionnalité précédente).

Tout ce “pré-travail”, nous permet ensuite aisément à calculer les “super-paramètres” que l’on voit dans la liste suivante (avec un nom transparent sur ce qu’elle représente) et qui consisteront la nouveau base de donnée d’entraînement (une ligne par bateau -> réduisant drastiquement par rapport à la taille de la base initiale):



A partir de là, nous avons suivi à peu près le même programme qu'initialement pour obtenir un algorithme de prédiction à quelques différences près :

- Nous avons choisi de mettre en place une normalisation supplémentaire avec RobustScaler après constat d’outliers coriaces.
- Nous avons utilisé SMOTE pour palier à des problèmes importants de déséquilibre de classe.
- Nous avons choisi d’entraîner sur des nouveaux modèles qui semblent (après recherches dans la documentation) relativement pertinents : LightGBM, CatBoost(que nous n’avons pas réussi à run car très long et lourd), XGBoost et RandomForest(car c’était le meilleur de la précédente version).

- Nous avons mis en place une couche pour cross-valider en stratification permettant d'améliorer la représentativité des données et la pertinence du calcul de précision.

Finalement, avec cette nouvelle version de prédiction nous atteignons des performances de prédiction d'environ 88%. Cela reste meilleur que la version précédente mais d'assez peu, mais étant donné les délais nous n'avons pas abouti à mieux pour l'instant mais cette seconde version aurait pu largement être améliorée étant donné son potentiel (dû à la pertinence de l'explicativité des variables explicatives).

Script de Prédiction

Le script final peut être exécuté 3 façons, pour entraîner le modèle sur une base de données brutes et de features finales, pour prédire à partir d'un MMSI, pour prédire à partir d'une série de features qui sont aux normes pour passer le preprocessing. Pour utiliser les 3 paramétrages possibles pour exécuter le script, suivre les commandes suivantes en remplaçant par les variables voulues :

- python script_2.py - -train - -evaluate
- python script_2.py - -predict - -mmsi 205776000
- python script_2.py - -predict - -features 8.3, 2.4, 345, etc.

Prédiction de trajets de navire

Plusieurs approches à ce problème étaient possibles, nous avons choisi de prédire à partir d'un point, puis de prédire les points suivants afin de générer une trajectoire.

Transformation des données

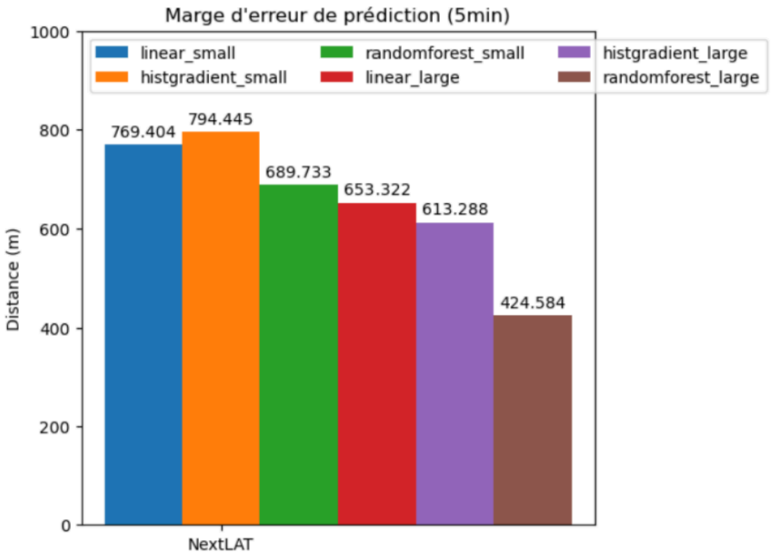
Afin de pouvoir prédire plusieurs points à la suite, notre modèle devra non seulement prédire une LAT et une LON, mais aussi COG, SOG, Heading… afin de pouvoir refaire une prédiction ensuite.
Nous avons donc modifié notre dataframe originale afin d'y ajouter les valeurs correspondant au point suivant du bateau sur chaque ligne.
Afin d'améliorer la précision du modèle, nous avons aussi appliqué quelques techniques de pré processing:
- La conversion des LAT/LON futures en delta au lieu de leurs valeurs réelles
- La conversion des angles (COG/Heading) en sin et cos
- L'application d'un MinMaxScaler pour mettre les données au même ordre
- Et la transformation des données catégoriques (VesselType) en matrice via get_dummies()

Évaluation des modèles

Afin d'évaluer la qualité de la prédiction d'un modèle, l'utilisation des R^2 n'est pas idéal, on va plutôt utiliser le mean_absolute_error, ce qui nous donne l'erreur absolue moyenne entre nos valeurs cible et nos valeurs prédites. Cependant ce chiffre est en LON/LAT, on le passe donc dans la formule de Haversine afin de le convertir en mètre. Cela nous donne une métrique plus compréhensible.

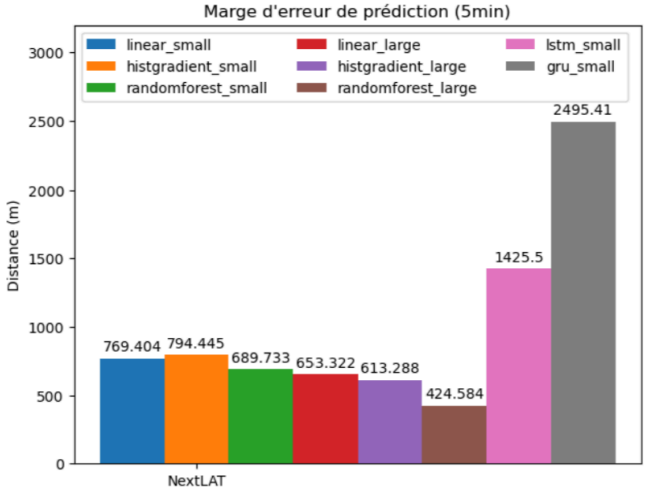
Modèles de régression

Tout d'abord nous avons tenté de prédire via une simple régression linéaire, et obtenu un résultat correct, environ 769m d'erreur à 5 minutes. Afin de trouver un meilleur résultat, nous avons aussi testé d'autres modèles: HistGradientBoosting nous donne environ 794m d'erreur et RandomForest (le meilleur) nous donne 689m d'erreur. Nous avions aussi à notre disposition un plus gros csv (15 fois plus gros), nous avons re-testé nos modèles et obtenu de meilleurs résultats.



Modèles neuronal

Au cours de la semaine, on nous a conseillé d'expérimenter avec un Réseau Neuronal Récursif de type Long Short Term Memory, ce que nous avons fait. Nous avons aussi découvert qu'une Unité Récurrente Fermée était particulièrement adaptée à ce problème. Nous avons donc implémenté ces deux modèles et comparé les résultats.



Malheureusement, ces modèles n'étaient pas aussi efficaces que nous le pensions, et par manque de temps (il s'agit d'un nouveau concept et les modèles étaient long à entraîner), nous n'avons pas pu parfaire les hyper paramètres des modèles, mais à la vu des résultats actuels, il est probable que ces modèles après quelques modifications puissent atteindre des précisions égales ou équivalentes à celle de RandomForest.

Script de prédiction

Le script final peut de prendre de deux manières différentes en entrée:
- Soit en json en argument de ligne de commande
- Soit un par un si aucun json n'est donné

« ./besoin_client_3.py --steps 15 --json='{"LAT": 26.13178, "LON": -92.04043, "SOG": 12.0, "COG": 12.0, "Heading": 12.0, "VesselType": 31}' »

Afin de ne pas avoir à entraîner un modèle à chaque fois, il charge un modèle pré-entraîné depuis son .pkl