NTNU

Kunnskap for en bedre verden

TDT4173 - Modern Machine Learning in practice

[146] The Italian Butei

Authors:

Member	**Student ID**	**Email**
Amadori Luca	133429	lucaam@stud.ntnu.no
Coppola Rodolfo Emanuele	133173	rodolfoc@stud.ntnu.no
Meschieri Andrea	133527	andremes@stud.ntnu.no

November, 2024

Table of Contents

List of Figures

List of Tables

1 Introduction

The aim of this project is, given AIS data from 1st January to 7th May 2024, the prediction of future positions of vessels at given timestamps for five days into the future. The work is based on a main dataset containing the positions of 689 vessels per time, which can possibly be integrated with additional information contained in ports, vessels and schedules datasets. We want to build a predictive model for the next positions of 216 of these vessels at their given timestamps.
Different models, features' configurations and data processing methods have been exploited. This report provides an overview of the preprocessing of raw datasets, the exploratory data analysis, the features selection process and the comparison among different models which we conducted in order to produce an effective prediction.

At the start of our work, we read and saved the three datasets *ais_train.csv*, *ports.csv* and *vessels.csv*. We sorted the main dataset by *vesselId* and *time*. Successively we integrated the information about vessels and ports, by merging these two additional datasets with the main one, respectively on *vesselId* and *portId*. In this way, we built a big dataset, called *train_preproc*, on which we conducted data analysis and cleaning. In particular, we conducted the construction of the prediction model in the following way: we used each observation from the *train_preproc* dataset to predict the position relative to the subsequent observation, grouping by vessels. By doing so, we left the last observation of each vessel out of the training dataset and used it to build the dataset for testing the model. The main idea behind our model is to use the last known position, along with the corresponding data for that observation (*cog*, *sog*, ...), to attempt to predict future positions.

2 EDA: Exploratory Data Analysis

In this section we dig into the understanding of our dataset, analyzing both the shape of the single features and the relations among them, besides the cleaning of the dataset itself.

2.1 Search Domain Knowledge

Automatic Identification System (AIS) data is primarily used for tracking the movements of vessels to ensure maritime safety, optimize shipping routes, and support port operations. AIS devices transmit positional and navigational information, allowing for near real-time monitoring of maritime traffic. The communications should occur about every 20 minutes. Still, as it's summarized in Table 1, we noticed that the observations were very irregular, meaning that there were significant variations in the intervals between recorded transmissions. While some observations aligned with the expected 20-minute interval, many others occurred with gaps of several hours or even days, while others were recorded much more frequently. This irregularity could be due to a number of factors, including:

- Signal Interference or Reception Limitations: AIS data is transmitted via VHF signals, which are subject to line-of-sight limitations. Obstacles such as terrain, weather conditions, or congestion in maritime traffic could disrupt regular transmissions, especially in remote areas or near busy ports.
- Vessel Behavior and Speed: Faster-moving vessels or those in high-traffic areas often transmit data more frequently, while stationary or slower vessels might have longer intervals between transmissions.
- Data Collection Variability: Differences in the sources of data collection (e.g., satellite-based vs. terrestrial AIS) could contribute to irregularities. Satellite AIS often has sparser coverage than terrestrial sources, especially in open oceans, resulting in less frequent updates for certain vessels.
- Device or Transmission Errors: Outliers in the dataset, such as extremely short or excessively long transmission intervals, may be due to errors in AIS equipment or network issues affecting transmission consistency.

Table 1: Descriptive Statistics for *time_horizon* expressed in minutes

Statistic	Value
Count	1,519,762
Mean	77.63
Std Dev	883.11
Min	0.033
25% (Q1)	19.07
Median (50%)	20.63
75% (Q3)	21.02
Max	98,722.92

Clearly, the median is much lower than the mean value. This behavior is due to the fact that the dataset contains very large outliers for the time intervals between observations. These outliers, representing unusually long gaps between transmissions, skew the mean to a higher value. This variability in observation intervals introduces challenges for modeling, as it can lead to inaccurate predictions if not addressed. We considered adding intermediate observations between time-distant rows, ensuring more consistent time series data for each vessel. To achieve this goal, we developed a portion of code to cycle through the rows of the dataset, grouped by each individual vesselId, and check for time gaps between observations. If a gap of 4 hours or more is detected, the code adds a number of observations proportional to the time gap, and each new observation is constructed through interpolation. This approach was theoretically supposed to bring benefits to our predictions. Unfortunately, based on the tests conducted, this did not happen, so we decided to proceed with our analysis without this attempt to repopulate the dataset. However, we still present the portion of code used below.

```

# Set the max threshold
time_threshold = pd.Timedelta(hours=4)

# List for storing data of each vessel
vessel_dfs = []
cont = 0

# Loop on each vessel_id to analyze its own data
for vessel_id, vessel_data in train_preproc.groupby('vesselId'):
    # Print for debug
    print(cont)
    cont+=1

    # Sort the observations by time to avoid errors
    vessel_data = vessel_data.sort_values('time').reset_index(drop=False
        )

    # List for iperpolated obs
    interpolated_rows = []

    # Loop on every couple of consecutive observations
    for i in range(1, len(vessel_data)):
        current_row = vessel_data.iloc[i - 1]
        next_row = vessel_data.iloc[i]

        # Time difference between consecutive obs
        time_diff = next_row['time'] - current_row['time']

        # Populate the interpolated rows list
        interpolated_rows.append(current_row)

        # If there is a time jump in the data
        if time_diff > time_threshold:
```

```python
            # Number of intervals to add
            num_new_rows = int(time_diff / time_threshold)
            time_interval = time_diff / (num_new_rows + 1)

            # Add new rows
            for j in range(1, num_new_rows + 1):
                # New timestamp
                interpolated_time = current_row['time'] + j * time_interval

                interpolated_row = current_row.copy()
                interpolated_row['time'] = interpolated_time

                # Interpolation of other numeric columns
                for col in ['latitude_x', 'longitude_x', 'cog', 'sog', 'rot']:
                    interpolated_row[col] = np.interp(
                        j / (num_new_rows + 1),
                        [0, 1],
                        [current_row[col], next_row[col]]
                    )

                # Add the new interpolated row to the list
                interpolated_rows.append(interpolated_row)

    interpolated_rows.append(vessel_data.iloc[-1])
    vessel_dfs.append(pd.DataFrame(interpolated_rows))

# Building of the final complete dataset
final_dataset = pd.concat(vessel_dfs, ignore_index=True)

# Sort again the dataset by time and vessel_id
final_dataset = final_dataset.sort_values(by=['vesselId', 'time']).reset_index(drop=True)
```

2.2 Cleaning up features

1. **Cog**: This variable indicates the actual path the vessel is following over the Earth's surface, measured in degrees from 0 to 360. However, as stated in the features' documentation, 360 value is inputed as default when no information is available. For this variable, we adopted two different approaches in the two short notebooks chosen for prediction. In the first one we just decided to drop the rows with value of COG equal to 360, losing less than 6000 rows on a total of more than 1.500.000 observations.
   In the second notebook, we used an Iterative Imputer to fill in the `NaN` values of the column *cog* (i.e. the ones with value 360), trying to preserve our dataset from leaks of information caused by the drop of invalid rows of *cog*. As a result, in both notebooks, we got a column with values lying in the interval [0,360).

2. **Sog**: This variable represents the speed over ground of the vessel. It is an important feature for prediction, as it explains if and how quickly the position is changing over time. We decided to keep values of *sog* lying in the interval [0,102.2], dropping the other rows. By doing so, our dataset lost only 393 observations.

3. **Rot**: This indicates how quickly the vessel is changing its heading, measured in degrees per minute. As it's stated in the features' explanation document, values of rot outside of [-127,+127] means that no information is available. Thus we dropped the rows which did not respect this constraint.

4. **Heading**: This shows the direction in which the vessel's bow is pointing, measured in degrees from 0 to 360. We dropped the rows with values lying outside the domain of [0,359].

5. **Latitude** and **Longitude**: We verified that all observations were consistent with the definitions of these two physical quantities, that is [-90,90] for latitude and [-180,180] for longitude. None of the observations failed this type of check, so no cleaning was necessary from this perspective.

6. **Navstat**: This variable contains different codes from 0 to 15, representing the current navigation state of the vessel. After a brief research about the meaning of the codes (illustrated also in Table 4), we decided to drop rows with *navstat* value bigger or equal than 9, as the codes in [9,15] are not informative at all (e.g. not available or reserved for future use).

7. **etaRaw**: this variable provides the expected time of arrival at the destination in a raw format. It is set locally on the vessel and cannot be updated, for this reason we decided to drop this entire column because it was unreliable in our opinion.

8. **Vessels dataset**: This dataset contains a lot of information about the vessels, including their technical settings. However, many column are full of NaN values, and this complicates the extraction of useful and reliable data. Consequently, we decided to keep only three features of this dataset: *CEU* (Car Equivalent Unit), *GT* (Gross Tonnage) and *length*. These variables act for the dimensions of the vessel, in particular the first two are volume measures, while the third one, as its name says, stands for the length.

9. **Ports dataset**: We decided to include this dataset as it gives important information about the destination ports of the vessels. In particular, we included the features containing the longitude and the latitude of the ports in our analysis, as *longitudePort* and *latitudePort*.

2.3 Exploring individual features and their intuitiveness

At first, we wanted an overview of how the features were composed, thus we got their mean, standard deviation, minimum, maximum and 25%, 50%, 75% quantiles.

Table 2: Statistical description of features (Part 1)

	cog	sog	rot	heading	navstat
count	1.513006e+06	1.513006e+06	1.513006e+06	1.513006e+06	1.513006e+06
mean	1.775515e+02	6.327782e+00	3.777513e-02	1.751572e+02	2.077464e+00
std	1.071683e+02	7.385728e+00	1.586388e+01	1.055242e+02	2.394922e+00
min	0.000000e+00	0.000000e+00	-1.270000e+02	0.000000e+00	0.000000e+00
25%	7.800000e+01	0.000000e+00	0.000000e+00	7.500000e+01	0.000000e+00
50%	1.824000e+02	5.000000e-01	0.000000e+00	1.800000e+02	0.000000e+00
75%	2.677000e+02	1.420000e+01	0.000000e+00	2.640000e+02	5.000000e+00
max	3.599000e+02	1.022000e+02	1.270000e+02	3.590000e+02	8.000000e+00

Table 3: Statistical description of features (Part 2)

	latitudePort	longitudePort	CEU	GT	length
count	1.513006e+06	1.513006e+06	1.513006e+06	1.513006e+06	1.513006e+06
mean	3.640340e+01	1.227537e+01	4.164179e+03	4.706690e+04	1.918902e+02
std	2.299371e+01	6.910138e+01	2.528528e+03	1.894055e+04	2.981348e+01
min	-4.546635e+01	-1.733000e+02	0.000000e+00	8.659000e+03	9.990000e+01
25%	3.419194e+01	-4.783000e+00	1.600000e+03	3.331300e+04	1.799900e+02
50%	4.224250e+01	4.828333e+00	4.872000e+03	4.763500e+04	1.990000e+02
75%	5.133639e+01	2.251700e+01	6.400000e+03	5.995200e+04	1.999900e+02
max	6.993300e+01	1.784261e+02	8.500000e+03	1.004300e+05	2.960000e+02

Successively, we went deeper in trying to understand the distribution of the features originally in AIS dataset, namely *cog*, *sog*, *rot*, *heading* and *navstat*.

1. **Cog**: The frequencies are relatively stable across most *cog* values, suggesting a fairly uniform distribution with some minor peaks and dips. There are noticeable spikes around 0 degrees and 359, which are basically the same angle, indicating that many observations have *cog* values close to north direction. We decided to show also the density of *cog* angles in polar coordinates, to better understand the dynamics inside this feature.

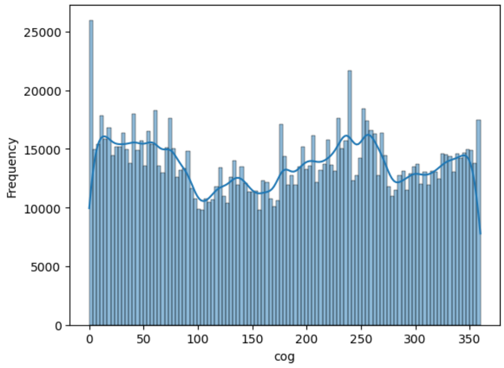

Figure 1: *Cog* feature distribution

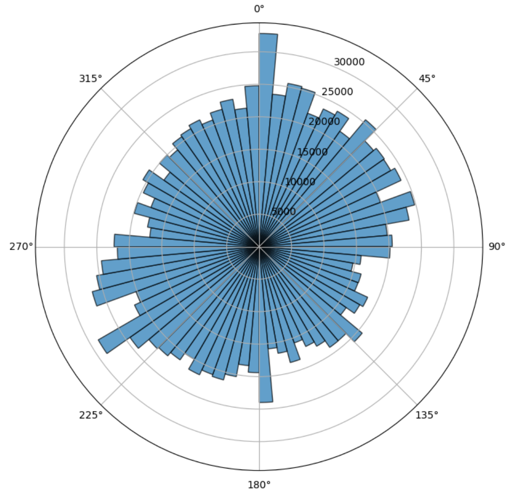

Figure 2: *Cog* feature polar distribution

2. **Sog**: The boxplot of this feature clearly shows that a lot of observations were taken with stationary vessels, so with *sog* values equal, or very close to 0. It's possible to notice also that most of the vessels travelled at low speed. It's possible to notice the presence of outliers as well, with *sog* which reaches values even bigger than 100 knots. It's very unusual for merchant ships to reach very high speed like 100 knots, but since some smaller ships or ferries actually can travel with that conditions, considering also the effects of wind and sea current, we decided to keep these outliers in our dataset.

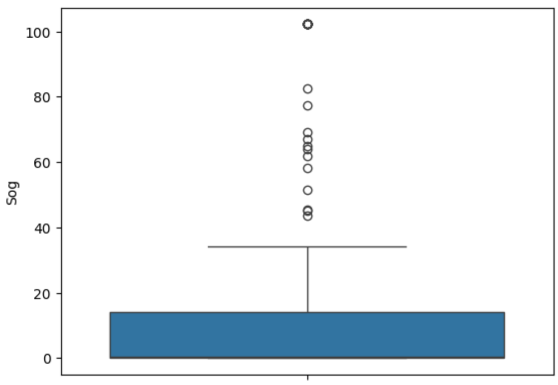

Figure 3: *Sog* feature boxplot

3. **Rot**: The distribution of *rot* values in the dataset is centered around zero, exhibiting very thin tails. This indicates that the majority of observations are likely to be very close to zero. This phenomenon arises from the dataset containing numerous observations taken at close intervals, which leads to consistently small *rot* values across most entries.

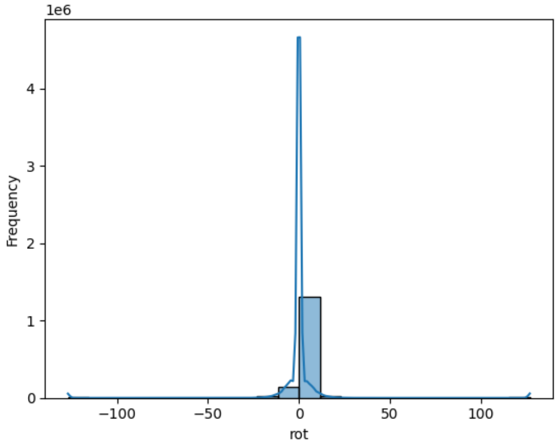

Figure 4: *Rot* feature distribution

4. **Heading**: As *cog* feature, *heading* values are quite uniform on the domain as well, with some peaks in preferred directions of north-west and north-east.

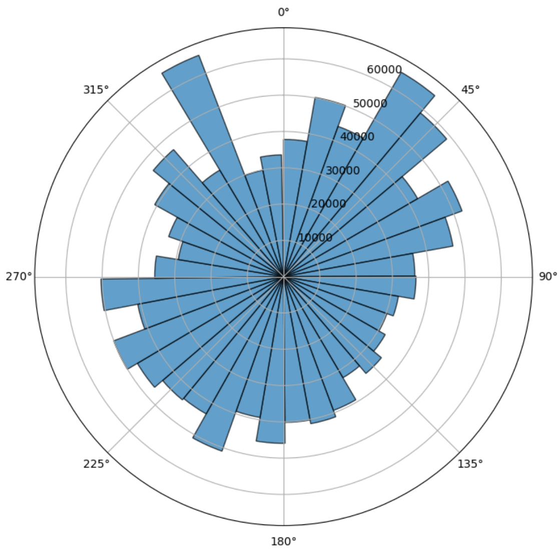

Figure 5: *Heading* feature polar distribution

5. **Navstat**: The navigation status observations in this dataset are predominantly concentrated in three categories: 0 (underway using engine), 1 (anchored), and 5 (moored). The accompanying histogram clearly illustrates that the data is evenly split between moving vessels and those that are stationary.

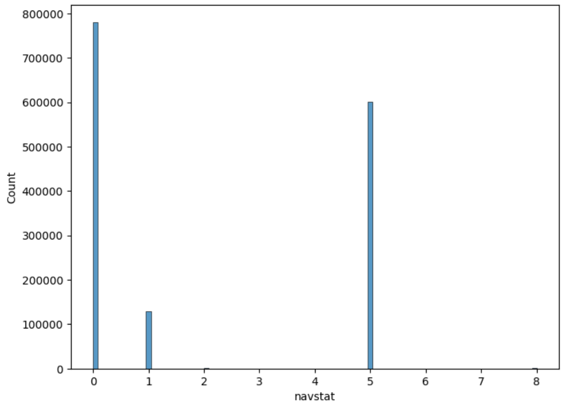

Figure 6: *Navstat* feature distribution

Then, we have analyzed the PACF (Partial Autocorrelation Function) of the latitude and longitude columns with 100 lags for the ships for which we needed to make predictions. For each of these ships, we calculated the three most significant lags and arranged them in order of lag. We obtained the following results:

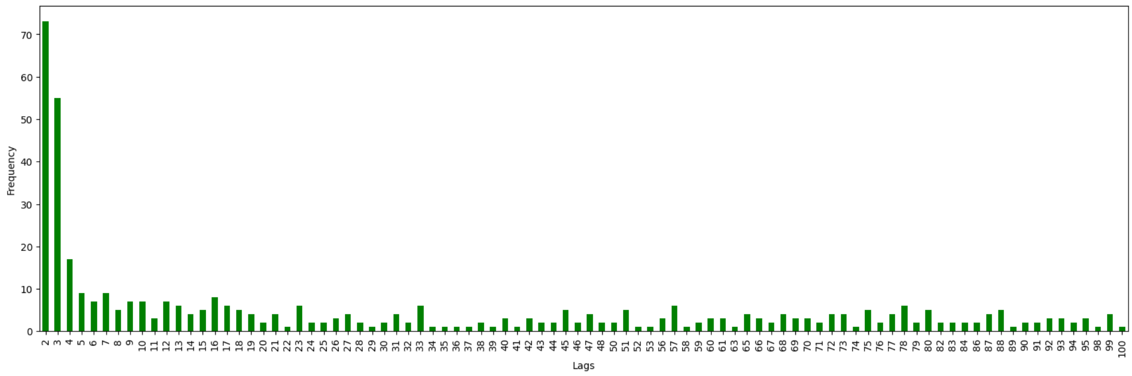

Figure 7: Latitude most significative past own lags

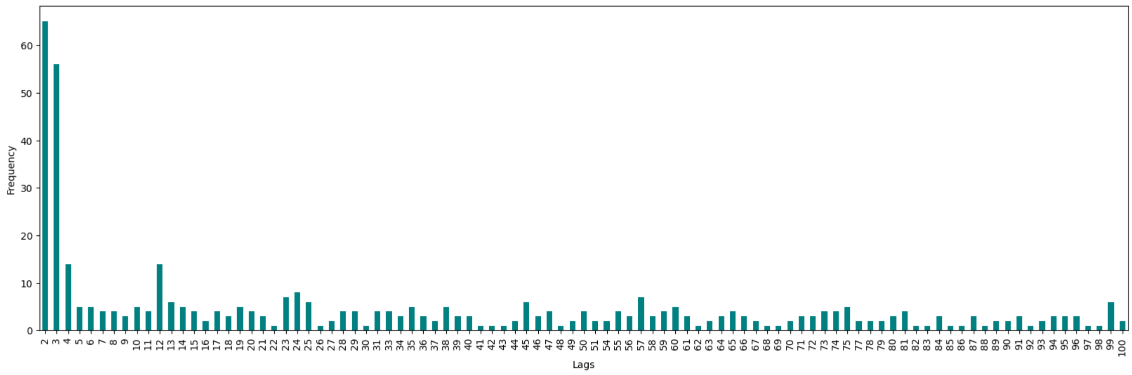

Figure 8: Longitude most significative past own lags

The plots reveal that, for more than 70 ships, lag 2 in their latitude time series consistently ranks among the top three most significant lags. Lag 3 is also notable, ranking third in frequency after lags 1 and 2, with approximately 55 ships showing it as a significant lag. This trend holds for the longitude time series as well, although lag 2 is slightly less prevalent. This pattern likely reflects the recurring presence of temporally close observations.

2.4 Exploring pairs and groups of features

In order to understand possible relations between our features and the target variables, we performed an analysis on their correlation matrix.

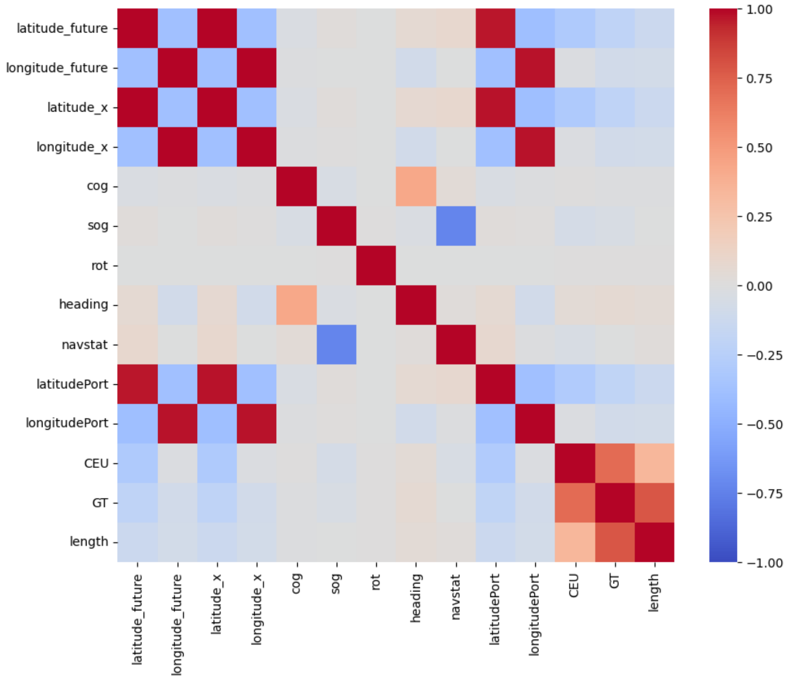

Figure 9: Correlation matrix of features

It can be observed that future latitude (*latitude_future*) is highly correlated with the last known latitude of the vessel (*latitude_x*) and the latitude of the destination port (*latitudePort*). The same holds for longitude. Moreover, as expected, *cog* and *heading* are quite correlated (ca. 0.5), since they express very similar information. This kind of reasoning applies to *navstat* and *sog* as well: they are negatively correlated. When the speed over ground is bigger than 0, it means that the vessel is moving, leading to a *navstat* value equal to 0. On the contrary, if *sog* is null, the vessel is anchored or moored (2 or 5 as values of navigation status). It's possible to notice that the variable *rot* is uncorrelated with all other features, suggesting its potential importance in predictive models, as it conveys unique information not captured by any other feature.
Lastly, the correlation matrix shows that *CEU*, *GT* and *length* variables are highly correlated with each other, as all of them represent the vessel dimensions in different ways. We used this important insight in order to develop a new variable which sums up these three variables, as explained in 3.1.2

The relationship between *cog* and *heading* is clearly illustrated in the following graph, which highlights the existing pattern between the two.

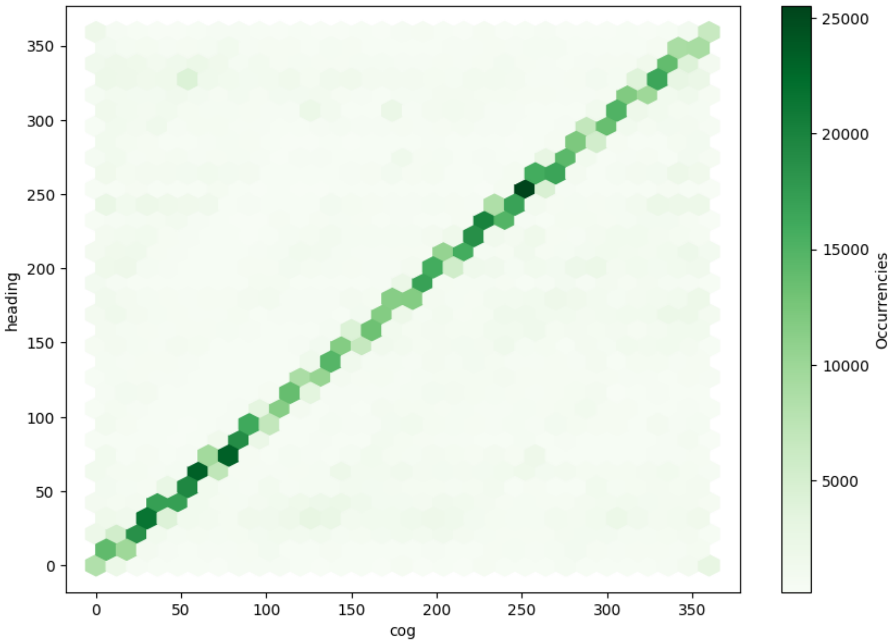

Figure 10: Hexbin plot of *cog* and *heading*

From the following scatterplot, it's possible to see a quite linear relationship between *CEU* and *GT*, as they both represent the volume of the ship: the first measures the cargo capacity, while the second one expresses the total volume of the vessel. Obviously there is also a relation with *length*: ships with a low *GT* tend to be shorter and vice versa.

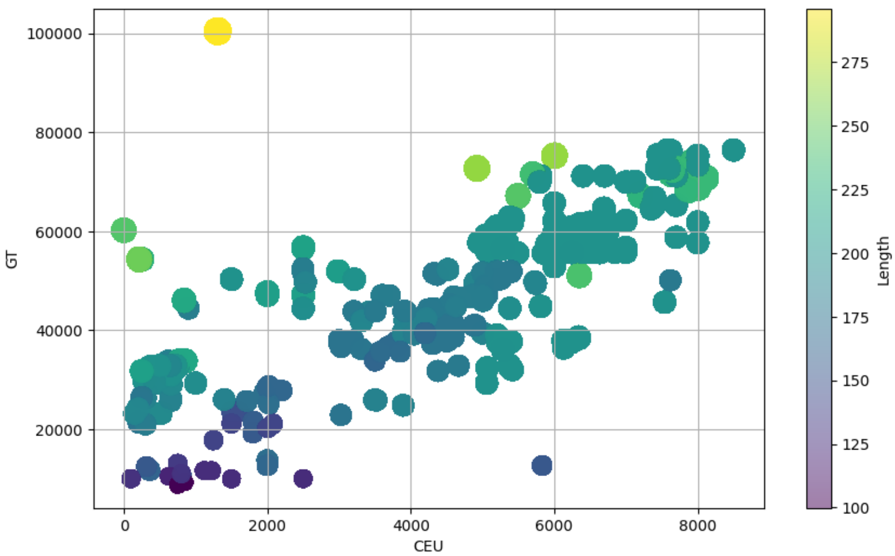

Figure 11: Scatterplot of *CEU*, *GT* and *length*

3 Feature Engineering

Talking about feature engineering, we decided to perform feature generation in order to enhance the model's predictive ability. In this section, two modeling configurations are treated: the first included in the short notebook 1 and the second implemented in the short notebook 2. In particular, the first model is basically an auto-regressive one, thus is based on past lags of latitude and longitude. The second one does not have auto-regressive components, but has more features transformed by us.

3.1 Feature Extraction

3.1.1 Short notebook 1 - Iterative Random Forest

In the first notebook, we introduced lagged values of each observation of the latitude and the longitude, shifted one, fifty, and one hundred observations back, respectively. The choice of 50 and 100 is due to the fact that they are the average number of observations that correspond to a lag of 2.5 and 5 days, respectively, as in the test set we need to produce predictions up to 5 days in the future. This technique can capture patterns and potential trends in the data over different periods. In particular, a coordinate lagged by 1 observation helps capture short-term effects or immediate changes in the data, one lagged by 50 observation provides a mid-range view and can capture cyclical or periodic patterns that may not be immediately obvious with shorter lags and a value lagged by 100 observations helps identify longer-term trends or seasonal effects, providing insight into how a variable might behave over a larger time frame. We also added the variables *latitude_ma10* and *longitude_ma10*, that represent the moving average computed on the last 10 observations of latitude and longitude, respectively. The introduction of this features allows the model to keep track of the information contained in the most recent observations, which are very informative (as illustrated in Figure 7 and Figure 8) without introducing too many new covariates. Furthermore, we added a new feature called *time_horizon*, that contains the difference in seconds between the time instants of the next and the current observation, serving as an indication of the prediction horizon.
Moreover, we introduced the variable *distance_to_port* obtained using the function `haversine`, that computes the distance between two points on Earth, approximating the globe as a sphere. This variable represents the distance between the last known observation of a vessel and the port to which that vessel is directed to.
Finally, in order to add more information about the destination port and the route followed by the ships, we developed a feature called *bearing_to_port*: it represents the angle (in degrees) the ship would need to turn from its current position to face directly towards the port location. The angle is positive for bearings to the east and negative for bearings to the west, relative to true north.

```python
def haversine(lat1, lon1, lat2, lon2):
    # Average Earth radius expressed in km
    R = 6371.0

    # Convert coordinates to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Differences between coordinates
    dlat = lat2 - lat1
    dlon = lon2 - lon1

    # Haversine formula
    a = np.sin(dlat / 2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(
        dlon / 2.0)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

    # Final distance
    distance = R * c
    return distance
```

Listing 1: `haversine` function

3.1.2 Short notebook 2

Talking about feature generation of the second notebook, we introduced a dummy variable called *navstat_dummy* with value 1 if the vessel is moving at a certain timestamp and 0 otherwise. The reason for this addition is that we wanted to transform the information contained in the different

navstat conditions into the practical outcome of each, namely, whether the vessel is moving or not. In particular, as we dropped observations with *nav_stat* bigger than 8, we mapped the statuses 0 and 8 into 1 (moving), while the other values, i.e. 1 to 7, into 0 (stopped or moving very slowly).
Furthermore, we added *time_horizon*, computed in the same way as in the first notebook. In addition, since in Subsection 2.4 we highlighted a strong correlation between *CEU, GT* and *length*, we decided to summarize the information contained in these three variables into a new feature, called *vessel_dimensions*, obtained with a PCA. In these way we reduced a bit the dimensionality of the problem, preserving more than 99% of the total information (variance) contained in the three covariates, and we lowered the risk of overfitting due to redundant splits, that sometimes happens in presence of strongly correlated features.

Code	Status description
0	Under way using engine
1	At anchor
2	Not under command
3	Restricted manoeuverability
4	Constrained by her draught
5	Moored
6	Aground
7	Engaged in Fishing
8	Under way sailing
9	Reserved for future amendment of Navigational Status for HSC
10	Reserved for future amendment of Navigational Status for WIG
11	Reserved for future use
12	Reserved for future use
13	Reserved for future use
14	AIS-SART is active
15	Not defined (default)

Table 4: Navstat AIS documentation

3.2 Unused variables

Additionally, along the project, we performed more feature transformation that we did not include in our two final models, due to not enough predictive capability highlighted in the public tests.

- Splitting the *cog* feature: Since *cog* (Course Over Ground) is measured in degrees, we transformed it into its sine and cosine components, resulting in *cog_sin* and *cog_cos*. This approach ensures that directions of 0° and 359.9°, which represent nearly identical angles, have consistent feature values in both components, preserving the continuity of the directional data.

- Combination of *sog* and *navstat_dummy*: as *sog* and *navstat* features are highly negatively correlated, as showed by Figure 9, we tried to merge them into an unique feature: *speed_new*. This was generated multiplying *sog* and *navstat_dummy* values, so it resulted in 0 if the ship was stopped, and in the actual sog value if the vessel was moving instead.

- Merging *heading* and *cog*: we created a new variable called *cog_heading_diff* as the difference of this two angles, trying to avoid the usage of two correlated variables. This feature represents the route deviation caused by winds and sea currents.

4 Different predictors

The aim of the project was the achievement of an efficient prevision of the future positions of a selection of vessels. In order to do so, we exploited 4 different predictors, comparing them on the basis of their performances on the public test set. We present them in the following subsections.

4.1 Elastic net

Our first attempt was the development of a simple model as the elastic net. Despite the big advantage of being very fast on the CPU, this kind of model cannot identify non-linear relationships inside the data.
The first combination of features just contained the previous position (that is *latitude_x* and *longitude_x*), *cog*, *sog*, *rot*, *heading*, *nav_stat* and *time_horizon*. The parameter $alpha = 0.1$ controls the overall strength of regularization. An higher alpha enforces more penalty on the model's complexity, which can prevent overfitting but may reduce model accuracy. Then we chose $l1\_ratio = 0.5$, which balances evenly between L1 (Lasso) and L2 (Ridge) regularization.

```python
distinct_Id = data_test['vesselId'].unique()

#Loop on distinct vesselId
for id in distinct_Id:
    subdatasets[id]['time_horizon'] = -subdatasets[id]['time_numeric'] +
         subdatasets[id]['time_numeric'].shift(-1)

    # Target
    Y = subdatasets[id][['latitude_x', 'longitude_x']].iloc[1:]
    # Train set
    X = subdatasets[id][['latitude_x', 'longitude_x', 'cog', 'sog', 'rot
        ', 'heading', 'navstat', 'time_horizon']].iloc[:-1]

    # Elastic Net
    Model = ElasticNet(alpha=0.1, l1_ratio=0.5)
    Model.fit(X, Y)

    target_time = subdatasets[id]['time_numeric'].iloc[-1]
    time_horizon = test_subdatasets[id]['time_numeric'] - target_time

    cont = 0
    array_pred = np.zeros((len(time_horizon), 2))

    # Loop on values in time_horizon
    for t in time_horizon.tolist():
        subdatasets[id].at[subdatasets[id].index[-1], 'time_horizon'] =
            t

        feature_names = ['latitude_x', 'longitude_x', 'cog', 'sog', 'rot
            ', 'heading', 'navstat', 'time_horizon']
        X_test = pd.DataFrame([subdatasets[id][feature_names].iloc[-1].
            values], columns=feature_names)

        # Prediction on X_test
        Y_pred = Model.predict(X_test)
        array_pred[cont][0] = Y_pred[0][0]
        array_pred[cont][1] = Y_pred[0][1]
        cont += 1

    test_subdatasets[id][['latitude_predicted', 'longitude_predicted']]
       = array_pred
```

Public Score
177.16

At the very first stages of the project, we built our prediction on the development of elastic nets, one for each *vesselId* to be predicted. However, we noticed that this kind of approach was not very effective: we were losing all the information contained in the dataset that was not related to the ships contained in the test set. Thus, as it's possible to see in the next section, we changed the main idea of our code.

4.2 Random Forest

Here, we built a unique model to be trained on the entire dataset. Moreover, we decided to resort to more complex models, as an ensemble of decisional trees, like Random Forest, is. We worked a lot on finding the best combination of features and hyper-parameters for this model, also recurring to feature engineering, as illustrated in Section 3. We noticed that introducing a scaler in order to normalize the features in the train set, really improved the performances of our model. Moreover, we conducted hyper-parameter tuning for our Random Forest Regressor using Random Search to optimize model performance while minimizing computational time. Random Search allowed us to explore a wide range of hyper-parameter values without the exhaustive computation required by grid search. By randomly sampling combinations of parameters, we were able to identify the best-performing model configuration more efficiently. The tuned hyper-parameters are the one used in the code showed above. In this new setting we added three new features with respect to the previous model: *latitudePort*, *longitudePort* and *vessel_dimensions*. As the Exploratory Data Analysis carried out in Section 2.4 suggested, the position of the destination port is highly correlated to the present position, thus it might have big importance for prediction. Additionally, we tried to include the information about the dimensions of each vessel in the prediction model, by mixing the variables of *length*, *CEU* and *GT*.
In this model configuration we applied post-processing: we noticed that many predicted position were on earth, thus we recurred to the provided land and ocean maps to move those predictions to the closest points on the sea. This kind of method improved our public score of ca. 0.8 points, but it might produce even better improvements on private test set.

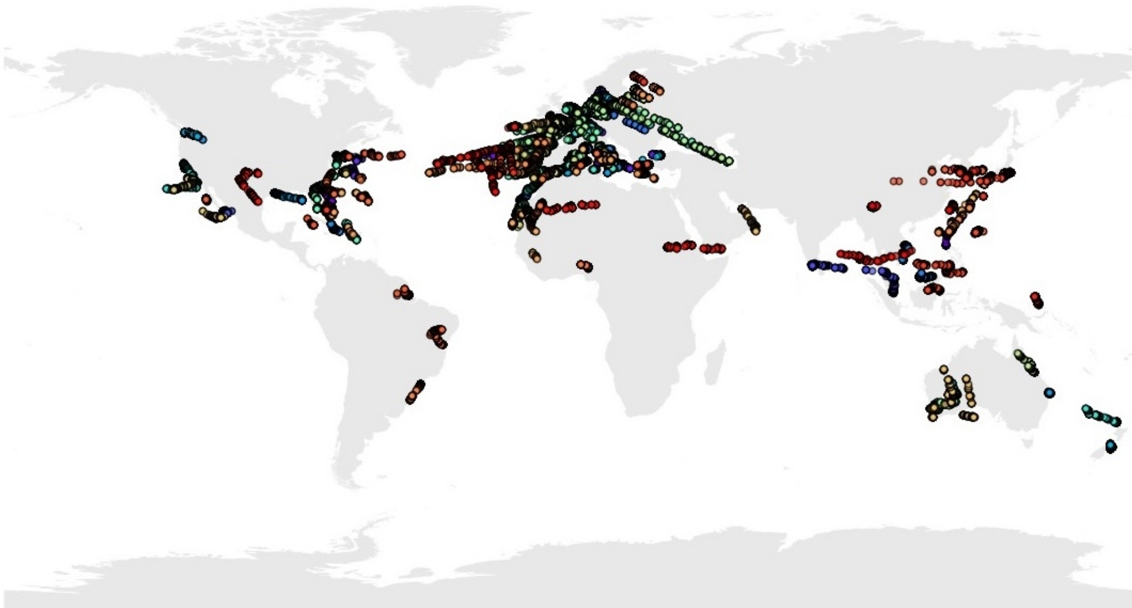

Figure 12: Predictions before post-processing

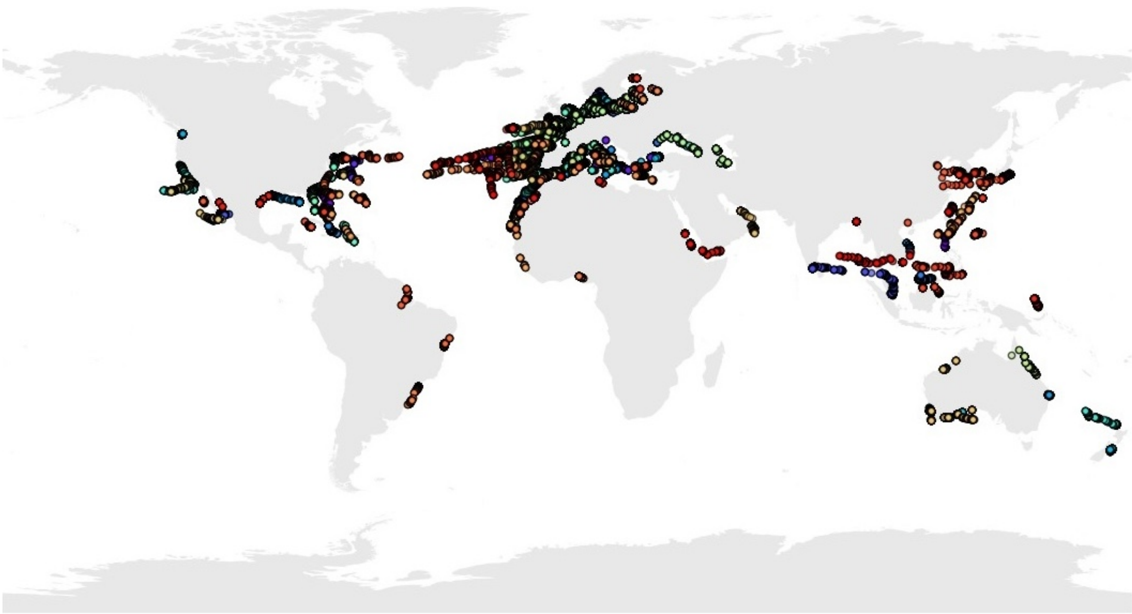

Figure 13: Predictions after post-processing

```
# APPLICATION OF THE MODEL

features = ['latitude_x','longitude_x', 'cog', 'sog', 'heading', '
    time_horizon','latitudePort','longitudePort', 'vessel_dimensions', '
    rot']
target = ['latitude_future','longitude_future']

# Initialization of the scaler for normalizing the features
scaler = StandardScaler()

# Train set
X = train_preproc[features]
X_scaled = scaler.fit_transform(X)

# Target
Y = train_preproc[target]

# Definition of the model
Model = RandomForestRegressor(n_estimators= 200, min_samples_split= 2,
    min_samples_leaf= 1, max_features= 'sqrt', max_depth= 20, bootstrap=
    True, random_state=11)

# Fitting of the model
Model.fit(X_scaled,Y)

# Predictions
X_test = data_test_complete[features]
X_test_scaled = scaler.transform(X_test)
Y_pred = Model.predict(X_test_scaled)
```

Public Score
124.21

4.3 Neural network

We also tried a totally different approach, implementing a neural network. It was built through the function `build_model`:

```python
def build_model(n_nodes, activation, lr, n_hidden_layers):
    first_model = models.Sequential()
    input_dim = X_scaled.shape[1]

    # Input layer
    first_model.add(layers.Dense(n_nodes, activation=activation,
        input_shape=(input_dim,)))

    # Hidden layers
    for _ in range(n_hidden_layers):
        first_model.add(layers.Dense(n_nodes, activation=activation))

    first_model.add(layers.BatchNormalization())
    first_model.add(layers.Dropout(0.2))

    # Output layer
    first_model.add(layers.Dense(2))

    # Compile the model
    optimizer = Adam(learning_rate=lr)
    first_model.compile(optimizer=optimizer, loss='mean_squared_error',
        metrics=['mean_absolute_error'])

    return first_model
```

This function received in input the number of nodes, the activation function, the learning rate and the number of hidden layers that we retrieved using a Random Search. Then it built a neural network with *n_hidden_layers* hidden dense layers, each one with *n_nodes* nodes, and the activation function passed as input. We introduced `BatchNormalization` and `Dropout` after the hidden layers, since they are beneficial for regularization and can help prevent overfitting and we used a dense output layer with 2 nodes, since we wanted to predict both latitude and longitude. Finally, the model was compiled using Adam optimizer with the input learning rate and mean squared error (MSE) as a loss function, since it is suitable for regression tasks.

```python
features = ['latitude_x', 'longitude_x', 'cog', 'sog', 'rot', 'heading',
    'navstat_dummy', 'time_horizon','latitudePort','longitudePort','CEU'
    ,'GT','length']
target = ['latitude_future', 'longitude_future']
X = train_preproc[features]
Y = train_preproc[target]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

model = KerasRegressor(model=build_model, verbose=0, n_nodes=None,
    activation=None, lr=None, n_hidden_layers=None)

param_grid = {'n_nodes': [128, 256, 512],
              'lr': [1e-4, 1e-3],
              'activation': ['tanh', 'relu'],
              'n_hidden_layers': [1, 2, 3],
              'epochs': [10, 20],
              'batch_size': [128, 256]}

random_search = RandomizedSearchCV(estimator=model, param_distributions=
    param_grid, n_iter=30, cv=3, verbose=1)

random_search.fit(X_scaled, Y)
best_params = random_search.best_params_
nodes = best_params['n_nodes']

Model = build_model(n_nodes = nodes,
```

```
                        activation = best_params['activation'],
                        lr = best_params['lr'],
                        n_hidden_layers = best_params['n_hidden_layers'])


early_stopping = EarlyStopping(monitor='loss', patience=5)

epochs = best_params['epochs']
batch_size = best_params['batch_size']

Model.fit(X_scaled, Y, epochs=epochs, batch_size=batch_size, callbacks=
    early_stopping)

X_test = data_test_complete[features]
X_test_scaled = scaler.transform(X_test)

Y_pred = Model.predict(X_test_scaled)
```

Public Score
143.90

In order to perform the Random Search we used `KerasRegressor`, a wrapper that allows using Keras models in scikit-learn workflows, passing to it *n_nodes*, *activation*, *lr* and *n_hidden_layers* initialized to `None`, so that `RandomizedSearchCV` could pass the parameters during tuning. Moreover, in order to save computational time, we added `EarlyStopping`, that stops the training when the monitored loss has stopped improving for 5 epochs.

4.4 Random Forest (iterative version)

Our best scoring in the public test set was reached with a Random Forest regression model, with the fundamental change with respect to Section 4.2 that here we engaged an iterative approach. It means, first of all, that we introduced as main features the autoregressive observations of both present latitude and longitude, with the corresponding time differences between the present observation and the lagged one. Moreover, for each row of the test set, we iteratively predicted the future latitude and longitude positions of vessels based on historical data and then we used those predictions as last known observation of that ship for the next prediction. By doing so, we had to re-compute time differences and autoregressive latitude and longitude every time, as well.
In this configuration we used as features different past lags of latitude and longitude, the time intervals between the lagged observations and the time at which the prediction is made, besides the moving average terms of latitude and longitude. We fitted the model using $n\_estimators = 50$, as it produced the best result on the public test set among multiple attempts

```
def cycle(model, X_test, df, N):
    # Predict latitude and longitude

    data = {}
    for vessel_id, group in df.groupby('vesselId'):
        # Each group becomes a list of tuples
        data[vessel_id] = list(zip(group['latitude_x'], group['
            longitude_x'], group['time']))

    # Initialize the features in X_test
    X_test['latitude_x'] = np.nan
    X_test['longitude_x'] = np.nan
    X_test['latitude_ar1'] = np.nan
    X_test['longitude_ar1'] = np.nan
    X_test['latitude_ar2'] = np.nan
    X_test['longitude_ar2'] = np.nan
    X_test['latitude_ar3'] = np.nan
    X_test['longitude_ar3'] = np.nan
    X_test['time_horizon'] = np.nan
```

```
    X_test['time_ar1'] = np.nan
    X_test['time_ar2'] = np.nan
    X_test['time_ar3'] = np.nan
    X_test['distance_to_port'] = np.nan
    X_test['bearing_to_port'] = np.nan
    X_test['latitude_ma10'] = np.nan
    X_test['longitude_ma10'] = np.nan

    # Dictionary to hold the last known informations for each vessel from the training set
    vessel_last_positions = df[['vesselId', 'latitude_x', 'longitude_x', 'time', 'latitudePort', 'longitudePort']].groupby('vesselId').last().to_dict(orient='index')

    # Lists to store predictions
    predicted_lat = []
    predicted_lon = []

    # Loop through each row in the sorted X_test
    for i, row in X_test.iterrows():
        vessel_id = row['vesselId']
        index_offset1 = len(data[vessel_id]) - N[0]
        index_offset2 = len(data[vessel_id]) - N[1]
        index_offset3 = len(data[vessel_id]) - N[2]

        # Initialize features for this vessel
        row['latitude_x'] = vessel_last_positions[vessel_id]['latitude_x']
        row['longitude_x'] = vessel_last_positions[vessel_id]['longitude_x']
        row['time_horizon'] = (row['time'] - vessel_last_positions[vessel_id]['time']).total_seconds()
        row['latitude_ar1'] = data[vessel_id][index_offset1][0]
        row['longitude_ar1'] = data[vessel_id][index_offset1][1]
        row['latitude_ar2'] = data[vessel_id][index_offset2][0]
        row['longitude_ar2'] = data[vessel_id][index_offset2][1]
        row['latitude_ar3'] = data[vessel_id][index_offset3][0]
        row['longitude_ar3'] = data[vessel_id][index_offset3][1]
        row['time_ar1'] = (row['time'] - data[vessel_id][index_offset1][2]).total_seconds()
        row['time_ar2'] = (row['time'] - data[vessel_id][index_offset2][2]).total_seconds()
        row['time_ar3'] = (row['time'] - data[vessel_id][index_offset3][2]).total_seconds()
        row['distance_to_port'] = haversine(vessel_last_positions[vessel_id]['latitude_x'],vessel_last_positions[vessel_id]['longitude_x'],vessel_last_positions[vessel_id]['latitudePort'],vessel_last_positions[vessel_id]['longitudePort'])
        row['latitude_ma10'] = np.mean([x[0] for x in data[vessel_id][-10:]])
        row['longitude_ma10'] = np.mean([x[1] for x in data[vessel_id][-10:]])

        delta_longitude = np.radians(vessel_last_positions[vessel_id]['longitudePort'] - row['longitude_x'])
        latitude1 = np.radians(row['latitude_x'])
        latitude2 = np.radians(vessel_last_positions[vessel_id]['latitudePort'])

        row['bearing_to_port'] = np.degrees(np.arctan2(np.sin(delta_longitude) * np.cos(latitude2),np.cos(latitude1) * np.sin(latitude2) - np.sin(latitude1) * np.cos(latitude2) * np.cos(delta_longitude)))
```

```python


        # Reorder the row to match the feature order expected by the model
        row_reordered = row[model.feature_names_in_]
        row_np = np.array(row_reordered).reshape(1, -1)
        row_df=pd.DataFrame(row_np, columns=['latitude_x', 'longitude_x', 'latitude_ar1', 'longitude_ar1', 'latitude_ar2', 'longitude_ar2', 'latitude_ar3', 'longitude_ar3', 'time_horizon', 'time_ar1', 'time_ar2', 'time_ar3', 'distance_to_port','bearing_to_port','latitude_ma10','longitude_ma10'])

        # Predict latitude and longitude
        pred = model.predict(row_df)

        # Assuming the model outputs a 2D array, where pred[0][0] is latitude and pred[0][1] is longitude
        predicted_lat.append(pred[0][0])
        predicted_lon.append(pred[0][1])

        # Update latitude and longitude in the vessel_last_positions dictionary
        vessel_last_positions[vessel_id] = {'latitude_x': pred[0][0], 'longitude_x': pred[0][1], 'time': row['time'], 'latitudePort': vessel_last_positions[vessel_id]['latitudePort'], 'longitudePort' : vessel_last_positions[vessel_id]['longitudePort']}

        data[vessel_id].append((pred[0][0], pred[0][1], row['time']))

    return predicted_lat, predicted_lon
```

```python
# APPLICATION OF THE MODEL

# Features and target selection
features = ['latitude_x', 'longitude_x', 'latitude_ar1', 'longitude_ar1', 'latitude_ar2', 'longitude_ar2', 'latitude_ar3', 'longitude_ar3', 'time_horizon', 'time_ar1', 'time_ar2', 'time_ar3', 'distance_to_port','bearing_to_port','latitude_ma10','longitude_ma10']
target = ['latitude_future','longitude_future']

# Train set
X = train_preproc[features]

# Target
Y = train_preproc[target]

# Definition of the model
Model = RandomForestRegressor(n_estimators = 50, random_state=42)

# Fitting of the model
Model.fit(X,Y)
```

Public Score
104.08

5 Model interpretation

5.1 Short notebook 1

In this section we report the model interpretation of the model created in notebook 1. In particular, we focused on the feature importances analysis.

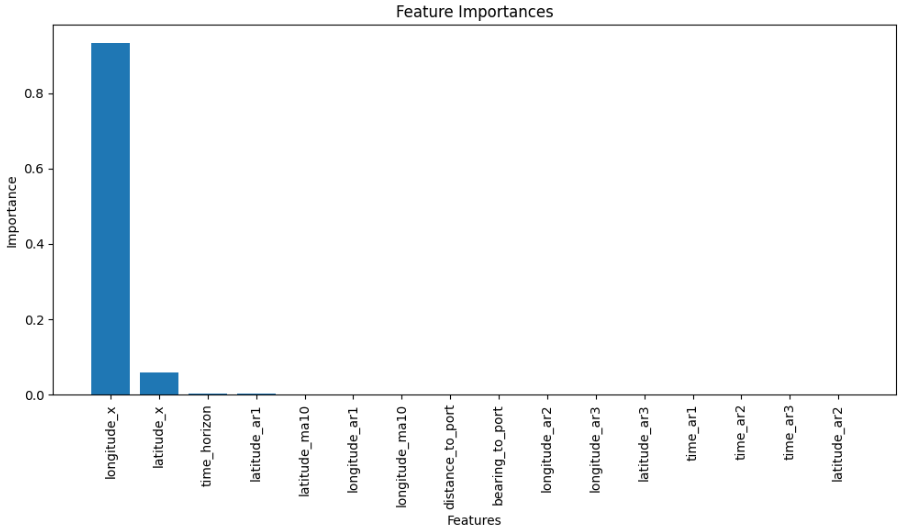

Figure 14: Barplot of the features' importances

Table 5: Feature Importance

Feature	**Importance**
longitude_x	0.934109
latitude_x	0.058979
time_horizon	0.004053
latitude_ar1	0.002151
latitude_ma10	0.000165
longitude_ar1	0.000107
longitude_ma10	0.000073
distance_to_port	0.000053
bearing_to_port	0.000048
longitude_ar2	0.000045
longitude_ar3	0.000045
latitude_ar3	0.000039
time_ar1	0.000037
time_ar2	0.000035
time_ar3	0.000033
latitude_ar2	0.000030

The feature importance analysis shows that the current latitude and longitude features are the most influential in predicting a ship's future latitude and longitude, with longitude being the primary predictor. The time horizon (*time_horizon*) also plays a small yet relevant role, implying that the prediction time interval affects accuracy. Lagged position features like *latitude_ar1* and *longitude_ar1* show minimal importance, indicating limited value from past positions. Similarly, moving averages over 10 past lags (*latitude_ma10*, *longitude_ma10*) contribute negligibly, as do port-related features such as *distance_to_port* and *bearing_to_port*. Finally, other time and lagged features provide only marginal predictive power, emphasizing that current position data largely drive the model's accuracy over historical or contextual data.

5.2 Short notebook 2

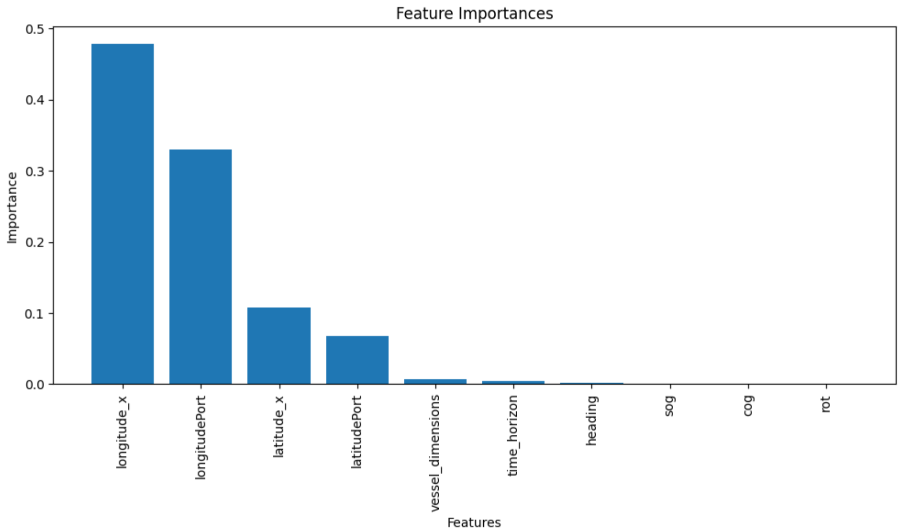

Figure 15: Barplot of the features' importances

The feature importance plot for this Random Forest model predicting latitude and longitude shows that *longitude_x* and *longitudePort* are by far the most influential features, with *longitude_x* being the primary contributor. This indicates that the current longitude and port longitude are critical for predicting future positions, which makes sense as past geographic location and port destination can strongly impact future location estimates. *latitude_x* and *latitudePort* also contribute to the predictions, though to a much lesser extent than longitude, potentially reflecting that longitudinal variation has a greater impact on positional prediction within our specific AIS dataset. The remaining features *vessel_dimensions*, *time_horizon*, *heading*, *sog* , *cog*, and *rot* are of negligible importance, suggesting that these additional nautical and dimensional parameters might not add significant predictive value for latitude and longitude. We tried to avoid the inclusion of *cog*, *heading* and *rot* among the features, but this caused an important decrease of the prediction performance of the model on the public test set.
Interestingly, the feature *time_horizon* has very low importance in this model configuration: this is a major issue, as for each vessel we base subsequent predictions on the last known observation registered in the train dataset. By doing so, future predictions will differ only in terms of the time horizon, since the other features remain fixed at the last recorded observation.

Comparing the two final models, it's clear that the second one, even if it has lower prediction power, is more balanced on the features which actually are effective. The first model, however, likely owes its total reliance on previous observations to the iterative approach used. We have therefore decided to keep both models, even though the first has a much better score on the public test, in order to diversify the approaches and the features that play an important role.