NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4173 - Modern Machine Learning in Practice

AIS Vessel Prediction - Report

Authors:
Einar J. Rye, Henrik S. Grønlund, Theodoros Xenakis

Team name:
[56] Lanternfish

November, 2024

Contents

List of Figures

List of Tables

1 Introduction

This report summarizes our work in creating a machine-learning based prediction model for vessel positions, based on AIS data. The work follows from a project in the course *TDT4173 - Modern Machine Learning in Practice* at NTNU, during the autumn of 2024.

We will first begin with a quick interpretation of the task at hand, along with a rather short explanation of how we chose to solve it.

Afterwards, we will delve deeper into the analysis that led to the development of our solution. This includes the initial exploratory data analysis (EDA), a discussion on the features that we chose to use (and not use), and an exploration of different machine learning models.

Lastly, we will conclude with an interpretation of how our chosen machine learning model works. By - among other things - highlighting which features the model assigns the most importance to.

2 Problem description and solution

Given a dataset containing AIS data for numerous ships in the period 1st of January - 7th of May 2024, we were tasked with predicting the position of a set of ships for timestamps between the 8th and 12th of May.

In general, supervised machine learning revolves around training on many sequences of input/output datasets $(\mathbf{x}_0, \mathbf{y}_0), (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{N-1}, \mathbf{y}_{N-1}), (\mathbf{x}_N, \mathbf{y}_N)$, such that one will be able to predict a dataset $\mathbf{y}_k$, given a *test-set* $\mathbf{x}_k$; where it is implicit that $k \notin \{0, 1, \ldots, N-1, N\}$. Conventionally, the *inputs* - $\mathbf{x}$'s, all tend to have the same dimension, i.e. number of parameters.

The problem with time series forecasting, inherent to this task, was that the test set contained much fewer parameters than those in the training set. Specifically, the test set that we were to evaluate did not contain - among others - information about speed over ground (SOG), course over ground (COG), and heading. Our task was simply to predict latitude and longitude based on timestamp and vessel-identification.

To solve this, we created an enlarged dataset using shifted versions of the data.

To help with our notation, let's denote by t_k, the timestamp for the k-th row in the training set. $\mathbf{y}_k$ is the pair (latitude, longitude). $\mathbf{x}_k$ is the values for all the other parameters in the training set, such as sog and cog. A given row - with index k - in the training set can then be written as $(t_k, \mathbf{x}_k, \mathbf{y}_k)$.

The function *dataSet.shift(n)* has the effect that it shifts all rows by n. That is to say that

$$(t_k, \mathbf{x}_k, \mathbf{y}_k).\textit{shift(n)} = (t_{k-n}, \mathbf{x}_{k-n}, \mathbf{y}_{k-n})$$

We used this function to create a dataset that was roughly N times greater than the original dataset. To illustrate how this was done, we can look at the single row with index k and see how we transformed it:

$$(t_k, \mathbf{x}_k, \mathbf{y}_k) \rightarrow \begin{array}{c} (t_k - t_{k-1},\ \mathbf{x}_{k-1},\ \mathbf{y}_k) \\ (t_k - t_{k-2},\ \mathbf{x}_{k-2},\ \mathbf{y}_k) \\ (t_k - t_{k-3},\ \mathbf{x}_{k-3},\ \mathbf{y}_k) \\ \vdots \\ (t_k - t_{k-(N-1)},\ \mathbf{x}_{k-(N-1)},\ \mathbf{y}_k) \\ (t_k - t_{k-N},\ \mathbf{x}_{k-N},\ \mathbf{y}_k) \end{array}$$

As one can see, all the $\mathbf{y}$'s remain the same, while the $\mathbf{x}$'s are replaced with their shifted counterparts. Time is no longer kept as a variable, but rather replaced by an increment given by the

difference between t and $t.shift(n)$. It's important to note that initially, we sort the whole training set by both time and vessel. This was done with all rows of the training set, rendering a dataset that contained roughly N times more rows than the original training-set.

The idea of constructing such an enlarged dataset is to force the model to learn to answer the question *Given these initial values,* $\mathbf{x}$*, and a time increment* Δt*, where will the ship be in* Δt *seconds?*

By training on such a dataset, we then inserted $\mathbf{x}$ from the last timestamp in the original training-set, into the test set.

Determining N was largely a result of trial and error. As we are predicting quite far into the future (5 days), and the time increments often are not more than 20 minutes, it should be beneficial to use N up to $5 \times 24 \times 60/20 = 360$. We found that stopping at $N = 150$ gave quite good values. Furthermore, we found out that it was not necessary to include all shifts $1, 2, \ldots, 149, 150$. Rather, we could include *some* rows with higher number of shifts, e.g. $1, 2, \ldots, 14, 15, 20, 21, 22, 50, 51, 52, 100, 101, 102, 150, 151, 152$.

3 Exploratory Data Analysis

3.1 Search of domain knowledge

In this section we will explore the data contained in ais_train.csv and ais_test.csv, as well as complementary files such as schedules_to_may_2024.csv.

3.1.1 Train dataset

The main training dataset is a quite large file. It contains 1522065 rows. Each of the rows represents a vessel at a timestamp. To each vessel at each timestamp, there is assigned information such as the vessel's position, its speed, its heading and so on. There are only 688 different vessels in train, which means that, on average, each vessel has approximately 2212 recordings. This is what the training dataset, hereafter denoted by "train" looks like:

time	cog	sog	rot	heading	navstat
2024-01-01 00:00:25	284.0	0.7	0	88	0
2024-01-01 00:00:36	109.6	0.0	-6	347	1
2024-01-01 00:01:45	111.0	11.0	0	112	0
2024-01-01 00:03:11	96.4	0.0	0	142	1
2024-01-01 00:03:51	214.0	19.7	0	215	0

Table 1: Part 1 of train

etaRaw	latitude	longitude	vesselId	portId
01-09 23:00	-34.74370	-57.85130	61e9f3a8b937134a3c4bfdf7	61d371c43aeaecc07011a37f
12-29 20:00	8.89440	-79.47939	61e9f3d4b937134a3c4bff1f	634c4de270937fc01c3a7689
01-02 09:00	39.19065	-76.47567	61e9f436b937134a3c4c0131	61d3847bb7b7526e1adf3d19
12-31 20:00	-34.41189	151.02067	61e9f3b4b937134a3c4bfe77	61d36f770a1807568ff9a126
01-25 12:00	35.88379	-5.91636	61e9f41bb937134a3c4c0087	634c4de270937fc01c3a74f3

Table 2: Part 2 of train

Most of the columns and the units in which they are given are self-explanatory. For course over ground (cog) and speed over ground (sog), however, some clarification might be helpful. Course

over ground is measured in degrees, and indicates in which direction the vessel is moving, ignoring which way the bow is pointing. Heading, on the other hand, tells which way the bow is pointing, and is also given in degrees. In the maritime industry, it is common practice that 0 degrees means straight north, and 90 degrees is straight east. The vessels' rate of rotation is measured in degrees per minute. Speed over ground is measured in nautical miles per hour (knots).

3.1.2 Test dataset

The test dataset, hereafter denoted by "test", differs significantly from train. It contains only a vesselId, a timestamp and a scaling factor. The data points are weighted differently based on their timestamp. Each row is assigned an ID for the purpose of easier scoring in Kaggle.

ID	vesselId	time	scaling_factor
0	61e9f3aeb937134a3c4bfe3d	2024-05-08 00:03:16	0.3
1	61e9f473b937134a3c4c02df	2024-05-08 00:06:17	0.3
2	61e9f469b937134a3c4c029b	2024-05-08 00:10:02	0.3
3	61e9f45bb937134a3c4c0221	2024-05-08 00:10:34	0.3
4	61e9f38eb937134a3c4bfd8d	2024-05-08 00:12:27	0.3

(1)

There are 215 different vessels in test, all of which also appear in train. Test has length 51 739.

3.2 Check if data is intuitive

One can begin by checking for missing values in train. It appears that only portId has missing values; 1615 to be exact.

time	0
cog	0
sog	0
rot	0
heading	0
navstat	0
etaRaw	0
latitude	0
longitude	0
... portId	1615

(2)

PortId is a value that the captain of each vessel can set if he or she wishes to do so. The portId can identify either the destination port or the origin port. It might not be very reliable. Now, let's check if the data is within the expected range:

	cog	sog	rot
count	1.522065e+06	1.522065e+06	1.522065e+06
mean	1.782494e+02	6.331703e+00	5.054561e-01
...	...	...	...
max	3.600000e+02	1.023000e+02	1.280000e+02

Table 3: Part 1: Data columns cog to rot

	heading	navstat	latitude	longitude
count	1.522065e+06	1.522065e+06	1.522065e+06	1.522065e+06
mean	1.762737e+02	2.092604e+00	3.658497e+01	1.153646e+01
...	...	...	...	...
max	5.110000e+02	1.500000e+01	7.055720e+01	1.788054e+02

Table 4: Part 2: Data columns heading to longitude

For all the columns with number values, their input seem to be within the expected range, according to "Datasets_definition_and_explanations.doc". In heading, the value 511 means that a reading is not available. It is necessary to check that there are no values between 360 and 511.

time	cog	sog	rot	heading	navstat	etaRaw
2024-02-20 07:36:36	275.4	0.0	8	483	5	02-18 15:00

Table 5: Part 1 of row in dataframe with error

latitude	longitude	vesselId	portId
53.57729	8.5507	61e9f3adb937134a3c4bfe35	61d375e793c6feb83e5eb3e2

Table 6: Part 2 of row in dataframe with error

Tables 5 and 6 show a row in which heading is 483, which makes no sense. This must be handled.

3.3 Understanding of how the data was generated

Most of the data is obtained automatically from GPS- and satellite readings. Some data points, however, such as "portId" are registered manually by the captain, and could be ambiguous. In supplementary datasets such as schedules, there might be more missing values than in train. A vessel's schedule can't be obtained by GPS, and is therefore missing more often than data from train.

3.4 Exploration of individual features

Using histograms, one can obtain a general idea of the distribution of each column in a dataset.

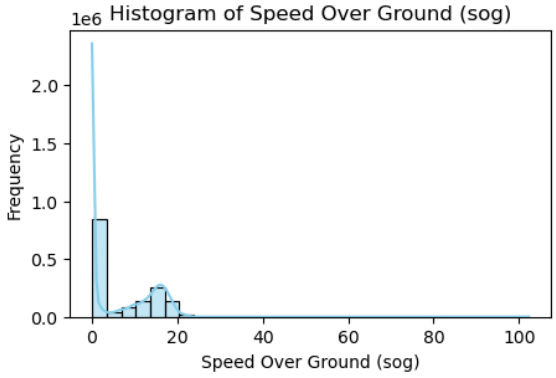

Figure 1: Histogram of speed over grounds recorded

650180 of the vessels are standing still (sog = 0). Does that correlate to navstat? Navstat can take values from 0 to 15. The values 1-6 might indicate that a vessel is standing still, and has speed over ground 0. 738341 of the vessels have either of these navigational statuses. Therefore there seems to be a correlation between the amount of vessels that are standing still, and their navigational status. In the following plot, the distribution of navigational statuses found in train is viewed:

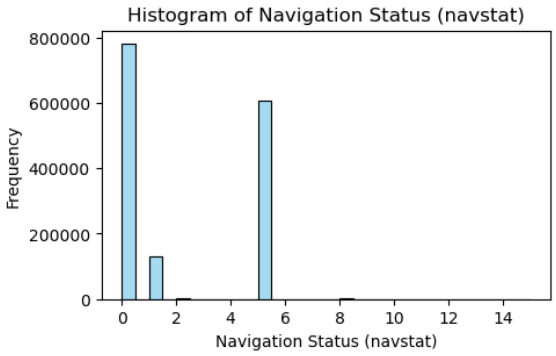

Figure 2: Histogram of navigational statuses found in train

As expected, most vessels have navigational status 0 (under way using engine). Remarkably many also have status 5 (moored) meaning it is not moving for one of several reasons. A fair amount also have status 1 (at anchor).

Let's now focus on the vessels who's navigational status is 0 (under way using engine). What does the speed over ground look like for the vessels with that navigational status?:

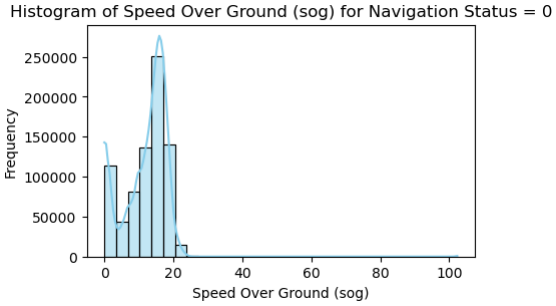

Figure 3: Speed over ground when navstat = 0

The mean time difference in train between data points is just over 20 minutes. It might be a good idea to see if there are any periods where there are more or less data points than usual.

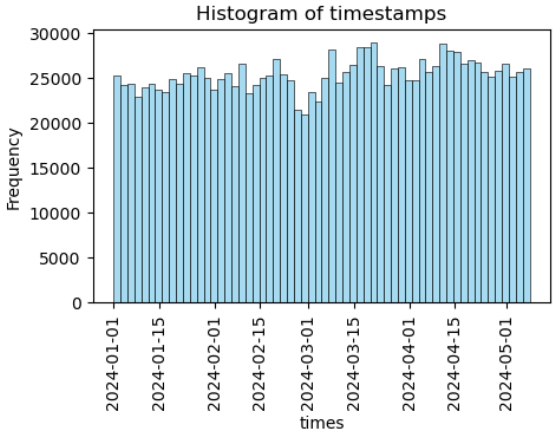

Figure 4: Frequency of measurement of vessels in different time periods

From a histogram of timestamps that appear in train, there seems to be slightly fewer that were registered around the end of February/ start of March. However, there seems to be no time-periods where frequency of measurements dropped or increased very significantly. 2024 was a leap-year so there are some irregularities occurring at the end of February with regards to that.

It is also interesting to note that most of the vessels seem to be sailing around the same longitude as Greenwich:

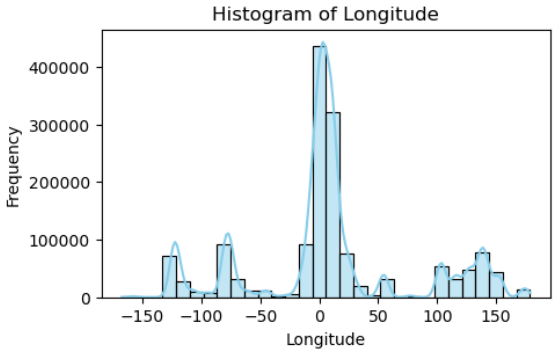

Figure 5: Distribution of all measured longitudes

3.4.1 Distance to port

In the given data, there is a file called "schedules_to_may_2024.csv". It contains information about some of the vessels' planned destinations. The file looks like this after two unnecessary columns have been removed:

vesselId	arrivalDate	sailingDate
61e9f3b1b937134a3c4bfe53	2023-10-02 00:00:00+00:00	2023-10-03 00:00:00+00:00
61e9f3b1b937134a3c4bfe53	2023-10-27 00:00:00+00:00	2023-10-27 00:00:00+00:00
61e9f3b1b937134a3c4bfe53	2023-10-19 00:00:00+00:00	2023-10-20 00:00:00+00:00

portName	portLatitude	portLongitude
Port of Brunswick	31.140556	-81.496667
Port of Southampton	50.902500	-1.428889
Port of Bremerhaven	53.563611	8.554722

Table 7: Schedules after shippingLineName and shippingLineId have been removed

Let's explore the dataset further; we begin by checking for missing values:

vesselId	4615
shippingLineId	0
shippingLineName	0
arrivalDate	5515
sailingDate	1881
portName	4402
portId	4402
portLatitude	4402
portLongitude	4402

Table 8: Overview of missing values in schedules

There are quite a few more missing data points in schedules than in train. It must me changed before it can be used to train a model. One possible solution is removing all the rows in which one of the values are NaN.

The dataset has 121324 rows after rows containing NaN values are removed. 14926 rows were removed.

Not all the columns in schedules are self-explanatory. Here are some clarifications:

- arrivalDate: The date when the vessel is supposed to arrive at the scheduled port.
- sailingDate: The date when the vessel is supposed to depart from the scheduled port. sailingDate will therefore always be later than arrivalDate.

Not all the vesselId's in train also appear in schedules. There are 688 unique vesselId's in train, and 203 unique vesselId's in schedules. 193 of these appear in both datasets, meaning there are 10 in schedules that cannot be found in train.

For the lines in train corresponding to a vesselId that can also be found in schedules, one can identify the vessel's target port, and calculate the distance to that port. For each row in train, one can compare it's "time" - column to all the "arrivalDate" - columns in schedules that have the same vesselId as the specific row in train. From there, one can assume that the row in schedules who's arrivalDate is closest into the future, gives the target port of the row in train. The target port's latitude and longitude can be added to train. If a vesselId in train is not to be found in schedules, the target latitude and target longitude column could be set to -1 by default. Here is an updated version of train once port_lat and port_long columns have been added:

time	cog	sog	rot	heading	navstat	etaRaw
2024-01-12 14:07:47	308.1	17.1	-6	316	0	01-08 06:00
2024-01-12 14:31:00	307.6	17.3	5	313	0	01-14 23:30
2024-01-12 14:57:23	306.8	16.9	5	312	0	01-14 23:30
2024-01-12 15:18:48	307.9	16.9	6	313	0	01-14 23:30
2024-01-12 15:39:47	307.0	16.3	7	313	0	01-14 23:30

Table 9: Part 1: Data columns time to etaRaw

latitude	longitude	vesselId	portId	port_lat	port_long
7.50361	77.58340	61e9f38eb937134a3c4bfd8b	61d376b393c6feb83e5eb50c	18.941944	72.885278
7.57302	77.49505	61e9f38eb937134a3c4bfd8b	61d376d893c6feb83e5eb546	18.941944	72.885278
7.65043	77.39404	61e9f38eb937134a3c4bfd8b	61d376d893c6feb83e5eb546	18.941944	72.885278
7.71275	77.31394	61e9f38eb937134a3c4bfd8b	61d376d893c6feb83e5eb546	18.941944	72.885278
7.77191	77.23585	61e9f38eb937134a3c4bfd8b	61d376d893c6feb83e5eb546	18.941944	72.885278

Table 10: Part 2: Data columns latitude to port_long

When we have created these two new columns, it is easy to calculate the absolute distance (kilometers along the curvature of the earth) between a vessel's position and it's target position. This is added as a separate column.

3.5 Exploration of pairs and groups of features

It is reasonable to assume that there is a correlation between a vessel's distance to target and it's speed over ground:

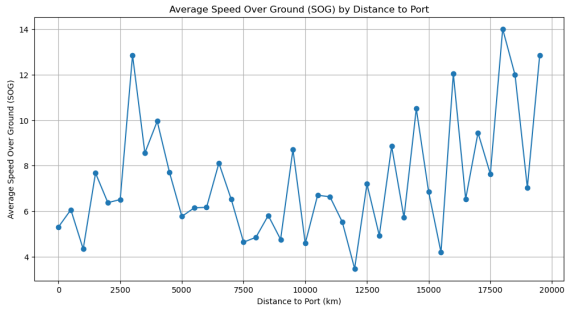

Figure 6: Average speed over ground for given distance to target port

Vessels seem to move fastest when they are furthest away from the target port, and slower when they are close to the target port.

Likewise, there might be a correlation between a vessel's speed over ground and the season at it's position. All the readings from train are from January to May. North of the equator, that time period spans most of winter and spring. During the same time period, summer and fall passes in the southern hemisphere. It seems likely that the climate varies little, if anything at all, near the equator with the different seasons. Further north or south, however, the sailing conditions might differ from season to season. In the training dataset, one can assign to each row a season (summer, fall, winter or spring) based on the vessel's position and the time of year. If the vessel is closer to the equator than 30 degrees, the season is set to "other" and does not contribute in the following plot:

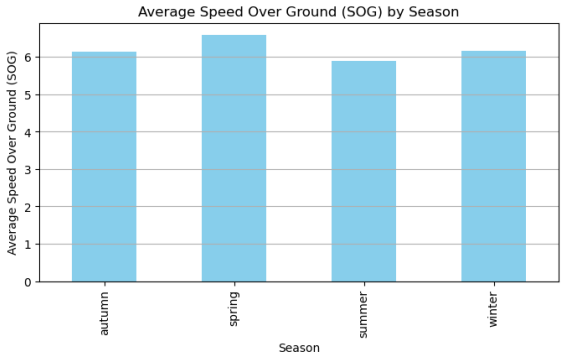

Figure 7: Average speed over ground for each season

A vessel's speed over ground does not seem to vary dramatically with the current season, though it might travel slightly faster in the spring.

To end this subsection, here is a plot of latitude vs longitude for all the rows in train:[1]

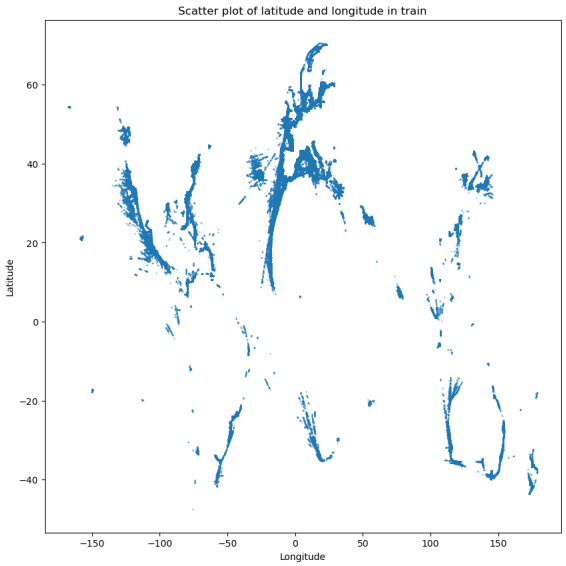

Figure 8: Longitude vs latitude of all rows in train

3.6 Cleaning up of features

To clean up the datasets and prepare them for training, it is possible to both remove rows that contains NaN values, or fill them with a value that makes sense. Additionally, there are several rows that are not needed, such as "shippingLineName" and "shippingLineId". The models we used, RandomForestRegressor and XGBRegressor, do not require scaling, so scaling was not necessary.

As mentioned, there was an error in one row in train in the heading column, as the value read 483. This row could easily be removed.

Schedules had more columns containing missing values than train. We ended up dropping many of the columns in schedules, using only vesselId, arrivalDate, portLatitude and portLongitude. Another problem arose when trying to compare times in train with arrivalDates in schedules; arrivalDates contained time-zone-corrections, which made them difficult to compare with timestamps from train, who did not use time-zone-corrections. This problem was easy to solve after discovering

[1] Notice that what we see are the contours of the world's continents.

that all the time-zone-corrections were +00:00, because then all we had to do was some string-manipulation to remove them. Then we could easily compare the two forms of timestamps after converting them both to datetime-format with pandas.

4 Feature Engineering

As explained in 2, we chose to create a training-dataset that contained *shifted* values. The process of choosing which parameters the **x**'s should contain was by large a guided trial-and-error process, using both the errors in prediction and feature importance plots to validate the features. In short, we created new parameters, incoroporated them individually into the training set, and then tested the model on a validation set. For many of the parameters, we also submitted the results on Kaggle to see if there was an improvement.

We tried creating many new parameters, but almost all of them had a negative impact on the predictions.

In the end, we decided to use 'latitude', 'longitude', 'speed over ground', 'cog', 'rotation', 'heading', 'timeDiffSeconds', '3DayAvgSpeed', 'Port-Longitude', and 'Port-Latitude', and kept them mostly unchanged. In the submitted notebooks the shifted values are denoted by an *s* before the variable name

Feature's we tested but ended up not using

time. We tried using time as a parameter. Furthermore, we tried to feature engineer time as a parameter by separating it into **days after New years**, and **minutes after midnight**. The idea was to capture the difference in days between the different time stamps and the periodicity for individual days. Similarly, we tried with a feature **weekday**, and a binary feature **isWeekend**. Neither of these variables gave any difference in Kaggle results.

isMoving. A binary variable denoting if sog is not zero, gave no difference.

4.1 Principal Component Analysis

4.2 Feature selection

Partial dependence plots (PDP's) were used to distinguish the important features from the features that have little or no impact on the predictions. The following graphs are PDP's for the RandomForestRegressor (RF) model.

The figures show that the models interpret the training set differently. However, there are some clear resemblances between the models, mainly that the most important feature is the previous position of the coordinate we are after. The other variables have a much smaller effect on the result but are important for fine-tuning the results.

5 Models and Algorithms

5.1 Multi-output regression

The task at hand required predicting *two* values per input example. Initially, we handled this by simply wrapping our ML-model with Scikit-learn's *MultiOuputRegressor*[2]. However, we rather quickly realized that we would rather like to use two separate prediction models: one for predicting latitude, and another for predicting longitude. The reason being that this gave us more degrees of freedom to optimize each model separately.

[2]https://scikit-learn.org/dev/modules/generated/sklearn.multioutput.MultiOutputRegressor.html

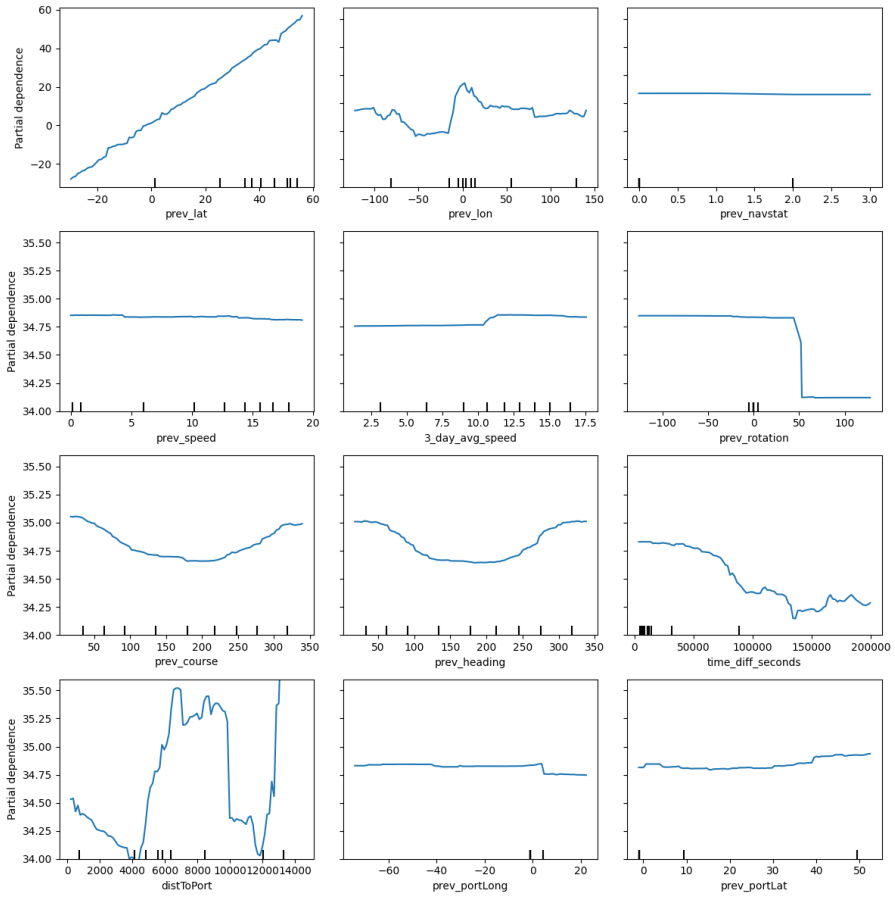

Figure 9: PDP for RF Latitude model
Note that prev_lat has y-axis from -20 to 60, and the other plots have y-axis from 34 to 35.5, this is because of the large difference in importance.

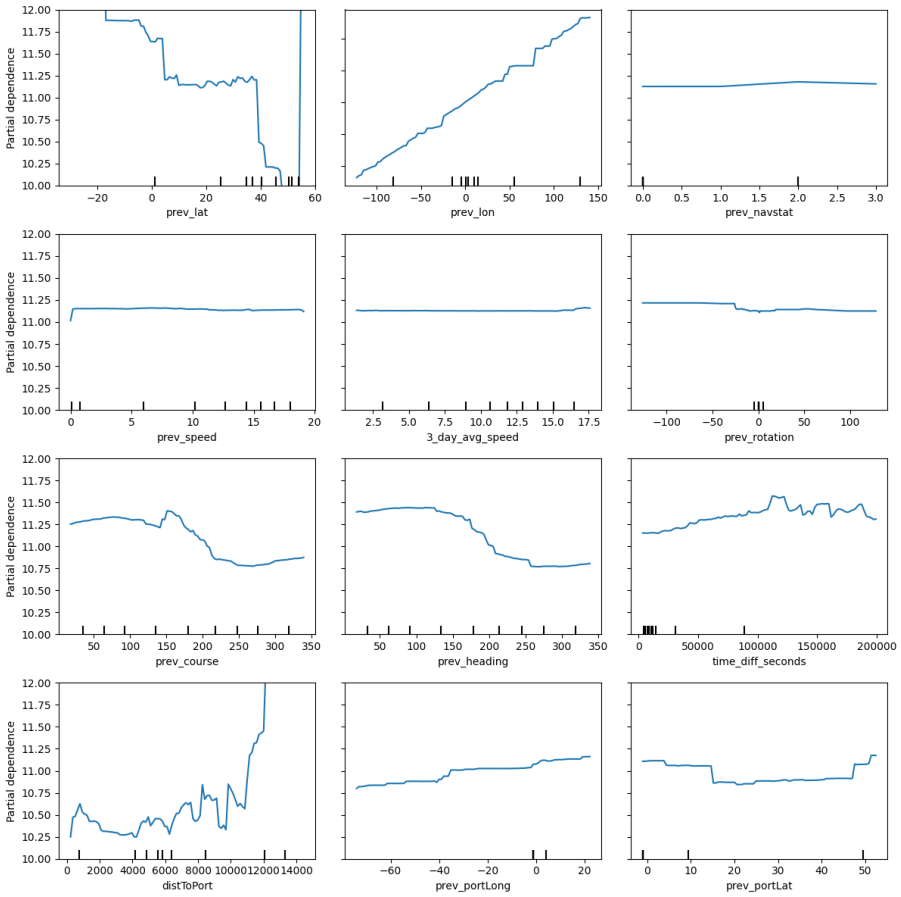

Figure 10: PDP for RF Longitude model
Note that prev_lon has y-axis from -150 to 150 and the other plots have y-axis from 10 to 12, this is because of the large difference in importance.

5.2 Finding optimal ML-software

We tried different models and compared their results to select the models used for the final evaluations. The models tested were XGBoost Regressor, SKlearn RandomForest, CatBoost, and LightGBM.

The corresponding mean average errors (MAE) for the first three selected models are shown in table 11:

Table 11: MAE for the selected models

Model	Lat	Lon
RandomForest	0.050	0.061
XGBoost	0.055	0.064
CatBoost	0.492	0.781

In the tables 12 and 13 we can see the difference between the models and how close the predictions are to the known true values of the vessels.

Table 12: Comparison of latitude predictions for selected models

index	true lat	RandomForest	XGBoost	CatBoost
27409	43.399	43.3878	43.3969	43.4535
854154	58.0106	58.0037	57.9871	58.0259
2523614	37.9595	37.9278	37.9422	38.0931
337882	36.4468	36.6092	36.6139	25.1292
2488530	33.6101	33.622	33.6159	33.7931
774021	43.7779	43.8065	43.8111	43.3885
516757	51.7677	51.7747	51.7928	51.7554
1800421	51.4947	51.4864	51.4823	51.4885
1787207	37.9605	37.9424	37.8869	38.128
1259936	-39.7186	-39.7185	-39.6444	-38.8495

Table 13: Comparison of longitude predictions for selected models

index	true lon	RandomForest	XGBoost	CatBoost
1845415	-8.74984	-8.75235	-8.74885	-8.55013
620374	-71.4061	-71.404	-71.4039	-71.1111
2347597	8.54955	8.54819	8.54023	8.04632
346874	12.257	12.2665	12.255	12.1797
1062195	121.26	121.259	121.264	122.022
1609671	-73.6576	-73.6428	-73.6009	-74.0944
1379588	0.92658	0.926354	0.878679	1.11454
1675588	-0.32445	-0.322958	-0.318593	-0.607173
1035054	14.8702	14.881	14.916	14.9241
1175963	-74.6448	-74.6575	-74.6379	-75.0477

Choosing between XGBoost and RandomForest was not an easy decision as we for quite a long time obtained better results on Kaggle with XGBoost. However, in the end, we managed to get better results with Random Forest, and chose to go with that one.

5.3 Tuning of hyperparameters

Most machine learning models have a wide range of hyperparameters that can be tuned to obtain maximum predictive performance. Finding the optimal such hyperparameters can be rather cumbersome, but is much easier with built-in packages such as Scikit-learns GridSearchCV [3]. Choos-

[3]https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html

ing the hyperparameters was done by letting GridSearchCV "try" out different combinations, and then evaluating the model on a validation set. We mostly conducted this tuning on the XGBoost models, and then used the same hyperparameters for the RandomForest models. The reason for doing so was simply to leverage the speed of the XGBoost training time. As can be noted in *Short_notebook_1.ipynb* and *Short_notebook_2.ipynb*, the hyperparameters for the longitude-model and latitude-model are not equal .

For concrete example of the code used for hyperparameter-tuning, refer to appendix A.

5.4 Postprocessing - Set Nearest Port

Using the package **Global-Land-Mask**, we were able to see how many of our predictions which were points that are in reality on land. Surprisingly, the number was often rather high. For some of our latest predictions, it turns out that up to $30 - 40\%$ of the predicted points were actually on land. We had two hypotheses related to this. *a)* - the trivial one - our ships should stay in the sea. *b)* for predictions on land, it might be beneficial to change the prediction to a position that is closer to the sea. To provide a solution to the problem, we decided to create a postprocessing routine that changed the predictions that were on land. To reduce complexity, we decided not to look for *the* point in the sea that was the closest to the predicted point. Instead, we chose to simply look for the nearest port coordinates for the predicted point on land.

Our postprocessing routine was therefore as follows:

1: **for all** $\mathbf{y} = (\text{lat/lon}) \in$ predictions **do**
2: **if** $\mathbf{y}.onLand ==$ TRUE **then**
3: $\mathbf{y}_{new} \leftarrow \text{argmin}_{\mathbf{p} \in \text{portCoordinates}} ||\mathbf{y} - \mathbf{p}||_H$
4: $\mathbf{y} \leftarrow \mathbf{y}_{new}$

Remark: the $||\cdot||_H$-norm refers to the distance calculated by the Haversine formula[4].

5.5 Automated Machine Learning

In the last years there have emerged packages that handle the whole "pipeline" related to machine learning based problems. That is to say the preprocessing, training (with subsequent validation), and prediction are all automated; often leaving little need for human effort. An example of such a software package is **AutoGluon** [5]. Among others, AutoGluon trains up different machine learning models, with different parameters, and then chooses among the best. This simplifies the work of *a)* choosing a model (such as XGBoost), and *b)* tuning its hyperparameters - a process that often is very much time-consuming. We tried to use AutoGluon several times, but ended up not doing so as we were not content with the score that we got on Kaggle.

6 Model Interpretation

As explained in parts 2, 5 and 3 our models works by predicting the next position based on the last known position and the time difference to the next measure point, along with more data about where the ship is headed. This chapter will therefore focus on the weights assigned in a sample of predictions.

As is shown in the LIME sample figures: 11, 12, 13, 14, 15, the latitude predictions weight the previous latitude position and the time difference most for all the samples. In the longitude model on the other hand weights the previous longitude most in only three of the five samples, in the other samples the previous latitude, and port longitude position were weighted the most. It is

[4] https://en.wikipedia.org/wiki/Haversine_formula
[5] https://auto.gluon.ai/stable/index.html

however evident that the greatest weights are assigned when the previous longitude gets specific values.

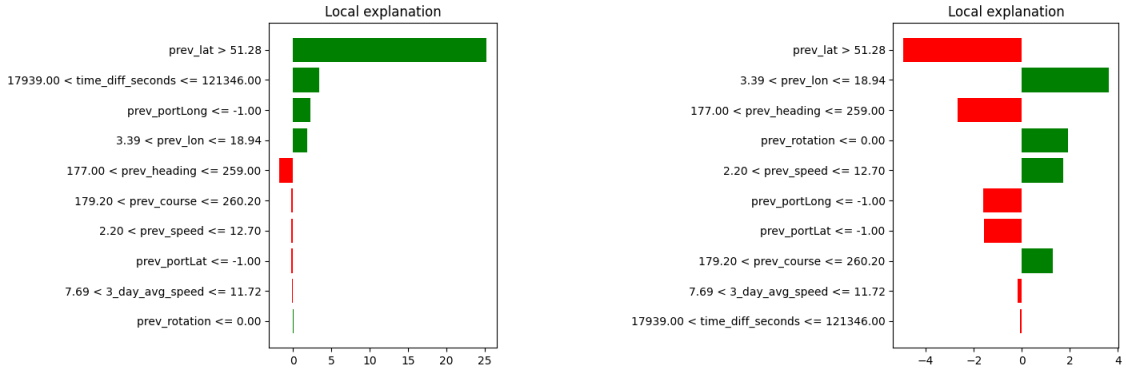

Figure 11: LIME sample 1

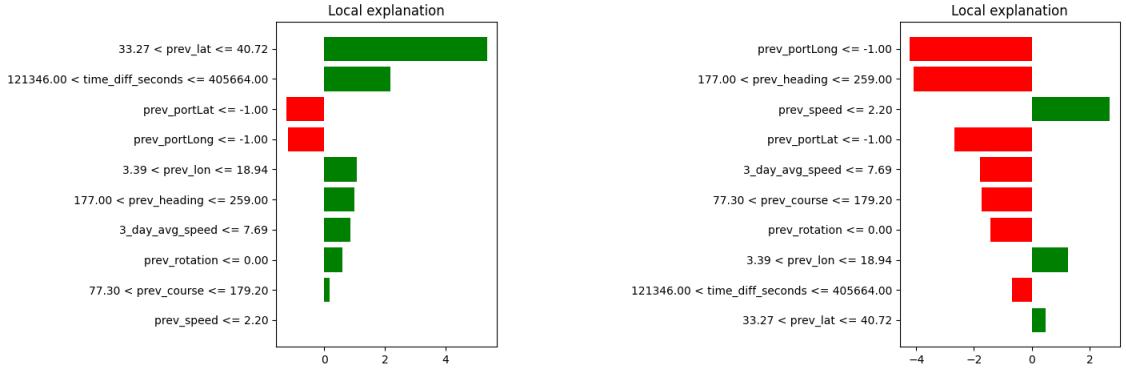

Figure 12: LIME sample 2

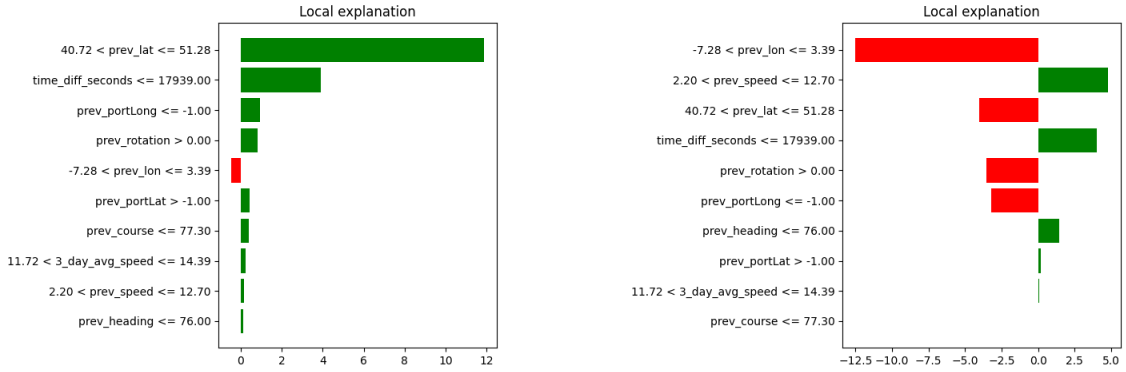

Figure 13: LIME sample 3

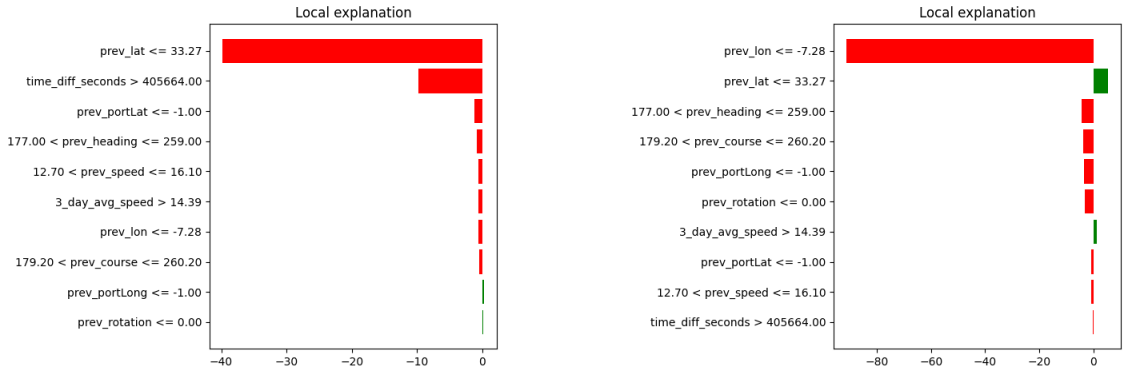

Figure 14: LIME sample 4

7 Appendices

A Code for tuning of hyperparameters (latitude-model)

```
# XGBOOST with Hyperparameter Tuning

# Parameter grid for XGBoost
param_grid_xgb = {
    'n_estimators': [10, 20, 50, 100],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [25, 35, 50],
    'subsample': [0.5, 0.7, 1],
}

# Initialize RandomizedSearchCV for the latitude model
grid_search_lat = GridSearchCV(
    estimator=XGBRegressor(objective='reg:squarederror', random_state=11),
    param_grid=param_grid_xgb,
    #scoring='neg_mean_absolute_error',
    cv=3,  # 3-fold cross-validation
    verbose=3
)

# Fit GridSearchCV on the latitude training data
grid_search_lat.fit(X_lat_train, y_lat_train)

# Retrieve the best parameters and initialize the best model
best_params_lat = grid_search_lat.best_params_
model_lat = XGBRegressor(**best_params_lat, objective='reg:squarederror',
    random_state=11)
```

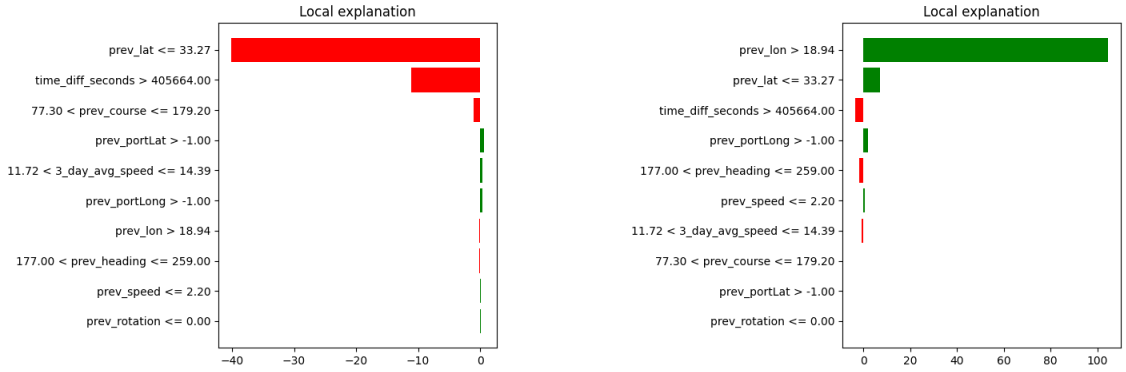

Figure 15: LIME sample 5

```
model_lat.fit(X_lat_train, y_lat_train)

# Make predictions on the validation set
y_lat_pred_val = model_lat.predict(X_lat_val)

# Evaluate performance on the validation set
mae_lat = mean_absolute_error(y_lat_val, y_lat_pred_val)

print("Best parameters for Latitude model:", best_params_lat)
print(f'Mean Absolute Error for Latitude [XGBOOST]: {mae_lat}')
```

Listing 1: Code used to find optimal hyperparameters for XGBoost latitude-model