Predicting the Expected Travel Time of a Vessel

Development Documentation

Authors:

Artur Marschenkulov, Klevi Elezi, Mohammad Aziab
(Group 02)

Lecture:
Software Development

University:
Hamburg University of Technology (TUHH)

1. Introduction
The "Predicting the Expected Travel Time of a Vessel" project aims to develop a software application that predicts the expected travel time for a vessel based on historical vessel movement data. The application provides a graphical user interface for getting predictions for their destination. In addition, the project visualizes the training data to give a simple overview for the user what data is used. The project utilizes the Streamlit framework for building the GUI and integrates various data cleaning and plotting functionalities.

2. Installation and Setup
To set up and run the application locally, follow these steps:

2.1. Dependencies
The implementation of the Vessel Visualization application requires the following dependencies:

- `pandas` : A powerful data manipulation and analysis library in Python.
- `streamlit`: A framework for building interactive web applications with Python.
- `numpy`: A library for numerical computing in Python.
- `scikit-learn`: A machine learning library for Python.
- `matplotlib`: A plotting library for Python, however only its submodule `pyplot`is used.
- `seaborn`: A plotting library which builds upon matplotlib.pyplot. Basically allows prettier and more advanced plots with less code.
- `plotly`: A library for creating interactive and customizable plots. Mainly used for its interactive map feature of “plotly.express”.
- `joblib`: A library for serializing and deserializing data, used to save models.

Since we have a `requirements.txt` it is enough to run the following command to install the above:
```shell
pip install -r requirements.txt
```

In addition, one can also use it with docker:
```shell
docker build -t ett_predictor
docker run-p 8501:8501 ett_predictor
```
Then one simply goes to a browser with the URL “http://172.17.0.2:8051”. The exact URL is shown in the terminal.

2.2. Running the Application
To run the ETT-Predictor application, follow these steps:

1. Create a Python script and copy the provided code into it.

2. Ensure that the required dependencies and modules are installed (as described in section 2.1).

3. Run the script using the following command:
```shell
streamlit run app.py
```
4. The application will start running locally, and a web page will open in your default browser.

3. Infrastructure

In this section, the infrastructure of the "Predicting the Expected Travel Time of a Vessel" project will be described. The infrastructure consists of several components, including the frontend, backend, and necessary dependencies. These components work together to ensure the smooth and efficient operation of the software application.

3.1 Overview

The infrastructure for the project is designed to support the deployment and operation of the software application. It provides the necessary components for data processing, visualization, and user interaction. The key components of the infrastructure include the frontend, backend, and the required dependencies.

3.2 Frontend

The graphical user interface (GUI) of the application is built using the Streamlit library, which provides an interactive and user-friendly framework for creating data-driven web applications. Streamlit simplifies the process of developing modern web applications by utilizing Python and seamlessly integrating with data processing and visualization libraries.

The Frontend GUI application is designed to be very streamlined as it is mainly a demo. Thus, the workflow is presented in top-to-bottom fashion.

Firstly, the user is asked to select an appropriate CSV file with historical AIS data.

After that, the user is given several options. Firstly, one can optionally clean the data. This is actually required if one wants to use the predictor, but we decided to allow to explore the application without cleaning, as it takes quite a lot of time and there is a possibility to firstly look at the visualizations.

Further down there are given a few options, Data Visualization and Prediction. Data Visualization allows us to explore the data further and see whether it is desirable. There one can see whether the general direction of the AIS data is desirable and whether it matches the route.

Then there is the actual prediction part. Since there one has to actually have cleaned the data, if one hasn’t already cleaned the data, it will be done here automatically after one has entered this panel. In addition, the various ML algorithms are trained. This is optimally only done once, and after that the trained ML are cached.

Then one also gets an input form, where one inputs the needed data for a prediction. Those are the destination latitude and longitude, the current latitude and longitude, the length of the ship and the current speed. To make it easier for the user, one can actually see the current location and destination location as red and green dots. We recommend using the actual coordinates given by external devices, but we still provide this possibility of exploration.

3.3 Backend

The backend of the application is responsible for coordinating client requests, handling data processing tasks, and serving the required data to the frontend. In this code, the backend functionality is embedded within the Streamlit framework itself, as it handles the data cleaning and plotting operations.

Here's an overview of the backend functionality:

- Data cleaning: When the "Clean Data" button is clicked, the backend performs data cleaning operations on the uploaded CSV file using the `clean_up` function from the `predictor.clean` module. The cleaned data is stored in the session state for future use.

- Data preparation: After the cleaning is done, one has to prepare the data for the machine learning algorithm. This is done with the `prepare_data` function from the `predictor.clean`module.

- Serializing and Deserializing data: After an Agent is trained, it is desirable to actually store the trained model, so that not to repeat this time consuming effort every time. This is done by the `joblib` module which is integrated into the `ETTAgent` via the methods `dump` and `load`.

- Data visualization operations: The backend handles two plot options selected by the user. The 2 options are “Entry Location” and “Polar Plot”. The first shows the locations of the entries. The goal of this plot is to showcase the data which will be used for generating a machine learning algorithm. The goal is that the user should only pick datasets which match their route as closely as possible. The second plot shows the most frequent direction of the ships in a given dataset. This is important because if the dataset is X to Y, but the user wants to go from Y to X, the predictions will not be very good.

Overall, the backend functionality in this code is tightly integrated with the Streamlit framework, allowing for seamless data processing and visualization based on user input.

4. Limitations

The "Predicting the Expected Travel Time of a Vessel" project has a few limitations:

- Single CSV file upload: The application currently supports the analysis of a single CSV file at a time. Multiple file uploads are not supported.

- Data cleaning limitations: The data cleaning process relies on the `clean_up` function from the `clean` module. Users may need to modify or extend the cleaning process according to the specific characteristics of their data.
- Graphical limitations: The graphical capabilities of the application are dependent on the Streamlit and Plotly libraries. Customizing the visualizations beyond the provided options may require additional development.

5. Data Cleaning

Here we will discuss how we did our data preparation analysis. For a more detailed look we recommend to actually look into the jupyter notebook `data-exploration.ipynb`. Here we will mainly summarize the results.

First of all, here is a short summarization of the labels themselves:

```
- ship:
    - Identifiers: 'MMSI', 'Name', 'Callsign'
    -
    - Dimensions: 'Length', 'Breadth', 'Draught', 'shiptype'
    - Attribute at a certain time:
      - 'time'
      - 'SOG,
      - 'COG', 'TH',
      - 'Latitude', 'Longitude'
- trip:
    - 'TripID'
    - 'StartTime', 'EndTime'
    - 'StartLatitude', 'EndLatitude', 'StartLongitude', 'EndLongitude', 'StartPort',
'EndPort', 'Destination'
- entry:
    - 'ID'
    - 'd'?, 'AisSourcen'
```

Above we tried to group those labels into semantic categories so that the reader understands our following decisions better. The idea is that the above “parent” categories “own” the below ones.

The first steps are that we removed as much duplicate data as possible. This included entries which were simply duplicated completely, however it also included entries with the same ID label, which is problematic as the “ID” is a primary key for an entry. We assumed that there must be something with if the ID is duplicated, so we also removed them, only keeping the first instance.

It was noteworthy that the databases we have gotten used frequently “?” as a NaN value. This at first created much confusion because a few obviously numerical columns were indicated to be “mixed” and the character “?” appeared quite infrequently thus there was quite a long search for finding it.

After fixing the obvious few things, we tried to fix and standardize the data as much as possible.

The labels `COG` and `TH`, are cyclical values from 0 to 360. We removed every entry where the value of those labels were lower or higher than 0 and 360 respectively.

Labels like “MMSI”, “Name”, “Callsign”, “Length”, “Breadth”, “shiptype” logically speaking can’t change within one trip from A to B. However there were a few of such instances. We simply considered the most frequent value within a trip the correct one and replaced any different value with it.

6. Data preparation

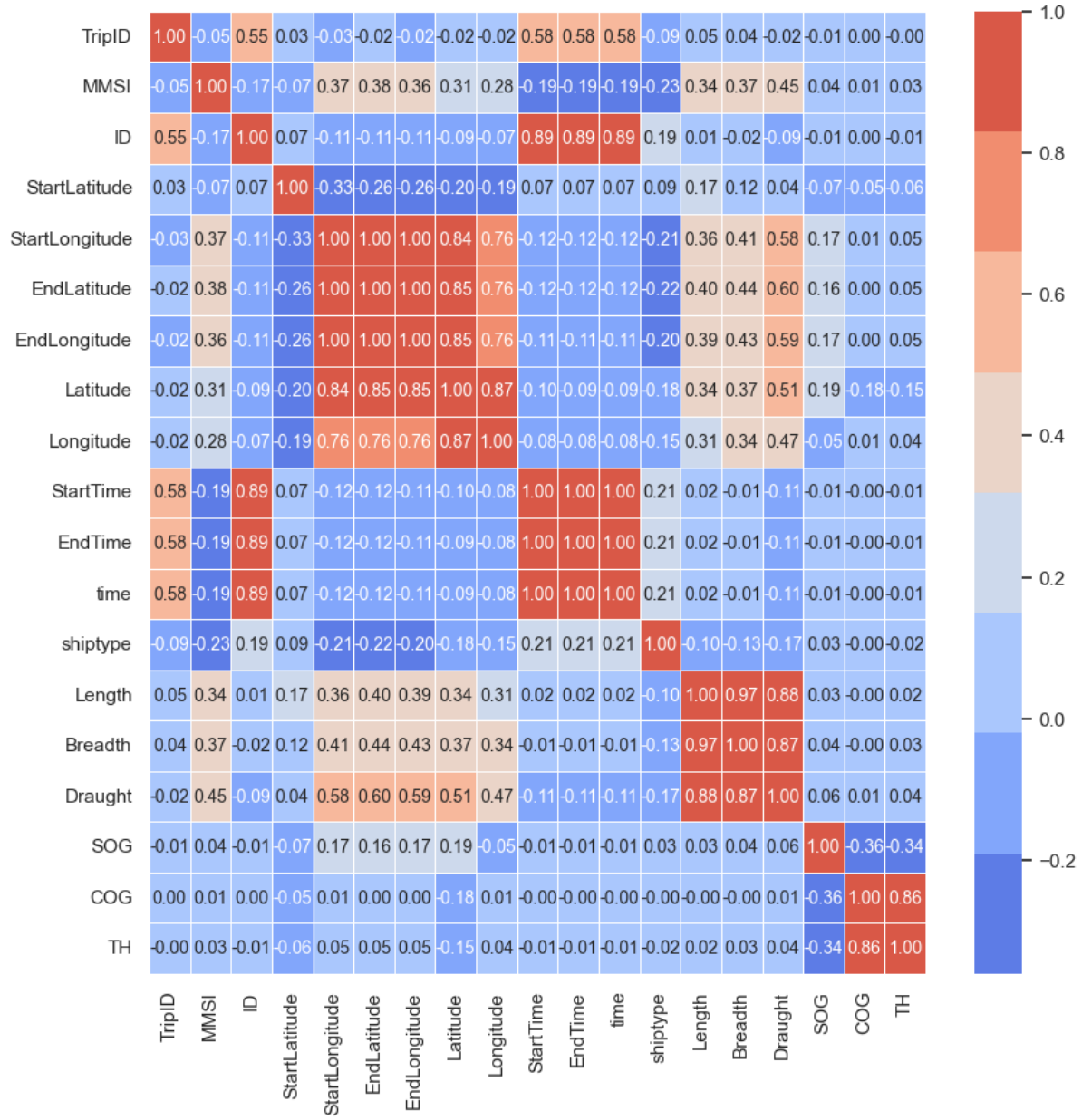
After data cleaning and fixing, data preparation is done. This step is mainly relevant for actually using the dataset for ML algorithms, as selecting which labels/features to use is of big importance. Different variables make an algorithm more or less accurate or fast and the more labels there are generally speaking the slower the algorithm.

Analyzing further, “StartPort”, “EndPort”, “shiptype”, “Destination”, “TripID”, “ID”, “MMSI”, “Name” and “Callsign” are all categorical variables. Meaning there is no inherent notion that one value is above or below the other, like it is the case with the numerical variable “Length”, where 20 meters is longer/bigger than 10 meters. ML algorithms generally speaking prefer numerical variables over categorical ones. So our goal is to find good motivations to remove those labels.

Let us do a logical analysis. Our goal is to find the remaining estimated travel time. For that, “TripID", “MMSI”, “ID” are simply irrelevant by themselves. “Endtime” is something which we only have in those datasets retroactively. We won’t have that information (this is the whole raison d’etre of this project after all). “StartTime”, “StartLatitude” and “StartLongitude” while somewhat relevant, are mostly pointless, since one wants to know the current time and not the total time a trip would take. “AisSource” is simply the source of the AIS data and has no relevance for our needs. “StartPort” and “EndPort” are also expressed with “StartLatitude/Longitude" and “EndLatitude/Longitude”. Thus, we can safely simply remove those labels from consideration. However, we keep “EndTime" for training purposes.

Below is a correlation heatmap. It shows the various correlations between the labels. The idea is that if one can roughly guess the value of one value from another value, it is not needed in the ML algorithm as it only takes up processing power and slows down everything. From Below we can see that “COG” and “TH” are highly correlated, which makes sense since one is the direction of movement, while the other is the direction of the nose of the ship. Thus one can safely drop one of them. “Length”, “Breadth” and “Draught” are also highly correlated, thus we simply choose “Length” as it seems to be the best proxy for the others.

The remaining labels are “COG”, “Length”, “shiptype”, “time”, “EndTime”, “Latitude”, “Longitude”, “EndLatitude”, “EndLongitude”.



Further we considered whether, “COG” and “shiptype” are worthwhile additions. We thought that at the end of the day, we thought that “shiptype” had little value, if we had “SOG” (speed-over-ground) and “Length, especially considering that “shiptype” is a categorical variable. Also we removed “COG”, because it is a numerical cyclical variable, which doesn’t play along nicely with most ML algorithms. There is the option to transform “COG” into cosine and sine values, but then it increases the amount of labels, which makes the algorithms slower.

The last step is to actually create the target label/feature. That this is the feature which we want to predict. We calculate it from the other features like this “(EndTime - StartTime) - (time - StartTime)”.

7. Machine Learning algorithms

Here the various Machine Learning algorithms will be shortly described. We used the scikit-learn library for Python. The ML algorithms used are LinearRegression, RandomForestRegressor, BaggingRegressor, KNeighborsRegressor.

7.1. Linear Regressor
This is a very basic yet also very powerful ML algorithm that establishes the relationship between two or more features using a best fit straight line. It assumes a linear relationship between input variables (X) and the output variables (y). Because of the linear nature of this algorithm, it is not always the best, however it is usually the fastest and does rough insight into the data and its ML potential

7.2 Random Forest Regressor
This is an ensemble learning ML algorithm, constructing several decision trees during training and outputting the average prediction of the individual trees. This method is robust to overfitting due to the averaging process. It is a good method for estimating the importance of various features, so that one understands better which variables contribute more to predictions.

7.3 Bagging Regressor
This is also an ensemble learning ML algorithm, similar to Random Forest.

7.4 KNeighborsRegressor
It is a ML algorithm based on the K nearest neighbor process. Basically, the algorithm looks at the nearest neighbors of a point in a multidimensional space. One then assumes that the point in question must be what its neighbors are.

7.5 Results
Our results showed that from those 4, Linear Regressor is the worst, while Random Forest with Bagging Regressor the best, however Random Forest usually beats bagging Regressors with most scalars.

8. Future Enhancements

The "ETT Predictor” project has potential for further improvements, such as:

We did not implement a broker system. The main reason being that we did not see a big enough benefit, as the error rate was already within 8 minutes usually speaking, while the implementation would be very complex, and also complex to integrate with the GUI. While in our opinion, the costs do not outweigh the benefits, it is still a possibility to do the above. Right now the process is that every dataset is trained on a combination of scalars and estimators (the one mentioned above). The best combination, that is best R2 score, is then chosen to be the ML algorithm with which the predictions are made.

The GUI could be further extended, to make it more interesting. Because it is a demo, the GUI is designed to be very streamlined, however in the future it could be developed into an actual comprehensive tool.

9. Conclusion

The "Predicting the Expected Travel Time of a Vessel" project provides a user-friendly application for visualizing vessel data and predicting travel time. With its interactive GUI and various plot options, users can gain insights into vessel trajectories and movement patterns, as well as the main goal, the ETT prediction.