Personal information

Name: *Pawel Kozela*
TC Handle: *Mloody2000*
Email: *pawel.kozela@gmail.com*

Approach used

- *Approaches considered*

On the **high level**, I tried 3 families of approaches:

- Training a **3-stage model**:
    - In the first step, train a model to classify each line of data separately (predict probabilities of each class) using the raw features available
    - Aggregate the probabilities from the the first stage for each track (namely, calculate the quantiles of probabilities for each class)
    - Train a second model based on those aggregated features to classify the track
- Training a **model on features pre-calculated** for each track:
    - First calculate features (*detailed below*) for each track
    - Train a single model to predict the probabilities for each class based on the aggregated features
- An approach somewhat in the **middle** - the data we have in this match is quite long - 3 months. I figured out, it may be better to **'augment'** the data, by calculating the features for sub-periods. This is what I end up doing in the final submission, with a value of one month
    - First calculate features (*detailed below*) for each track, for each time period (31 days in the final submission, not really optimized though). This give a more data (3 times) to train
    - Train a single model to predict the probabilities for each class based on the aggregated features
    - Aggregate the predictions from each time period - I used a simple median here

Regarding the features, I use standard aggregations on each feature to calculate the values by track - **min, max, standard deviation, average, and some quantiles [.01, .1, .25, .5, .75, .9, .99]** for each feature.

The **features** I calculate for each track / time period are the following ones:

- All available in the raw data - *Latitude, Longitude, SOG, Oceanic Depth, Chlorophyll Concentration, Salinity, Water Surface Elevation, Sea Temperature, Thermocline Depth, Eastward Water Velocity, Northward Water Velocity*
- I add *Latitude / Longitude* difference from previous data point (for a particular track) as well as *'normalized'* difference (absolute value, modulo)

- For a subset of those (*Latitude, Longitude, SOG, Oceanic Depth, and the differences from previous point for Latitude and Longitude* - I use the subset only to limit the number of features, which is already very significant relative to the number of tracks) I calculate the percentiles for different conditions:
    - Based on the Oceanic depth (ie, the percentiles of values, for all data points in a given track / time period when the ship was on deep waters for example)
    - Based on SOG

I've also tried some additional features, but **didn't see any significant improvement**:
- Distance in kms since previous point
- Time difference from previous point
- Average speed (distance / time from previous point)
- Direction of change from previous point (I did only a very naive implementation - Latitude difference / Longitude difference)

The model choice was quite simple - **XGBoost**, that works really good for most types of numerical data. I didn't try other things (except some quick tests with kNN and RandomForest).

Other things tried but not kept:
- **Data cleaning** - the data was not very clean so I tried different cleaning strategies but ultimately decided not to apply them (I suspect though that it 'should' work, I probably didn't tested sufficiently well):
    - Multiple very close data points - there were multiple occurrences of data points with identical values, that were taken seconds apart. The idea was to keep only data point that were more than N minutes apart (so basically I grouped the data by bins of N minutes, and kept the first point in each bin - I tried some intuitive values, like 10, 30, 60 minutes). This didn't degrade the local scores, but didn't improved much neither
    - There were some obviously wrong measures, especially for Latitude / Longitude. I've made a simple filter to ignore the points that are too far apart from the previous one (for ex 10 latitude degrees). It somewhat improved the score (but again, not significantly and I didn't tested enough to be confident)
    - Remove the point where SOG / Oceanic Depth are unknown - since SOG and Oceanic depth were one of the most important features, it could make sense. But didn't
- Oceanic depth vs oceanic depth - there was a strange artifact in the files provided - 302 had in headers 'oceanic depth', while all others had 'Oceanic depth'. I tried to use the information as a feature, but didn't notice any difference (that would obviously be an information leak)
- Ignore data around **Alaska** - the training data contained significantly more ships data from around Alaska than testing (~100 for training, less than 10 for testing) and there was a much bigger proportion of trawlers in this area than elsewhere. I didn't have time though to check with a submission if this introduced some significant bias (my guess is that it would help to ignore this data in training, although I didn't have enough time to test with a submission or prepare an appropriate data split to evaluate

locally)

- *Approach ultimately chosen*

The final submission is the one that implements the third approach - a **single model trained on aggregated features, with the 'data augmentation'**.

Among the important things, that significantly improve the score are:

- The percentiles calculated on **different conditions** - I ended up calculating the values for 4 different oceanic depths (< -500, between -500 and -250, between -250 and -50, > -50) and 2 different SOG (>2 and <2). The idea was than ships behave differently when on high waters, while they may be somewhat more similar on shallow ones
- The data augmentation technique - I used a month (31 days), although this was not sufficiently cross-validated. I later found that something like a week or 2 week improved the score somewhat more
- The model I train is **XGBoost**

- *Steps to approach ultimately chosen, including references to specific lines of your code*

The code is organised in the following way :

- run.sh -> script containing just the export of the env variable and running the python code. **If you change the directory of the code, you will need to update the first line in the script.** As parameters for the script you need to provide:
    - The full path for data directory (where are training.txt, testing.txt and VesselTracks directory)
    - The path (or just filename) where you want to store the predictions for testing data
    - Example: "**./run.sh /home/ubuntu/Mloody2000-FishingForFishermenContest/data/datasets/raw/ predictions.csv**"
- run.py - the main data pipeline including loading / pre-processing / training and predicting
- data.py - helper routines to load / save data, as well as the code to extract the features
- eval.py - helper features for scoring - not really useful in the main pipeline

I've uploaded this on the **VM** into:
/home/ubuntu/Mloody2000-FishingForFishermenContest/src
I've also put in the data in:
/home/ubuntu/Mloody2000-FishingForFishermenContest/data/datasets/raw/

A more detailed look:

- **run.py** - the pipeline is rather straightforward:
    - Load training data (line 20) - just read all train the files in the data directory and put into a dataframe
    - Extract features (line 23, more on that later) from training data

- Recalculate the custom TrackNumbers (lines 27-29) given that I split each track in essentially three tracks (1 per month of data)
- Train XGBClassifier model (line 33)
- Load testing data (line 38)
- Extract features (line 41) from testing data
- Recalculate the custom TrackNumbers (lines 43-45)
- Make predictions (lines 49-50)
- Aggregate by TrackNumber (line 53)
- Save to a file (line 56) - there is actually a safeguard that prevents the save if the file already exists

- **data.py** - the only significant method is **pre_process_data**, which is used to extract the features. For all the calculation I use the pandas dataframe data structure with all the handy aggregation methods. Here are the highlights:
    - Lines 75-76 - Calculate the *Week* and *TrackNumber_NEW* variables based on the time in seconds and TrackNumber
    - Lines 78-82 - Calculate the 4 additional features (latitude / longitude raw and normalized difference from previous point)
    - Lines 84-86 - Calculate the quantiles for each *TrackNumber_NEW* of all the variables
    - Lines 88-89 - Add the number of data points as feature
    - Lines 91-119 - Calculate the quantiles for each subset of TrackNumber_NEW data (based on the Oceanic Depth / SOG conditions). Those are repetitive blocks for calculation with 1 line for calculation and 3 for joining back the data to the original dataframe. For example line 91 can be understood the following way:
        - **df_x[df_x['Oceanic Depth'] > -50]** - filter on Oceanic Depth > -50
        - **[COLS]** - get only desired columns
        - **.groupby('TrackNumber_NEW')** - aggregate per TrackNumber_NEW
        - **.quantile(QUANTILES)** - calculate quantiles for each group
        - **.fillna(0)** - replace the missing values with 0
        - **.stack()** - stack the columns to index
        - **.reset_index()** - put back the index as a simple column
    - Then the lines 92-94 add back the calculated features for each TrackNumber_NEW
    - Lines 121-125 - add the mean and std calculation for each feature

- *Advantages and disadvantages of the approach chosen*

The main advantage (apart from the fact that it works well) is that it's quite robust - that's probably the reason while the cleaning of the data didn't yield a big difference.

The obvious disadvantage though is that it ignores the details (the track path, speed changes). I feel like there wasn't enough tracks though to get something robust, given the time constraints.

- *Comments on libraries*

I use the following standard libraries:
import os
import sys
import datetime
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import KFold

Pandas is great for data analysis - DataFrame is my base data structure in all notebooks
Sklearn - used for its models, cross_validation, pre-processing routines
XGBoost - by far the fastest and most robust implementation gradient boosted trees

- *Comments on open source resources used*

I didn't use any external data.

- *Special guidance given to algorithm based on training*

NA

- *Potential improvements to your algorithm*

Some ideas:

- Improve the 'data augmentation' - I picked 1 month, just because it worked well - didn't have time to find the good value (probably something around 1 week). Also, I did a simple implementation - all time slots starting at time = 0. One could sample much more tracks with different starting points
- Explore more the 'conditional' features - it seems that ships behave differently in specific conditions (high waters) while very similarly in others
- Better aggregation of data - I used only simple moments, since I didn't manage to explore sufficiently the data to get something more interesting (also because the data had missing points / abnormal values) but this could definitely improve the score - I'm thinking about something like the longest period of straight path during a time window, or speed / direction changes
- I didn't try at all NN because we don't have a lot of data. However, with the augmentation technique, it may actually work (I was tempted by a CNN like approach - longitude / latitude as row / columns and other features as 'color' channels)
- Model ensembling - usually works well