Fishing For Fishermen

1 Competitor:

First name: Levente

Last name: Csapo

Topcoder handle: cs_lewy

Email address: cslewy@gmail.com

2 Approach Used

2.1 Approaches considered

I started the contest with a neural network based prediction, with variable number of hidden layers (between 2 and 5) and variable number of neurons per layer (between 10 and 200). This approach gave me a provisional score of ~900k (until submission 20), with its best configuration of 2 hidden layers with 4n neurons each, where n is the number of features.

Starting with submission 21, I switched to a random forest based prediction, which in general seemed to provide a 5% higher AUC score compared to the neural network, while using the same feature set. From that point on, I just tried to clean up the training data and fine-tune the number of trees and depth of the forest.

2.2 Approach chosen

The final approach used several random forests with 4n trees and 2n depth, n being the number of features, with some confidence level adjustments based on the locations of vessels.

2.3 Steps to chosen approach

During the development of my solution, I went with an iterative approach: in the beginning I set up a very basic model with the features that were available right away in the training data. Once this was working, I started improving it piece by piece:

- Extending the feature set
- Cleaning up the training data
- Optimizing the parameters of the model
- Using sub-models

I used an 80%/20% split of the training dataset into real training and cross-validation sets. For local testing, the split was done based on a random 10% of all the samples. For building the models for submissions, the split done based on all 100% of the samples.

2.3.1 Decoding AIS messages

The input data contained several types of AIS messages:

- Types 1, 2, 3, 18, 19 and 24 in the training data
- Types 5 and 24 in the static reports

At first, I was trying to find a free decoder for these messages, but none was really matching my needs. Some were ignoring input lines (possibly because of checksum errors), others were providing their results in ways that were more difficult to process, so I chose to write my own decoder. *(*see *decode/*)*

This decoder is limited to the most basic needs. It takes a filename as input and produced as many lines of output as there are messages in the input file. It only handles the above message types and extract the relevant fields:

- MESSAGE_ID, MMSI, LAT, LONG, POS_ACCURACY, NAV_STATUS, SOG, COG, TRUE_HEADING for message types in the training set
- MESSAGE_ID, MMSI, SHIP_TYPE, TO_BOW, TO_STERN, TO_PORT, TO_STARBOARD, DRAUGHT for message types in the static reports

Fields that are not present in certain message types (like NAV_STATUS in types 18 and 19 or DRAUGHT in type 24) are set to some defaults (16 and 0 respectively).

For message type 24, only Part B is decoded, any Part A produces a list of 0s.

The checksum field is not taken into account for any message.

In order to deal with possible problems of endianness inside bit fields in C++, I wrote a bit manipulator class (see *decode/bitqueue.h)*. While it has some limitations, it was good enough for the job.

2.3.2 Importing features from static reports

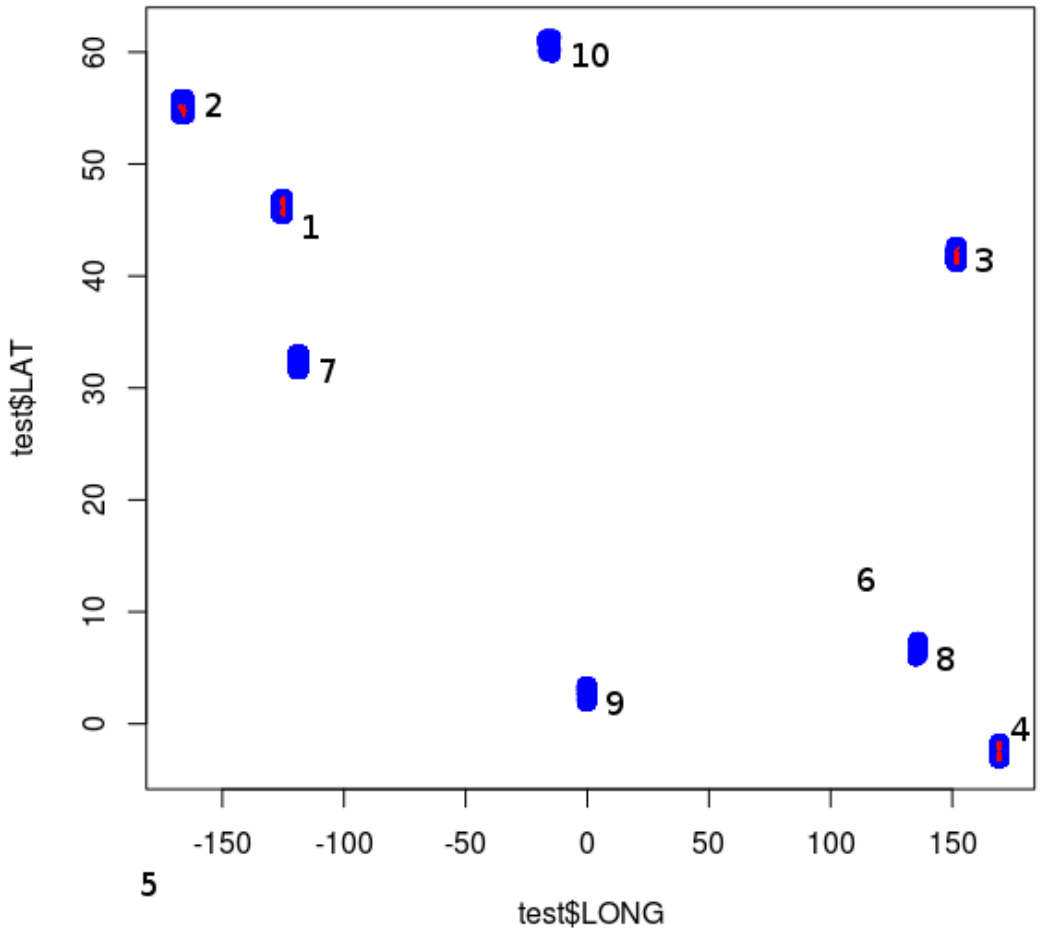
By decoding the static reports the way described above, I was able to obtain some new properties of the ships, identified by their MMSI at specific timestamps. As there were in total more than 9 million static reports, many referring to the same ships at different points in time, I was trying to reduce the size of this database. I also observed that some reports contained only partial information about a ship (e.g. one report contained the SHIP_TYPE but not the dimensions, other report contained some of the dimension fields, etc.)

Assuming that a ship’s (identified by MMSI) dimensions should not change from a timestamp to another, I compacted the existing static messages into a smaller dataset, having just one entry for each MMSI value, and collecting the maximum values of each ship property across all the messages for that MMSI (while observing that some messages contained 0s when the properties were not present) (see *compact.R)*.

This way the original static report could be reduced from ~350MB to ~90KB. This enabled a relatively quick extension of both training data and testing data with these additional features.

2.3.3 Observations on training data

By plotting the training data (later together with the testing data) by each entry's LONG and LAT fields, it could be seen that the observations are from a limited number of spots on the map.



After identifying these spots, I assigned each entry to a spot, resulting in an additional feature, ranging from 1 to 10. This feature was not used for prediction by itself, but for grouping of entries. (see functions *find_spots* and *spotify* from *functions.R)*

Figuring that the relative position of a vessel inside its spot might be a relevant information, I also created the X and Y features (later also X2 and Y2 as their squares), representing the position of each vessel inside its own spot (the position of the spot being that of the first vessel found from that spot).

Some of the entries had their NAV_STATUS and/or TRUE_HEADING features unspecified. In the beginning I just filled them with default values (16 and 512 respectively), but later actually replaced the two whole columns by values decoded from RAW_MESSAGE.

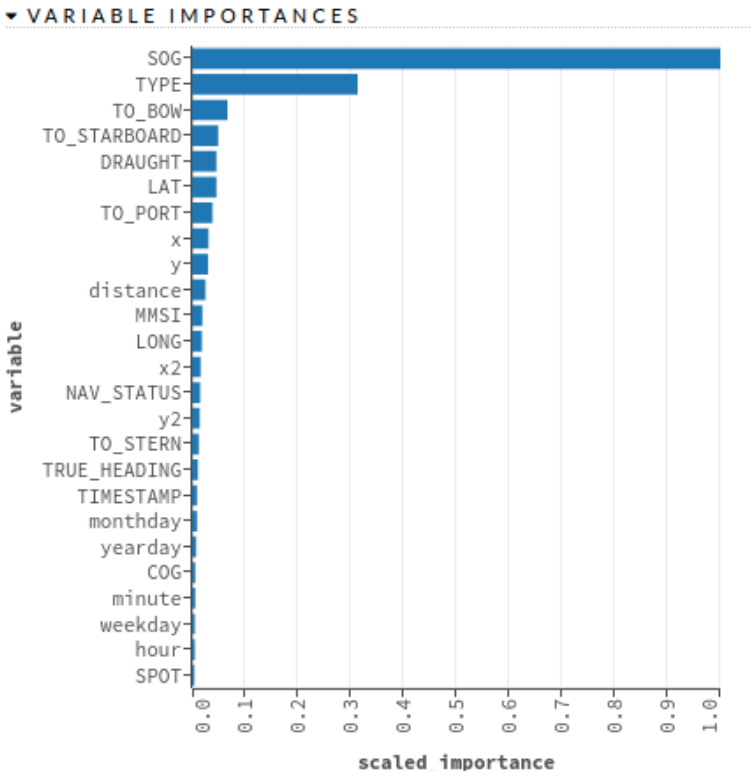
2.3.4 Adding more features

The training (and testing) data contained a TIMESTAMP feature, given as a UNIX timestamp. I broke this into pieces and included them as new features: MONTH, YEARDAY, MONTHDAY,

WEEKDAY, HOUR, MINUTE (being the minute of the day, between 0 and 1439). YEAR was constant over the whole dataset, so I did not include it at all.

2.3.5 Feature selection

By building a random forest with its default configuration (50 trees of depth 20) in H2O Flow (see later), I got a nice comparison about the importances of the features.



Using this comparison, I removed the least important features, as a try to avoid overfitting the model.

The feature set of the generic model ended up being: TIMESTAMP, MMSI, NAV_STATUS, SOG, COG, TRUE_HEADING, SPOT, TYPE, TO_BOW, TO_STERN, TO_PORT, TO_STARBOARD, DRAUGHT, minute, monthday, yearday, month.

This was extended for the spot-specific models (see later) with X, Y, X2, Y2, DISTANCE (from the spot's representative coordinate to the ship). All distances were calculated in degrees, without taking into account the Earth's curvature.

2.3.6 Sample selection

By further analyzing the fishing spots from the training set, there were two of them which did not contain shipping vessels. I removed these completely from the training set, especially that these spots were not present in the testing set at all.

I also removed all entries with FISHING_STATUS == *Unknown*.

2.3.7 Improving the model

Having the training samples cleaned up and the feature set extended with some additional data, I figured that each spot might have different fishing patterns, so it might make sense to build different models for each. This was fine for the 4 spots that were represented both in the training and the testing set, however there were other 4 spots only represented in the testing set. For these, I kept a generic model, built from all the samples from all spots (see *h2o-learn.R* and *functions.R*)

The only difference between the generic model and the spot-specific models was then their feature set, in the sense that I did include X, Y, X2, Y2 for the specific models, but not for the generic one.

2.3.8 Improving the final score

The models obtained in the above way already reached AUC scores above 0.999 on the validation set. The last thing was to realize that AUC of a dataset actually does not depend on the number TP/FP/TN/FN predictions, but only on the rates of matching and non-matching predictions. This means that applying a linear transformation on the resulting predictions does not modify the AUC score, even if the number of TP/FP/TN/FN will be different.

Combining this observation with the idea that the predictions for some spots will be more accurate that others, I thought that AUC can be further improved by scaling the per-spot predicted results in such a way that the more accurate ones get a boosts in confidence level, while the less accurate ones get a drop.

This can be found in function *proposeThresholds* in file *h2o-predict.R*. It increased my score on the validation set from 0.9998612 to 0.999893. I was hoping for it to give similar results during the system test, even though I'm not sure whether the benefit actually applies also to the unseen data.

2.3.9 Last thoughts about the testing data

Analyzing the testing data with regards to distribution of samples between spots, shows:

Spot	Samples
1	67358
2	80453
3	6711
4	5584
5	0
6	0
7	83064
8	841
9	5123
10	212

Having specific models built for spots 1-4, I was confident about the results for those spots, but not so confident about the results for spot 7, which would be predicted using the generic model, having the lowest accuracy. To compensate for this, I manually decreased the prediction confidence for spot 7. Not being able to test this locally, I tried several factors between 0.7 and 0.9, and followed their effect on my provisional score. Finally I settled with 0.8, however this was only a guess. (see *proposeThresholds* in *h2o-predict.R*)

2.4 Libraries and tools

The preprocessing of the input data was done using several basic Linux command line tools, like *grep, head, tail, cut, paste,* together with the AIS decoder written by myself (which requires a C++11 compatible compiler) and some basic R scripts.

The model creation and prediction part is written in R scripts, using these packages:

- *h2o* - *https://cran.r-project.org/web/packages/h2o/index.html*
  - *Prerequisites on Ubuntu: libcurl, jre*
- *data.table* - https://cran.r-project.org/web/packages/data.table/index.html
- *stats* - https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html

H2O's graphical front-end has been used for the analysis of variable importances.

2.5 Advantages and disadvantages

One significant advantage of the algorithm would be that it enables both higher accuracy predictions using the spot-specific models when the test data is part of an already well-described spot, as well as generic predictions for ships that are from previously unseen regions.

Disadvantages might come from some of the development choices that seemed to increase the provisional score, but have not been verified to be generally valid. Such choices include especially the confidence threshold modification and the threshold value of 0.8 being used for unknown areas.

2.6 Notes

This report relies on data as it has been determined during the development of the algorithm.

Some values, especially spot ids presented here might have been allocated in this specific order by random chance, as the training data might have been more or less filtered at the time of their identification. Later runs of the algorithm might produce different order, meaning that an already trained set of spot-specific models would not fit well, or at least would have to be reshuffled the same way as the spot ids differ.

2.7 Potential improvements

The compacting of static reports is done with the assumption that a ship's characteristics do not change. However, if this is not the case (which could be especially for DRAUGHT), it might be worth retaining several entries for the same MMSI and finding the most appropriate to be used for the extension of a sample based on its timestamp.

The confidence threshold modification for spot 7 is hardly a scientifically proven value. It's more like an intuitive guess. However, without additional knowledge about the testing data, especially without some known samples from spot 7 that could be included in the cross-validation set, there's no real chance to determine it more precisely.