**Contest:** Fishing For Fishermen 2

**Name:**  Wladimir Leite

**TC Handle:** wleite

**Email:**  wladimir.leite@gmail.com

**Approaches Overview**

My solution uses ten (two for each class present on training data, including “other”) binary classification random forests, combining two different approaches: one having the whole track as the object to be classified (as belonging or not to a given class), and other based on individual “reports” (each line of the input files).

**Approach Description**

**Parsing Data**: all fields present in the provided data files are parsed and kept as they are. The only exceptions are entries in which the latitude or longitude values are out of range, i.e. abs(latitude) > 90 or abs(longitude) > 180. These lines were completely ignored. Another special treatment was implemented for tracks that contains longitude values close to (and in both sides of) the -180/180 meridian. In such cases, longitude values are converted to the [0, 360] range, so sorting/averaging them would make more sense.

**Track-based models:** as mentioned before, five independent random forests were trained, one for each class present in training data (trawler, seiner, longliner, support and other), using the whole track as the object to be classified. This is the main model, which gave me the best results.

**Track features:** my solution extracts basic statistics features from eleven “sets” of values. Each of these sets takes all track’s reports values of following fields:

* chlorophyll;
* eastwardVelocity;
* northVelocity;
* elevation;
* latitude;
* longitude;
* oceanicDepth;
* salinity;
* sog;
* temperature;
* Haversine distance / delta time (from the current entry to the previous one).

So all values are raw fields belonging to track’s reports, except the last one, which is calculated using the current and the previous report.

Each set of values are then sorted and divided into 16 “bins” (of about the same size), resulting in 17 values, used as features. The first feature would be the smallest value for that field, the second would be about 1/16 of the sorted list, and so on, until the last one (17th), which is the largest value.

Additionally, the average and the difference between the maximum and the minimum values are used as features. In the case of these two features, entries with the special value of -99999 (missing values?) are ignored.

Another feature value calculated for each set is the “weighted average” of the raw values, using the elapsed time from the current entry to the previous one as the weight.

Finally, four “global” features are added for the track:

* Duration (time for the last report – initial time);
* Number of reports for the track;
* Track’s initial latitude (taken from the first report of the track);
* Track’s initial longitude.

**Data augmentation:** that is an important point of the solution, as I observed sensible improvement after starting adding extra “synthetic tracks” to the training data. For each real track, my solution adds other 12 tracks, randomly generated using four different kinds of modifications:

* Subsampling reports (ignoring reports with a certain probability);
* Oversampling reports (creating intermediate reports, by mixing randomly two consecutive real reports values);
* Discarding contiguous sequences of track’s reports, from the beginning and from the end;
* Discarding a contiguous sequence of reports, taken randomly from the middle of the real reports.

**Report-based models:** another five models were trained using individual reports as independent objects to be classified. These models have a much larger amount of samples, but conceptually each object doesn’t hold much information, so the observed results were worse, as expected. I implemented this later, and after getting not so good results I was about to discard it, but instead decided to combine its results with the main model (track-based), giving a smaller weight to the report-based model. So this model acts like a tie-breaker for the main model, and I believe that it helped reducing the variance of the whole solution.

In the case of these models, only 12 features were used: the raw field values, plus the calculated haversine distance between the current report and the first report of the same track, i.e. the total distance at that moment.

No augmentation was used here, since there were already too many objects to deal with.

**Testing:** initially the predict function of each trained random forest is called. This function returns a probability value between 0 and 1, for a given track or report, to belong to a certain class. The report individual predicted values are combined, using a simple average of individual values of all reports that belong to a track. Then this averaged value is combined with the predicted value returned by the track-based model, using a proportion of 1:6.

At this point we have 5 values, one for each class, all between 0 and 1. Before normalizing the sum to 1, a “bonus” based on the “ambiguity” is added to the class which had the best prediction. The added value is equal to the difference between the highest probability and the second best. For example, is we have trawler=0.7, seiner=0.6, longliner=0.3, support=0.2 and other=0.1, a bonus of 0.1 (0.7-0.6) will be given to “trawler”, which would have its value increased to 0.8. Now the sum would be 2.0 (0.8 + 0.6 + 0.3 + 0.2 + 0.1). After normalization, the final values for that hypothetical track would be: trawler=0.4, seiner=0.3, longliner=0.15, support=0.1 and other=0.05.

Before generating the output file, the raw predicted values are converted to a decreasing sequence, in order to avoid ties. For each class the first (highest probability) receives a value of 1, and the following tracks receive the previous value \* 0.999. So the values contained in the generated file are not the probabilities, but just a rank-based value.

**Comments on libraries/open source resources used**

My final submission did not use any library or open source resource.

I spent quite some time working with public vector maps. I tried to add the distance for a given point to the closest portion of land (in any direction and to each of four/eight directions). I believed that would help a lot, but in practice I didn’t observed any gain. I suspect that either my implementation was wrong or there were already enough longitude/latitude values in the training data, so that extra information would be redundant (and possibly noisy, due to map resolution for example).

**Potential improvements to my algorithm**

Using external data may help, but as the predictions are already accurate (considering final scores were near 95%), at this point, it can be hard to distinguish real gain from small random variations.