**Fishing for Fishermen 2**

* First Name: Peter
* Last Name: Novotný
* TopCoder handle: nofto
* Email address: [novotny.peto@gmail.com](mailto:novotny.peto@gmail.com)
* Final code submission file name: nofto-FishingForFishermenContest.zip

**Approaches considered**

The match fits to the general pattern which occurs frequently in the TopCoder Marathon Matches: We have some training data and we need to learn something from it. Out of the various machine learning techniques, I only implemented random forest before, which showed to be quite successful in the past matches. Since I was on holiday till September 4th, with only 4 days to go, I did not consider to use other techniques. So, my “approach consideration” was only about specific subparts, e. g., pre-processing, feature extraction, …

**Approach ultimately chosen**

For the desired classification, one may use 1 classifying forest with 5 classes, or 4 forests with binary classification (one forest for each ship type, except for “other”, which we do not need to classify separately, since its weight is 0 in the scoring formula). I have used 4 forests, but there should not be a big difference in performance between these two options.

Each ship was considered a single sample, so I have trained with 1209 samples. The data from “VesselTracks/” folder was used to extract features. Together, I have extracted 78 features, but some of them were skipped in the final submission.

**Steps to approach ultimately chosen**

* Pre-processing
* Training stage
  + Collecting statistics for specific features
  + Feature extraction for train samples
  + Creating random forest classifier
* Testing stage
  + Feature extraction for test samples
  + Applying random forest classifier

Description of the steps follows. The notation [*m*-*n*] refers to the line numbers in the file “run.cpp”.

**Pre-processing**

Since the most time consuming operation in my solution was parsing the textual data from “VesselTracks/”, I converted all those data into binary files and worked only with them in all the tests later. For this purpose, I set up “binarize.cpp” script, which:

* Creates the file “lengths.csv”, which for each file “*ID*.csv” located in “VesselTracks/” folder contains one line of the form “*ID*,*L*”, where *L* is the number of data lines in “*ID*.csv”.
* For each file “*ID*.csv”, generates the corresponding file “*ID*.dat” in “VesselTracksBin/” folder. This binary file contains the sequence of 12 × *L* doubles; each block of *L* doubles corresponds to one field column (so the first *L* values are “Relative Time”, the next *L* values are “Latitude”, and so on).

All the other job is done with “run.cpp” script.

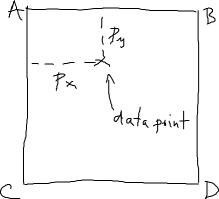
**Training stage**

At the beginning, lengths from “lengths.csv” are loaded [1010-1021]. Also, training IDs and ground truth is loaded from “training.csv” [1035-1048].

**Collecting statistics for specific features**

Some of the features I have used are calculated from statistics collected from the entire training data. These statistics are collected [1113-1116] and stored in binary files [1133-1163]. The method “processShipForRegionStats()” [516-675] is used for this collection:

* For each rectangular region (in terms of latitude and longitude) of various grid size (see [21]), the number of data points of each ship type which occur in this region and the number of different ships which contribute to this region is calculated. This statistic is calculated not per region, but per region-corners. That is, if a data point occurs in a region, then it contributes to each of the 4 corners of the region with weights with respect to the vertical and horizontal distances from the corners. The sum of weights for each data point is 1 (see [632-673]):



Hence, each region corner stores 6 values: sum of weights of data points for each ship type and number of different ships which contributed to it.

* In a similar fashion, but only in 1-dimensional space, the statistics about speed over ground (SOG) is collected [573-630]. That is, for each interval of a specific length, it is stored how many data points for each ship type have SOG in this interval, and the contribution is stored per interval end points. There are two copies of these statistics: one counts only data points with 0.2 ≤ SOG ≤ 20, the other with 5 ≤ SOG ≤ 100. The statistics is collected for different interval lengths so that the number of intervals varies from 2 to 128 (see [34]).
* Finally, in a similar fashion, the statistics for three different pairs of variables is stored [523-572]. That is, rectangular regions of different sizes are formed in the 2D-space of the given variable pair and contribution to the region corners are collected. This is done for the following three pairs of variables [12-13], but only for 0.2 ≤ SOG ≤ 20:
  + (SOG, oceanic depth)
  + (SOG, latitude)
  + (SOG, longitude)

**Feature extraction for train samples**

To each ship, 78 features are assigned [1118 – 1124]. It is done with “processShip()” method [832-990]. The list of the features, with zero based numbering, follows:

0: Number of data points for the given ship

1, 3, …, 23: There are 12 variable columns – so there are 12 arithmetic means (one for each column)

2, 4, …, 24: These are the variances of each variable column

Each data point is converted from (latitude, longitude) to the 3D-space. Hence, to each data point, *x*, y, and *z* coordinate is assigned, with (0, 0, 0) being the center of the Earth [839-849].

25, …, 29: Mean, variance, minimum, maximum and (maximum – minimum) of *x*

30, …, 34: Mean, variance, minimum, maximum and (maximum – minimum) of *y*

35, …, 39: Mean, variance, minimum, maximum and (maximum – minimum) of *z*

To every two consecutive data points, we can assign *t* – the time difference, *d* – the distance, and *v* – the average speed = *d* / *t*. This is done only if the resulting speed *v* is small enough [390], to avoid some clearly incorrect values which occur in the data set.

40, 41: Mean and variance of *d*

42, 43: Mean and variance of *t*

44, 45: Mean and variance of *v*

For every three consecutive data points, we calculate *α* – the angle between two segments formed by the ship movement [850-853]. It is a number between 0 and 180. If the ship turns very sharply, the value is very small. If it follows a line, the value is close to 180.

46, 47: Mean and variance of *α*

The remaining features are related to the statistics collected at the beginning. To avoid overfitting, before assigning the remaining features to a ship from the training set, we first strip the data points of that ship from the collected statistics [897], then assign the features [898-986], and afterwards we update the statistics to the previous state [987]. For the given variable or the pair of variables, we then calculate 5 features as follows:

* To each data point, we assign the 5-tuple of real numbers with sum 1 in this way:
  + We look at the smallest region (which may be 2-dimensional or 1-dimensional) containing this data point and collect the values from the vertices with weights like in the picture few pages above. We also count (with the same weights) the value of *S* – the number of different ships contributing to this region.
  + Define the 5-tuple *T* = (*t*1, *t*2, *t*3, *t*4, *t*5), where *ti* is the (weighted) number of data points of the *i*-th ship type in the region.
  + If *S* is at least 10.0 [23, 37], we assign to the data point the 5-tuple *T*.
  + If *S* is less than 10.0, then we look at the region one level up and calculate (recursively) the 5-tuple *T’* in the same way as *T* described above. Finally, we assign to the data point the 5-tuple *S* · *T* + (10 − *S*) · *T’*.
* The 5-tuples assigned to every data points are normalized: (*t*1, *t*2, *t*3, *t*4, *t*5) is replaced with (*t*1, *t*2, *t*3, *t*4, *t*5) / (*t*1 + *t*2 + *t*3 + *t*4 + *t*5).
* Arithmetic mean of all the normalized 5-tuples is calculated and the resulting 5 coordinates are added as features.

48, …, 52: features related to the statistics of (latitude, longitude) pair

53, …, 57: features related to the statistics of SOG variable with 0.2 ≤ SOG ≤ 20

58, …, 62: features related to the statistics of SOG variable with 5 ≤ SOG ≤ 100

63, …, 67: features related to the statistics of (SOG, oceanic depth) pair

68, …, 72: features related to the statistics of (SOG, latitude) pair

73, …, 77: features related to the statistics of (SOG, longitude) pair

**Creating random forest classifier**

My own implementation of random forest is used (I have used it in many matches before). 1000 trees are created for binary classification of each of the 4 ship types [1087 – 1095]. The possible parameters for the random forests are:

|  |  |
| --- | --- |
| *Parameter description* | *Value in final submission* |
| Number of features used to split nodes | 6 (see [15]) |
| Stopping criteria | MINNODE = 1, MAXLEVEL = 50 [46, 44] |
| Cost function used to split nodes | [4] |

MINNODE = 1 means that the tree will grow up to the nodes of size 1, which means there is no stopping criteria based on node size. MAXLEVEL is the maximum depth of the tree. I did not check if this level was ever reached, but with only 1209 samples, this stopping criteria is very probably irrelevant. I did not optimize these parameters very deeply, only tried several values and selected the ones with best results.

Since the number of features is quite big (78) when comparing to the number of train samples (1209), I tried to remove some of the features. The final list of ignored features is {29, 23, 24, 22, 1, 2, 19, 39, 21, 32, 0, 7, 18, 36, 26, 16, 34, 12, 4, 33, 28, 20, 6, 14, 27, 38, 11, 30} (see [18]). I created this list be checking, which features are used fewer times for splitting the nodes. There was only a small gain in score after ignoring these features, so it is probably not necessary to ignore any features. There are 28 ignored features, so the actual number of used features is 50.

**Testing stage**

**Feature extraction for test samples**

The same process as in training stage is applied [1204]. The only difference is that there is no need to strip the collected statistics before calculating features 48 – 77.

**Applying random forest classifier**

Each ship is evaluated on each of the four forests [1207], and the result of each forest is the ratio – confidence that the candidate qualifies for the given ship type. So this is written to the resulting file [1208].

**Open source resources and tools used**

No open source resources nor tools were used.

**Advantages and disadvantages of the approach chosen**

The algorithm runs very quickly. It terminates in few minutes using only one thread. And it needs only small amount of RAM (below 1 GB, and this is mainly to store the statistics, which may be easily optimized if necessary).

The algorithm does not extract much information about the ship trajectory shape. Some information of this kind is only in features 46 and 47.

**Comments on libraries**

Only the standard C++ library is used.

**Comments on open source resources used**

No open source resources were used.

**Special guidance given to algorithm based on training**

All the steps are described in above. The script “run.cpp” should be executed three times:

1. run train

This collects the statistics, extracts ground truth and the features for train samples and stores all these data in several binary files in “bindata1/” folder.

1. run rf

This reads the features and ground truth from binary files, builds the random forests and stores them in “bindata1/” folder.

1. run test

This reads the collected statistics, extracts the features for test samples, evaluates it on the forests and writes the result to stdout.

**Potential improvements to the algorithm**

One can improve the algorithm at various levels:

* add another set of features (there are many possible ways, how information can be extracted in the form of features),
* replace the random forest with some other ML technique,
* optimize various parameters.

**Evolution of my provisional score**

Here I will show how my provisional score evolved with adding new features, to get a better picture how useful the specific features are.

|  |  |
| --- | --- |
| Features | Provisional Score |
| Feature 0 and another 12 features with first value of each variable | 827036.89 |
| Features 0, 1, …, 24 | 890181.56 |
| Features 0, 1, …, 47 | 898001.29 |
| Features 0, 1, …, 52 | 912540.32 |
| Features 0, 1, …, 62 | 934798.55 |
| Features 0, 1, …, 67 | 937842.50 |
| Features 0, 1, …, 77 | 938586.88 |