# Information

First Name: Marcin

Last Name: Kowiel

Topcoder handle: crystals

Email address: komarcin@interia.pl

Final code submission file name: result024\_stack\_krl\_smot\_com9.csv

# Approach Used

#### Approaches considered

For the processing I used Python2 with scikit-learn as the main library for machine learning.

Initially, I used *RandomForestClassifier* and *KNeighborsClassifier* trained on vessel tracks data. To avoid overfitting I performed 5-fold cross validation on the train data. Since the train set consists of over 8 million rows the training time was quite slow, and I tried to subsample the training set to about 2 million rows. Additionally, I removed rows with missing data. I tested *OneVsRestClassifier with RandomForestClassifier and KNeighborsClassifier*, however the AUC score was worse in that case, therefore I abandoned this approach.

#### Approach ultimately chosen

After a week I decided to reduce the number of data rows and prepared one row of derived attributes for each vessel. For each vessel I calculated the *mean*, *std*, *min*, *1st quartile*, *median*, *3rd quartile*, *max* of every attribute. The same aggregates were computed for the 5 additional attributes described in the Data Preparation section below. To balance the class ratio I used the *SMOTE* algorithm on the data that was rescaled with uniform *QuantileTransformer*. Then I used *StackingCVClassifier* on *KNeighborsClassifier*, *RandomForestClassifier*, *LGBMClassifier* with *LGBMClassifier* as the meta classifier. The *StackingCVClassifier* used prediction probabilities and all the attributes available for the lower level of classifiers.

The hyper-parameters were manually tuned, based on weighted AUC score calculated from 5x5 fold cross validation.

#### Dataset preparation

I calculated 5 additional attributes: `*Total Distance*`, `*Distance*`, `*Time* *Diff*`, `*Speed*`, and `*Water* *Velocity*`.

* The `*Total Distance*` is the distance from the position (Longitude and Latitude) in the first row to the position in the current row (in the vessel dataset) in kilometers.
* The `*Distance*` is a distance from the position in the previous row to the current row in the dataset.
* *`Time* *Diff`* is a time difference from the position in the previous row to the current row in the dataset.
* `*Speed*` was ` *Distance`/`Time* *Diff`* in km/h*.*
* `*Water* *Velocity*` was *sqrt*[(`Northward Water Velocity`)2+(`Eastward Water Velocity`)2]

`*Speed*` was a good attribute to detect inconsistencies in the dataset (*Longitude* and *Latitude* could be wrong). I excluded all rows with speed >=60. Moreover, I kept the data with [look at prepare.py/ clean()]:

-90<=*Latitude*<=90

-180<=*Longitude*<=180

Time Diff > 60

SOG<25

From the raw vessel data provided I calculated *mean*, *std*, *min*, *1st quartile*, *median*, *3rd quartile*, *max* of each initial attributes (columns) and the 5 additional attributes described above.

Careful analysis of data shows that *SOG* is quite important to distinguish different types of vessels, especially ‘seiner’ vessels from the other types. After noticing this fact, I have calculated the *median* and *std* of all attributes for the rows with 0.2 < *SOG* < 5 and 5<=*SOG*<25 respectively. The attributes obtained in that way seemed to be the most important.

All attributes were rounded to 2 decimal points, and attributes calculated from *Latitude* and *Longitude* were rounded to closest integer value. Since the vessel tracks were a time series, the missing values were filled with the first value of the attribute that was not missing (not -99999). In case the following value was missing, I used the previous not missing value.

This way I obtained one row of new attributes for each vessel.

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

I successfully added new attributes to the data, then optimized the hyper-parameters by trial and error manually. I tested the *KNeighborsClassifier*, *RandomForestClassifier*, *LGBMClassifier* separately. Then I usually tried to remove attributes that had low importance score based on *LGBMClassifier* from the data. Look at purge\_data() function to see all removed attributes from the final submission. Then I tried to find the best *Scaler (Normalizer)* and so called *Imputer*. After that I started to play with *StackingCVClassifier.* I did not try to optimize meta classifier hyperparameters, instead I used the best *RandomForestClassifier or LGBMClassifier* for the meta classifier. The *LGBMClassifier* was consistently a little bit better so I stuck with it. In the last two days of the competition, I added extra data rows to the training data with the *SMOTE* algorithm. Even though my AUC score on test data was worse, I stuck with the approach since I believed I had overfitted the testing data. My speculation was right, since the provisional score was better with *SMOTE*.

#### Open source resources and tools used, including URLs and references to specific lines of your code

For data analysis and processing I used

Numpy <http://www.numpy.org/>

Pandas <http://pandas.pydata.org/>

Matplotlib <https://matplotlib.org/>

To build machine learning model I used open source Python libraries:

Scikit-learn <http://scikit-learn.org/stable/>

Mlxtend <https://rasbt.github.io/mlxtend/>

Lightgbm <https://github.com/Microsoft/LightGBM>

#### imbalanced-learn <http://contrib.scikit-learn.org/imbalanced-learn/stable/install.html>

#### Advantages and disadvantages of the approach chosen

Based on all the data I prepared one row of attributes for each *TrackNumber*. That led to a solution where machine learning model could be trained and evaluated quickly. Moreover, with this approach the CPU and memory consumption were quite low compared to training a model based on all data provided. That also led to better handling of class imbalance. On the other hand, it was hard to estimate the information loss, and preprocessing time was quite long.

#### Comments on libraries

I used well known Python libraries used in machine learning studies. They are already proven to be reliable and are used by many researchers. They are highly optimized, reliable and fast.

#### Comments on open source resources used

I did not used external open source resources.

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

I used 5x5 cross validation to avoid overfitting. The hyper-parameters were chosen manually, by trial and error, based on training AUC score confirmed by improvement in provisional score.

I used *SMOTE* with : ratio=0.80, k\_neighbors=5, m\_neighbors=15.

For *KNeighborsClassifier* optimal score I set k=15, with L1 metric and points weighted by the inverse of their distance.

The *RandomForestClassifier* was trained using n\_estimators=199 and max\_features=0.18, class\_weight='balanced'.

The *LGBMClassifier* was trained with num\_leaves=64, learning\_rate=0.09, n\_estimators=139, min\_child\_weight=4, min\_child\_samples=2, min\_split\_gain=0, subsample=1, colsample\_bytree=0.58, scale\_pos\_weight=1.

The StackingCVClassifier was used to improve the final probabilities. Stacking was based on all attributes with additional 15 (3x5) attributes based on class probabilities generated by the three mentioned classifiers. As the meta classifier I used LGBMClassifier with hyper-parameters as mentioned above.

#### Potential improvements to your algorithm

The further steps will require finding better attributes like closest distance to coast line, cleaning data from not reliable inputs. Since the attributes are highly correlated it would also be worthwhile to spend more time on feature selection for example with Recursive feature elimination (RFE).