Classifying Vessels via Trajectory Estimation and Clustering

CSDS 340: Machine Learning – Case Study 2
Clay Preusch & Grant Konkel

Choice of Algorithm:

Our approach to this case study was to create an algorithm that maps vessel trajectories over time, and mimics hierarchical clustering on these trajectories. The intuition behind this approach is that merging points into trajectories, and then clustering trajectories together using single-linkage would recreate complete vessel paths. Essentially, we constructed partial paths for each boat, and then tied the ends of these paths together.

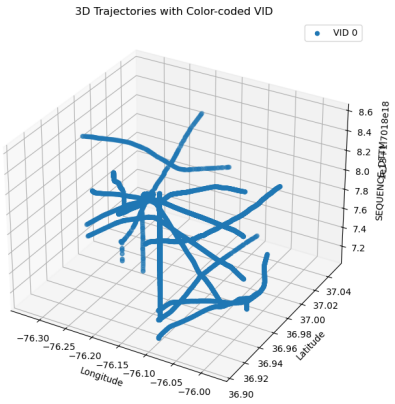
After doing some research, we found no hierarchical methods that use supervised learning, so we conformed and tried to find an unsupervised solution. Unfortunately, we similarly found no hierarchical methods that are based on time, so we created our own. Our unsupervised algorithm, as would be expected, does not use any training data.

When classifying points, simply using latitudinal and longitudinal coordinates to predict the boat a datapoint belongs to has significant consequences that we hoped to avoid in our algorithm. To solve this, we iteratively added a point to an empty trajectory, and predicted where that point would be after some amount of time (depending on the change in time from the last point of our trajectory to each sample point). Then we calculated the squared error between our expected trajectory and the sample point using a weighted squared error of latitude, longitude, course over ground and speed over ground. If the squared error was lower than some threshold (which we tested multiple values of, but landed on 1), we added the sample point to our trajectory and continued looping through the points. Once through the points, we assigned the first unchosen point to a new trajectory and repeated the process until all points had been added to some trajectory.

This process left us with lots of partial trajectories that still needed to be merged into complete boats, which is where the hierarchical aspect of our approach comes in. To do this, we assumed that most trajectories were only missing values before and after the terminal points (i.e. the algorithm would not group 1 and 3 and mistakenly miss point 2), which allowed us to next merge trajectories (similar to before) by comparing the squared errors of the start and end points. We then merge the closest trajectories until we are left with k boats (in the case where we are given k). When we are not given k, we choose estimated k (based on a plot of all points) and do the same process to limit the number of boats.

Preprocessing:

Because we created our own algorithm, we found no need for traditional data preprocessing, as we were able to scale any values as needed on the inside of our algorithm. This allowed for more flexibility and control because we know exactly what each scaling needs to be. All of our numerical calculations were either predicting where a point would be after some amount of time or calculating the squared error. Predicting the location of a point required no significant scaling (only scaling to the necessary units). To find the weighted squared error of two points, we *did* need to scale data so that we could find an error that is dependent on each feature (latitude, longitude, speed over ground, course over ground) by the amount we saw fit.



Latitude and longitude barely vary so we scaled both up by a factor of 1,000. The speed difference was very high, so we divided the difference in tenths of a knot by 10. Similarly, course over ground was giving an error that was higher than needed so we scaled it down by 1/10,000 *and* also multiplied by mean speed (because the value of course over ground is more important for higher speeds). Each value was squared and summed into our weighted error.

To choose our expected number of boats when there was none given, we tried two different techniques. The first was to "loosen" our threshold for merging points and simply report the number of boats predicted. This gave very promising results for set2, and very bad results for set1 (results with an ARI in the .95 and .15 ranges, respectively). Because of the poor results for set1, we ultimately decided to copy our technique that we used for PredictWithK using an estimate for k, hoping the results would turn out better. To estimate this value, we plotted all points by latitude, longitude and time (shown above) and we roughly counted 11 boats. We added 10 to 11 to not over-constrain the sets (in case any boats were not easily separable by eye) so we condensed our predictions to 21 boats when no k was given.

Selection and Testing of Algorithm:

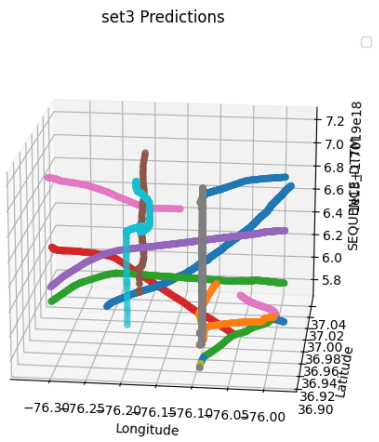
During our initial testing, we experimented with a variety of scikit-learn algorithms. Key among these were: Agglomerative Hierarchical Clustering, Spectral Clustering, Random Forest Classification, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). While these were not all terrific avenues of exploration in hindsight, they were each chosen for a reason. ARI values produced by all of the algorithms that we experimented with can be found in the Appendix.
Agglomerative Hierarchical Clustering was employed with single linkage. The motivation behind this approach was to connect clusters based on minimum inter-cluster distance, with the idea that building upon clusters would help chain paths of vessels together. This approach showed some promise, and served as inspiration for our ultimate implementation, but ultimately had limitations we hoped to avoid. Spectral Clustering was considered because of its ability to identify clusters with nonlinear boundaries, as we expected to see in our vessel data. Our hopes were that by transforming the data into a higher-dimensional space, Spectral Clustering would be able to capture some complex relationships between data that wasn't immediately apparent to us. This approach was not very successful. We also tried Random Forest Classification, simply because of the success we experienced with this algorithm during Case Study 1. While this algorithm is arguably not very applicable to our dataset, the hope was that bagging would account for the variance in the vessel data. DBSCAN was an algorithm that we weren't familiar with prior to this assignment. It was chosen because according to scikit-learn, it "Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density". Although we didn't expect to find dense clusters of signals from a single vessel, the hope was that DBSCAN would be able to find core sections of a trajectory, and expand paths from there.

After this experimentation, we decided to proceed with an approach that didn't fully leverage any scikit-learn algorithms. We felt that creating our own algorithm would allow us full control over the data flow, from start-to-finish. This was especially applicable to our dataset, as we felt that our features provided enough information to directly calculate vessel paths. This process is detailed in the *Choice of Algorithm* section, and our complete calculations can be found in our code.

Once our algorithm had performed trajectory estimation, we were not close to accurate classification of all points, but we were able to glean valuable information from these trajectories. By graphing partial boat paths provided by these trajectories and examining the cardinalities of each boat class compared to the boats given in sets 1 and 2, we realized that we were successful in classifying segments of each boat path. The next step was to tie each of these segments together.

Our initial idea was to perform hierarchical clustering, but we were unable to find an implementation that would allow us to use initial cluster assignments, as had been generated by our trajectory estimation. We wanted to use hierarchical clustering with single linkage, as this would tie the ends of each partial path together. After realizing that we would have to code this ourselves if we wanted to utilize our initial cluster assignments, we opted for a simpler solution. Our code found the maximum and minimum examples (by timestep) for each trajectory. We then computed the squared error between the ends of all trajectories, and found the pairs with the least amount of error. We combined trajectories in this pairwise fashion until we were left with the desired k number of boats.

After completing the initial creation of our algorithm, we were happy with the process that we had chosen, but were still getting poor results in testing the data sets. We believed this to be from issues with our threshold value and our weighted squared error. We ran on debug mode to see where issues were arising and they did come in the squared error function that weights different features differently. Specifically, points with speeds close to zero were dependent almost entirely on latitude and longitude which is where our classification broke. We tried scaling these values up much higher (from an initial scaling of 10) but couldn't find a scaling that gave great results. Ultimately we ended on 100,000 because it produces a more promising classification (though still with flaws) without spending too much time.



When examining our predictions graphed on the vessel paths of set3, there are some visible issues with our classification. However, to the naked eye, we are successfully reconstructing the paths of multiple boats. These results indicate to us that our approach had significant promise, and our methods have merit. In the future, we would spend more time testing a wider variety of squared error weights and subsequent threshold values. We believe that our approach is sound, and stand behind our results.

Recommendations:

If our algorithm were to be deployed as a multiple target tracking system, some possible choices of metric would be: trajectory accuracy, completeness/purity of trajectories, and

cluster quality. Trajectory accuracy, implemented as either MSE or RMSE, would serve as a good measurement tool for determining the difference between predicted and actual vessel trajectories. A higher trajectory accuracy would indicate that our algorithm accurately predicts the positions of vessels over time, reflecting the precision of our predictions. The completeness/purity of a trajectory would measure how well our algorithm is able to capture complete vessel paths without missing or falsely identifying data points. This could effectively be implemented as some combination of precision and F1 score. F1 score would particularly be effective, as it strikes a balance between precision and true positive rate. Cluster quality could be analyzed using silhouette scores for formed clusters. By analyzing these scores, we can see how well-defined and separated different vessel clusters are.

Dealing with practical issues, such as gaps in data, are a challenging but necessary hurdle in machine learning. Handling gaps in the data would best be done with interpolation techniques. Methods such as linear interpolation could be used to estimate vessel positions during data gaps. This would minimize the impact of missing data, and enhance the algorithm's ability to provide continuous trajectories. Additionally, collecting information about the typical states and conditions of the harbor would allow for better understanding of vessel movement. Current position, along with course and speed over ground give us an idea of a boat's trajectory, but understanding how harbor conditions explicitly impact vessel paths would allow for even more accurate interpolation. Another approach would be to include temporal features: that is, integrate time-related features in the trajectory prediction and merging processes. By incorporating temporal aspects, we would enhance the algorithm's ability to capture time-dependent vessel movements, which is essential for accurate tracking in a dynamic environment.

In conclusion, deploying our algorithm as a multiple target tracking system would benefit from a thorough evaluation using metrics such as trajectory accuracy, completeness/purity, and cluster quality. The metrics suggested would provide a comprehensive understanding of the algorithm's performance. To address practical challenges like data gaps, employing interpolation techniques, considering temporal features, and understanding harbor conditions are crucial steps. These recommendations aim to enhance the algorithm's robustness, ensuring accurate and continuous tracking even in dynamic and challenging maritime environments.

Appendix:

Table 1: ARI Values on set1.csv

Algorithm	**ARI given K**	**ARI for estimated K**
Trajectory Approximation + Custom Clustering*	.226	.229
Trajectory Approximation w/out Clustering	-Inf	.153
AHC	.118	Not implemented
Spectral Clustering	.057	Not implemented
Random Forest Classifier	.003	.011
DBSCAN	.018	Not implemented

Table 2: ARI Values on set2.csv

Algorithm	**ARI given K**	**ARI for estimated K**
Trajectory Approximation + Custom Clustering*	.65	.69
Trajectory Approximation w/out Clustering	.87	.960
AHC	.294	Not implemented
Spectral Clustering	.112	Not implemented
Random Forest Classifier	.024	.042
DBSCAN	.076	Not implemented

*These values for *Trajectory Approximation + Custom Clustering* were the highest possible values we were able to obtain on sets 1 and 2. Weights and thresholds in our implementation have shifted since then, and our current predictVessel.py performs worse on sets 1 and 2 than these values.