Clustering Algorithm Analysis of AIS Data

Brian Blakely

1. Introduction

The objective of this project is to correctly cluster ships based on their automatic identification system (AIS) data. In this project there will be two algorithms tested, one where the number of clusters k (the total number of ships) is given to the clustering algorithm, and the second having an unknown k. There are three datasets provided each of which are all in the same format. Each data point in the dataset is given in the form of a trajectory consisting of; a timestamp, latitude, longitude, course over ground (CoG), and speed over ground (SoG). The sampling of the data was taken over the same time period of the data over the span of multiple days in order to hopefully provide more consistent data. Two of the three datasets have labels provided, while the third is the held out dataset which the algorithms will be tested on. The evaluation metric being used for these algorithms is their adjusted random index score. This score is essentially measures how many points were correctly clustered together, not which ones were correctly clustered corresponding to the correct class.

2. Data Analysis

The data given is a trajectory with latitude and longitude coordinates, which makes it easily plottable. All figures mentioned in this section will be in the appendix as to ensure this section is readable. In figure 1, the latitude and longitude of all the vessels in the first data set are plotted, where a different color represents a different vessel. This is a good starting point to understanding the data set. Another useful interpretation is plotting it in 3 dimensions with the z-axis being time. This is shown in figure 2 for the same data set as in figure 1. In this graph the top of the z-axis is where time equals zero. It is easier for a human to get an idea on how to cluster the vessels with this graph as it gets rid of the nasty overlapping in figure 1. One last useful graph can be used to help interpret the data. Since these points are latitude and longitude points, they can be plotted over google maps as shown in figure 3. Figure 3 was plotted with gmplot and it is the plotting of the same data set, however not marking unique vessels with a different color. This perspective shows important locational features, such where ports might be or where vessels might be slowing down due to no wake zones. This actually shows 3 different ports, which is very useful to know. Knowing where boats are likely going to stop and where they are coming from makes clustering a bit easier. It also shows why some zones are extremely high in density. For example, the high density of boats on the left of the graph is because they are all coming from a different channel that leads out to the ocean. That point is where multiple different ports/harbors connect and lead out.

3. Data Preprocessing

The data given has already been partially preprocessed in order to make it easier to interpret. Part of this preprocessing removed the date from the timestamp of the AIS data, which adds complications. Without a date having the date it becomes difficult to interpret when two different ships are in the same location on different days or when one ship is following the exact same routine. It also makes it more difficult to process the data based on the frequency of the data received. At first it was proposed to predict the trajectory of a ship at a given point by computing the dissimilarity between trajectories between itself and points 60 seconds into the future. The idea was to predict the trajectory and find the most similar trajectory to the predicted and put them in the same cluster. From there it would recursively repeat until it has found k clusters or is out of data points to assign clusters. Ideally, they both would happen in the same iteration. Upon the implementation of this algorithm, the frequency of the data was found to be inconsistent which made this idea not hold up in practice. The frequency of the data would vary from seconds to minutes for the same VID's. If an arbitrary time in the future was selected, let's say 5 minutes, then there is a possibility that there is not a point within that time frame. If no time frame is given, then the computation time of the algorithm goes from minutes to hours. It can be noted that since the date is removed, the ships with lower frequency most likely means they were there for multiple days. In order to reduce the data and make it more consistent, points with speed less than or equal to .1 had their frequency reduced from every couple of seconds to once every minute. As for gaps in frequency, not much can be done when the data is unlabeled.

4. Algorithm Selection

The first algorithm tested was made from scratch. The premise of the algorithm was to compare difference in time, position, SoG, and CoG with heavy weights on the time and positional difference. In order to make this algorithm run there had to be a certain number of points selected in the future to look at. If it looked at all the points past itself, the algorithm would take millions of iterations to finish. This algorithm felt like the obvious solution to the task, however implementing it correctly is tricky. In the end, it is a slightly modified nearest neighbor algorithm, but took longer to run. The results were quite poor in the end and too much time was spent into the development of this algorithm.

Divisive Hierarchical clustering is a form of clustering which clusters data based on a similarity function. The outcome of this clustering algorithm depends almost entirely on the similarity function. In the case of AIS data, many different similarity functions can be used. Using the similarity of spatial location seems intuitive, however the performance looked a lot like DBscan with the haversine metric (discussed later on). A custom metric was tested, which was just the summation of each difference for each feature multiplied by some weight. In this summation equation, the time and position difference were heavily weighted whereas SoG and CoG were lightly weighted. The goal was to hopefully cluster points based on differences in an importance in the order of; time = position, SoG, CoG. On paper this would separate the points close together around the same time and then take into consideration how similar their speed and direction were, but in practice it performed worse than supervised learning. This idea still seems intuitive, the execution was probably just done poorly.

Supervised learning approaches were considered as well. In this project the algorithm constructed can use training data from the previous two datasets and predict the classes of the last dataset based on the two prior. Ideally this should be done as the data is in the same region and time frame as the last two datasets, so one can say the training data should help learn the new data. Random forest has worked well in the past for supervised learning, however it performed really poorly for this data. A neural network also seems reasonable for this data and since the dataset is so large, one can take advantage of early stopping and still get a confident result. Just like random forest, a neural network performed quite poorly in practice. In order to achieve good results with supervised learning methods for AIS data, the importance most likely comes down to the preprocessing of the data. In papers regarding GPS clustering and AIS clustering, most of them used incremental learning or supervised learning to train their algorithms. The most used algorithms seemed to be using a neural network on preprocessed data in the form of destination points. Then clustering ships based on the frequency of those points. This approach was considered as well, however it was difficult to implement.

DBscan was considered for the algorithm. DBscan works well for data with similar density. DBscan is extremely sensitive to the epsilon set. Epsilon is the threshold for which clusters are defined. For dataset 2, DBscan was getting up to .95 with epsilon at .18 on scalarized data, however for dataset 1 the highest it achieved was .2. It seemed to work well for smaller datasets, but the low score for a larger dataset is extremely inconsistent with a smaller one. DBscan can also be used with its metric as the haversine distance between two points. This gives a clustering based on only spatial locational data, which in the end is pretty poor for zones with high density.

Spectral clustering was the main algorithm considered. Spectral clustering is really good at spatial clustering as it considers a connected graph of points. It does a fairly good job separating points that intersect (relative to the other algorithms tested), however it still can yield some strange and undesirable results. This is shown in figure 4 and figure 5. Despite being better than average at separating intersects, there are still some very clear hiccups.

5. The Selected Algorithm

The selected algorithm ended up being spectral clustering. Spectral clustering has some tight restrictions on the data set. The first one being that the dataset must be a connected graph. This may make it tricky to implement. The most important part of spectral clustering is the affinity matrix. The affinity matrix is where the comparisons of the data are made. The top eigenvalues of the affinity matrix form the clusters. Setting the number of top eigenvalues equal to the number of clusters achieves optimal results. The affinity matrix can be precomputed, which probably should be done for data of this form, or automatically computed in scikit's implementation of the algorithm. The important parameters for spectral clustering were set as the following;

1. n_clusters= k (provided, if not then guessed)
2. random_state=1
3. affinity='rbf'
4. n_init=50
5. gamma=1

The n_clusters parameter is what indicates the number of clusters the algorithm should be finding. For the algorithm with unknown k, the number of clusters was guessed off the plotting of the data and assumed to be equal to 10. Ranom_state is just a preset randomization state set to make the results more consistent. Affinity is an important parameter. If affinity is set to precomputed, then an array X can be fed to the algorithm initialization which is assumed to be the precomputed affinity matrix. N_init is the number of times k-means is randomly ran and the best result is chosen. Gamma is a kernel coefficient for rbf, increasing or decreasing it will affect the affinity matrix which directly affects the output of the algorithm. Figure 4 and figure 5 are the comparisons of the affinity parameters rbf vs nearest neighbor, respectively.

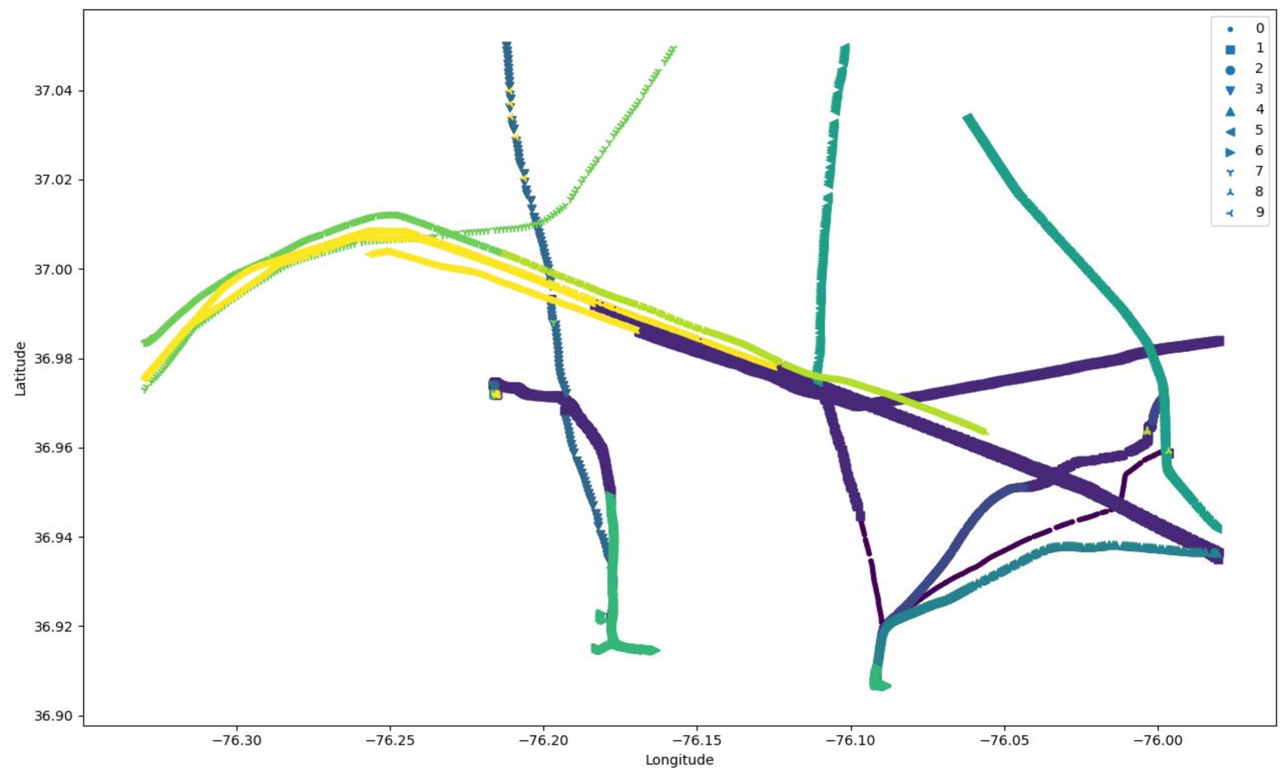

Fig. 4 Spectral Clustering affinity='rbf'

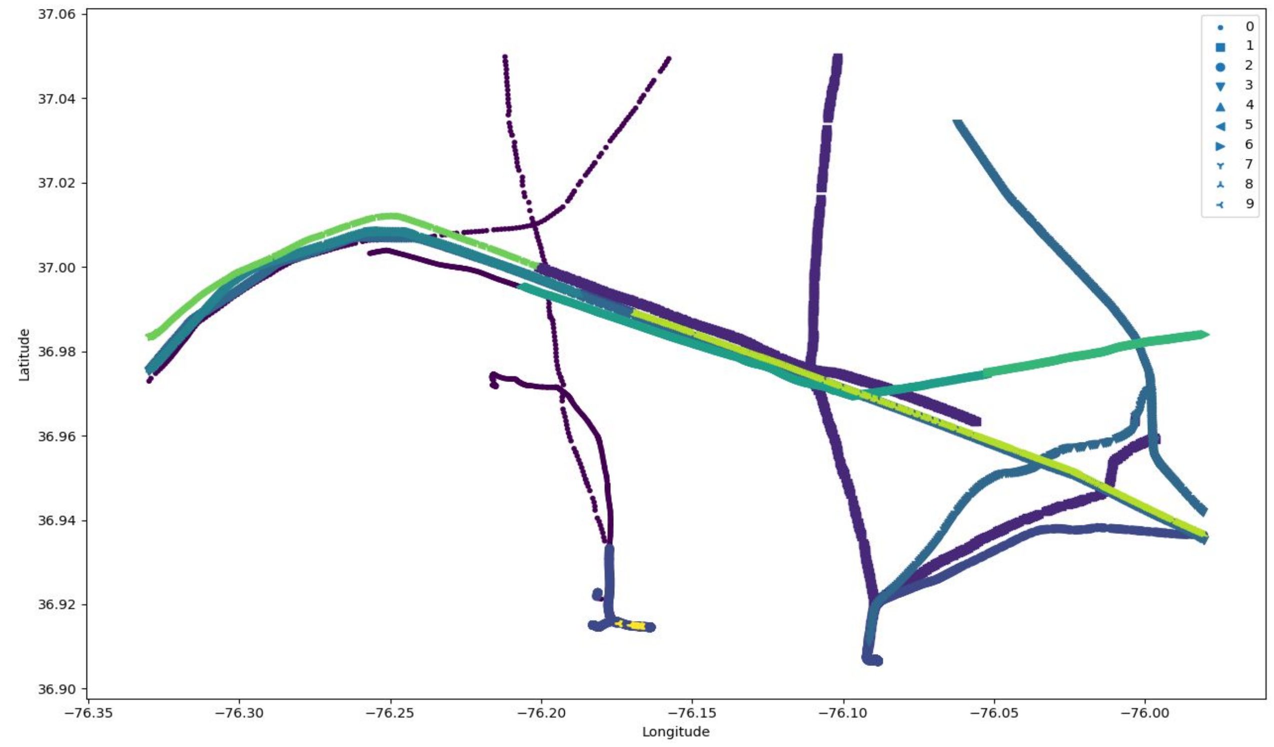

Fig. 5 Spectral Clustering affinity= 'nearest_neighbors'

These plots are on the dataset of unknown VID’s (dataset 3), however there are some clear mistakes being made. Despite that, these figures show a good comparison of the two parameters. Nearest neighbor is good at separating points where there is a lot of density. For example on the left of the graph where multiple lines are on top of each other. Since nearest neighbors looks at the closest n neighbors, it can differentiate high density areas. However, it is bad at differentiating scarce data points such as the leftmost line reaching the top of the graph in figure 5. It clusters those two lines together, despite coming from completely different locations. Figure 4 shows how setting the affinity to rbf solves this issue, however it does a poor job at differentiating high density zones. Just by looking at the results of figure 4, it is not as ideal as one would hope for and probably could be more accurately calculated by hand, but this algorithm had the best accuracy on the training data and gave the best looking result out of all the other graphs plotted by other algorithms.

6. Deployment Notes

In the actual deployment of this algorithm it would be a good idea to manually compute the affinity matrix. Doing so would ensure good results for clustering, however this paper won’t cover how to compute such an affinity matrix as it is its own topic. During the deployment of this algorithm, one should identify likely ports and anchoring points either manually or automatically. Doing this and filtering the data to reduce the frequency of these points makes the algorithm run faster and slightly increases the results. It is important to reiterate that the selection of k vessels was done manually, so just taking this implementation and using it for a different case with an unknown number of clusters will almost definitely result in poor performance.

Appendix

The first three figures are all over the first data set where the VID's are known.

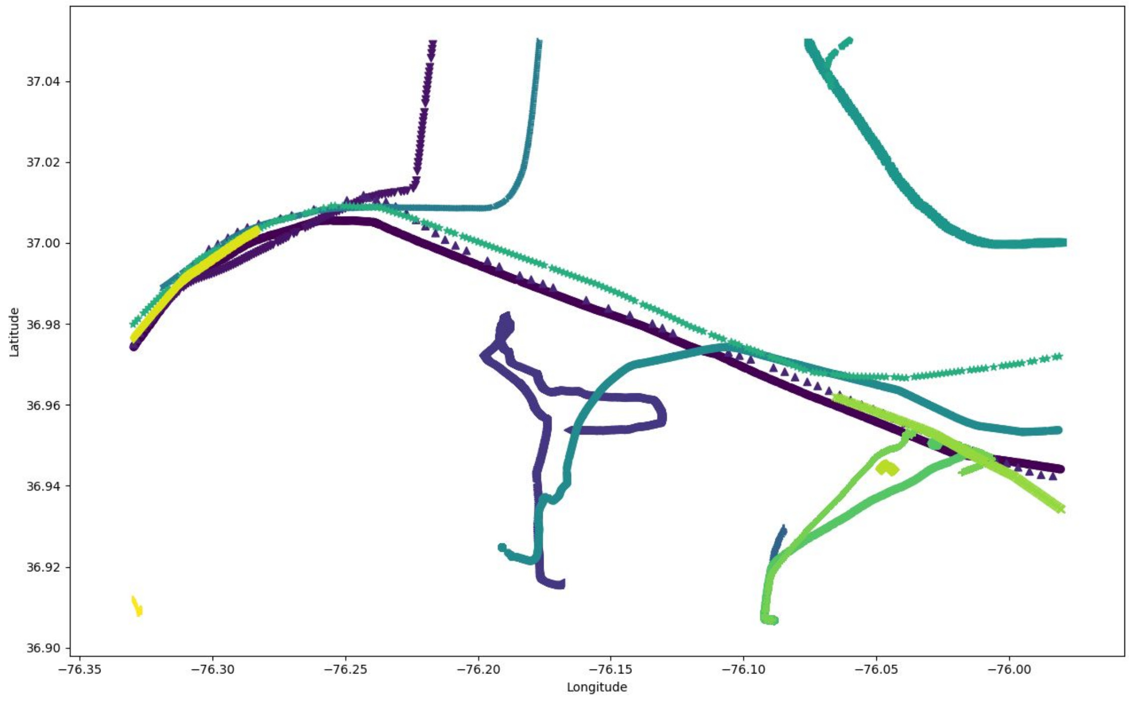

Fig. 1 Latitude vs Longitude for known VID's

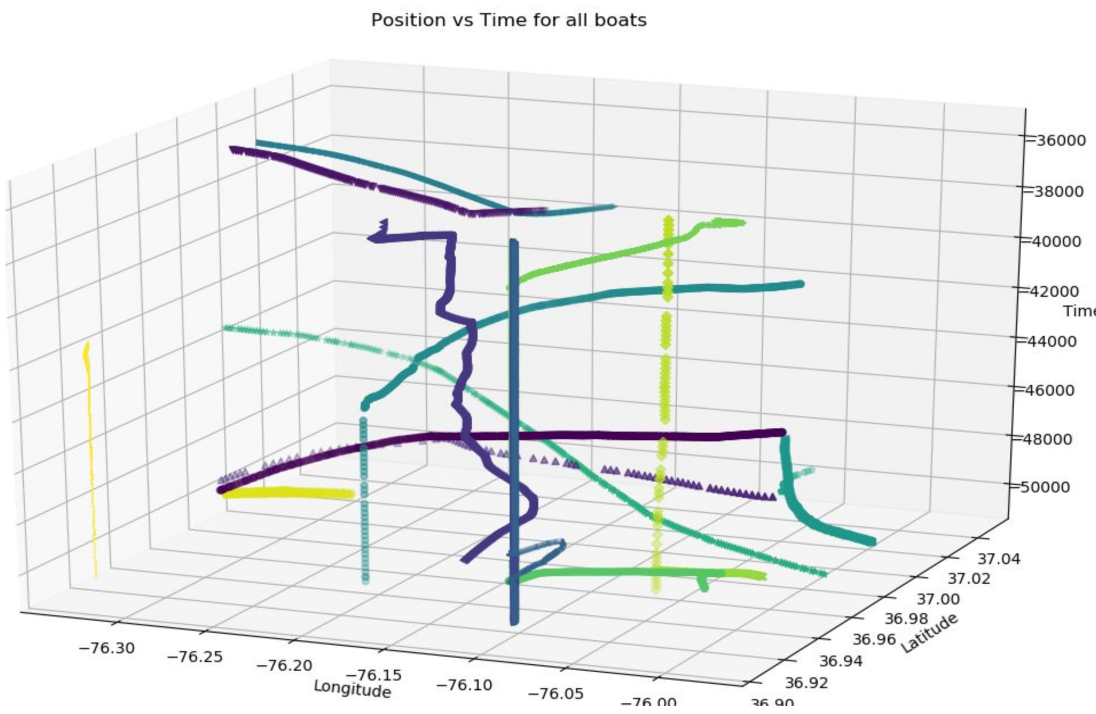

Fig. 2 Position vs Time for known VID's

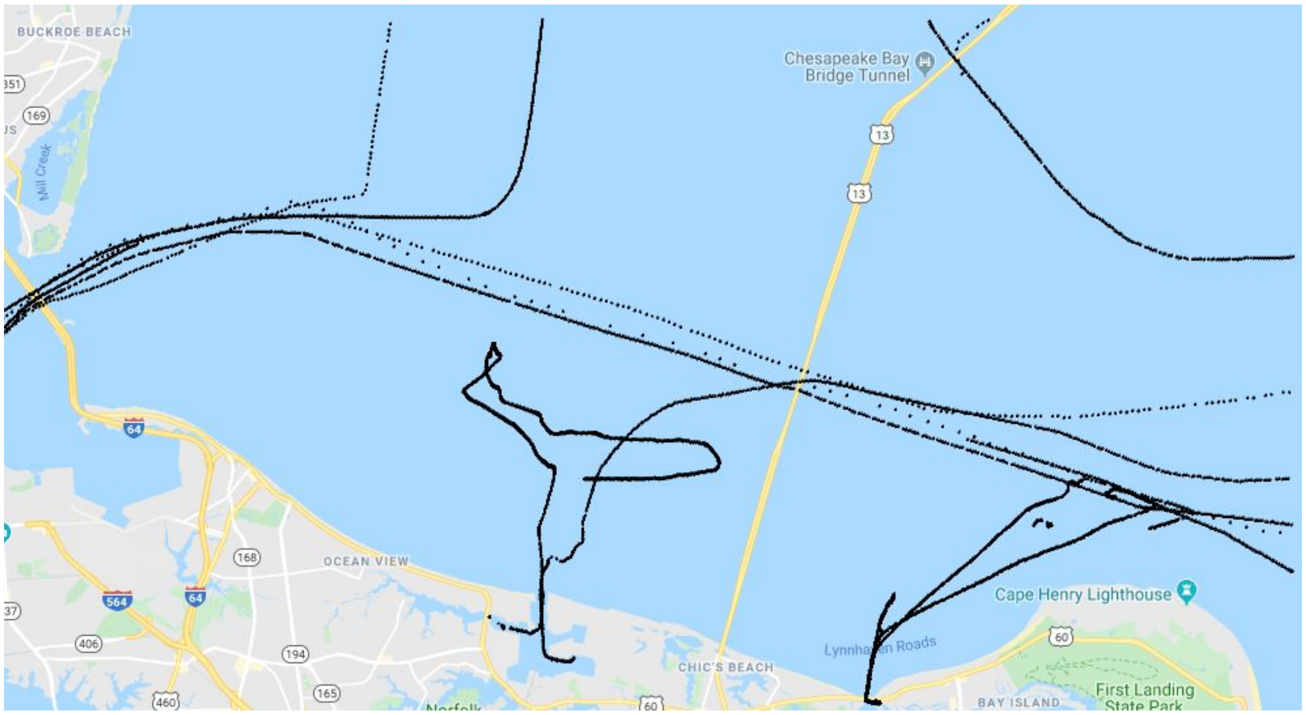

Fig. 3 Google Maps plotting of positions