Abstract

The explosion of both sensors and GPS-enabled devices has resulted in position/time data being the next big frontier for data analytics. However, many of the problems associated with large numbers of trajectories do not necessarily have an analog with many of the historic big-data applications such as text and image analysis. Modern trajectory analytics exploits much of the cutting-edge research in machine-learning, statistics, computational geometry and other disciplines. We will show that for doing trajectory analytics at scale, it is necessary to fundamentally change the way the information is represented through a feature-vector approach. We then demonstrate the ability to solve large trajectory analytics problems using this representation.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Large-Scale Trajectory Analysis via Feature Vectors

Jessica L. Jones, Benjamin D. Newton, Kyra L. Wisniewski,
Andrew T. Wilson, Melissa J. Ginaldi, Cleveland A. Waddell,
Ken Goss, Katrina J. Ward, Mark D. Rintoul,
Sandia National Laboratories

October 2020

1 Introduction

Position trackers and the data they generate are now all but ubiquitous. Aircraft, automobiles, trains, and ships carry them as a matter of course and sometimes as a matter of law [1, 2]. Every smart-phone and smart-watch generates timestamped position data. Even wildlife are fitted with tracking tags. As with other large bodies of data, trajectory data sets contain patterns of life that offer an invaluable basis for understanding what happens and planning for what may be. Just as with other large bodies of data, we need computers to handle the size and complexity involved. The real world problems that are driving the research in trajectory analytics include

1. What does that behavior tell us about the relationships between the actors, their motivations, and the value of the places they visit?

2. What collective behavior can we discern from a group of trajectories?

In air, sea, and land traffic management, these questions inform the design of road networks, air traffic control systems, and port operations. In biology and ecology, trajectory analysis allows scientists to track migration and dispersal patterns. In sociology, the identification of gathering points can reveal relationships and events [3]. In national security, the ability to characterize behavior (including anomalous behavior) can identify threats before they are realized.

Previous work on understanding motion has focused on rules-based expert systems types of approaches. The human understanding of trajectories was translated into rules that were then implemented as algorithms, allowing the computer to mimic the reasoning of a human expert, only faster. However, the proliferation of trajectory data has enabled machine-learning approaches that allow the computer to learn its own rules and develop its own approaches based on historical data. Many previous trajectory analysis algorithms originated in

computational geometry with the problem of aligning one curve to another. Since these measures are computed with respect to a pair of trajectories, they are most suitable for smaller data sets (see Section 2).

Machine learning algorithms that rely on spatial indices scale better to large data sets containing millions or billions of elements. The most efficient spatial indices require points of fixed dimension in a normed vector space.[1] In order to apply these algorithms to trajectory data, we need a fixed-dimension representation of a trajectory. Moreover, this dimension needs to be small enough to avoid the curse of dimensionality.

Our answer to this dilemma is to construct feature vectors over trajectories, then analyze those feature vectors instead of the original geometry. By constructing and selecting features, we gain the ability to indicate what factors are most important in our analysis. We gain increased explainability by tying our representation to descriptions of a trajectory instead of only its geometry. We also gain access to feature selection methods for cases where we may not know in advance which features are most informative.

1.1 Our Contribution

In this paper we explore the benefits and disadvantages of analyzing feature vectors created from trajectories. We emphasize the following points:

- Representing trajectories as feature vectors leads to asymptotically faster, more scalable algorithms than pairwise trajectory comparisons.
- Scalar features are rich enough to capture and characterize interesting behavior.
- Methods, such as distance geometry, for generating trajectory features allow efficient transformation-invariant comparison of trajectory geometry.

1.2 Definitions and Notation

In the rest of this paper we assume that a trajectory is composed of timestamped points in some vector space $\mathbb{V}$. This space is usually either Euclidean 3-space ($\mathbb{R}^3$) or the surface of the sphere[2] ($\mathbb{S}^2$). Points on the sphere are located using longitude and latitude. Trajectories can be parameterized either by time elapsed or distance traveled since their beginnings.

1.2.1 Definitions

Vehicle: A vehicle is an identified moving object for which we have timestamped position information.

[1] Spatial indices over non-normed vector spaces such as the ball tree and viewpoint tree exist but offer less advantage when compared to r-trees, kd-trees and their friends.

[2] While $\mathbb{S}^2$ is not a true vector space, it is close enough for our purposes. It has the notion of positions on the sphere, geodesics (segments of great circles), vectors between positions, and vector addition.

Trajectory Point: A trajectory point is a tuple comprising a position and a timestamp. Trajectory points often have other properties such as altitude and uncertainty.

Trajectory: A trajectory is a polyline whose vertices are trajectory points belonging to the same vehicle. These points are called the *vertices* of the trajectory and are ordered by increasing timestamp. In other work, the word *track* is sometimes used for the same concept.

Sub-Trajectory: A sub-trajectory is a trajectory that is a contiguous subset of some other trajectory $\mathbf{T}'$.

Head: A trajectory's head is its most recent vertex.

Tail: A trajectory's tail is its oldest vertex.

Duration: A trajectory's duration is the difference between the timestamps of its head and tail.

Total Distance: A trajectory's total distance is the sum of the distances between its vertices.

Straightness: A trajectory's straightness is a measure of how much it is like a geodesic between its head and tail.

1.2.2 Notation

$\mathbf{T}$ indicates a trajectory.

$N_{\mathbf{T}}$ is the number of vertices in trajectory $\mathbf{T}$.

$\vec{x}$ is a trajectory point.

$x(\vec{x})$ is the position component of trajectory point $\vec{x}$.

$t(\vec{x})$ is the timestamp component of trajectory point $\vec{x}$.

$\mathbb{V}$ is the space containing the position component of trajectory points.

$\|\cdot\|$ denotes the norm (distance function) for the space $\mathbb{V}$.

$\mathbb{D}$ is the space of timestamps. In practice, timestamps are real numbers. $\mathbb{D}$ and $\mathbb{V}$ are just for clarity in distinguishing positions and timestamps.

$\|\mathbf{T}\|$ denotes the total distance of trajectory $\mathbf{T}$.

p $(0 \leq p \leq 1)$ is the parameter we will use to index points along a trajectory.

$s_d(\mathbf{T}, p)$ $(0 \leq p \leq 1)$ is the point on trajectory $\mathbf{T}$ that is a fraction p along its total distance. Here the subscript d indicates distance.

$s_t(\mathbf{T}, p)$ $(0 \leq p \leq 1)$ is the point on trajectory $\mathbf{T}$ that is a fraction p along its total duration. Here the subscript t indicates elapsed time.

$S_d(\mathbf{T}, p_1, p_2)$ $(0 \leq p_1 \leq p_2 \leq 1)$ creates a sub-trajectory $\mathbf{T}'$ from the subset of $\mathbf{T}$ between parameter values p_1 and p_2. Specifically, $S_d(\mathbf{T}, p_1, p_2) = \{x \mid x = s_d(\mathbf{T}, p),\ p_1 \leq p \leq p_2\}$. As with $s_d(\cdot)$, the subscript d indicates parameterization with respect to distance traveled.

$S_t(\mathbf{T}, p_1, p_2)$ $(0 \leq p_1 \leq p_2 \leq 1)$ creates a sub-trajectory $\mathbf{T}'$ from the subset of $\mathbf{T}$ between parameter values p_1 and p_2. As with $s_t(\cdot)$, the subscript t indicates parameterization with respect to elapsed time.

$\phi(\mathbf{T})$ is a feature function that computes a single number that characterizes some property of a single trajectory.

$\Phi(\mathbf{T})$ is a feature *vector* function that computes several numbers that characterize some property of a single trajectory. Feature vectors can be concatenated to construct larger feature vectors.

$r(\mathbf{T})$ (short for *rectitude*) is the straightness of trajectory $\mathbf{T}$. We will define straightness in Section 3.2.2.

1.3 Outline

The remainder of this paper is organized as follows: Section 2 will talk about small scale applications such as various ways to compare trajectories. In Section 3, we cover feature-based representation of trajectories, rather than just a series of points. We then describe large-scale applications in Section 4 using the feature based representation covered previously. Finally, in Section 5, we cover future work and where trajectory analysis can go from here.

In this paper we do not address the problem of grouping unlabeled or partially-labeled points into trajectories. We also do not address *trajectory stitching*, where multiple shorter trajectories are identified as segments of one longer trajectory.

2 Examples of Previous Approaches

There have been a number of approaches that have been used in the past to compare trajectories [4, 5]. Most of these have had a focus on one-to-one types of comparisons, i.e. comparing trajectories $\mathbf{T}_1$ and $\mathbf{T_2}$, and generally do not scale well. These include (but are not limited to)

- Shape-Based Distance. There are two primary distances in this category. The Hausdorff distance for trajectories is the greatest of all distances from a point in one trajectory to the closest point in the other trajectory. The Discrete Frechet Distance [6] is similar to the Hausdorff Distance, but it takes into account the ordering of the points and looks at the maximum of the minimum distances between two points in $\mathbf{T}_1$ and $\mathbf{T}_2$. Here, the minimum distances correspond to following both ordered sets of points, starting with the first points and choosing to move forward in either one

of or both trajectories to the next point, such that the maximum distance is minimized. This is usually described as a "dog leash" distance between a person on one trajectory walking a dog on the other.

- Dynamic Time Warping Distance [4, 7]. This is a general category of distance measures that describes the minimum value of a series of steps that cause one trajectory to be mapped to another. This can be done by simply calculating the minimum distances required to move the points in one trajectory to points on another trajectory. More sophisticated measures can also be used to assign costs associated with bending and stretching a trajectory to align with another.

- Longest Common Subsequence (LCSS) Distances [8]. This approach attempts to find subsequences of the two trajectories that have matching positions within some distance ϵ. This approach has many difficulties if the trajectory points are sampled differently but is robust to noise.

These techniques and others have been used because they often correspond to a well-understood definition of distance for various problems in trajectory analytics. But they have two primary drawbacks:

- They only make sense when comparing one trajectory to another (rather than one trajectory to a large group of other trajectories).

- They are expensive to calculate. If $\mathbf{T}_1$ has M points and $\mathbf{T}_2$ has N points, then the computational complexity for each of these methods is $O(MN)$.

Because these and other similar methods have poor scaling performance for data sets with more than 10^6 trajectories, we focus our efforts on techniques that exhibit linear or log-linear scaling as a function of the number of trajectories. We also require that algorithms not scale quadratically in the number of points in the trajectories of interest.

3 Feature-Based Representation

Working with millions (or perhaps billions) of trajectories while maintaining computational efficiency and scalability requires new methods that do not rely on direct comparison of the points in two trajectories. Applying the techniques employed by the burgeoning field of machine learning is a natural choice. To use these methods each trajectory must be transformed into a fixed-length feature vector.

In this section we focus on numeric features derived from a trajectory and its points. Here we do not consider categorical features, nor features derived from trajectory metadata, as those tend to be domain-specific. For example, a ship's cargo type and an aircraft's model are very useful metadata, and have their place elsewhere in an analysis. However, we avoid using them here to ensure our analysis techniques can be applied generally.

The calculated numeric features describe each trajectory such that the succinct feature vectors can be analyzed in place of the verbose point data. The feature vectors are uniform in that they are all the same size and contain the same features in the same order, regardless of how many points the original trajectory contains. However, the values in each trajectory's feature vector will depend on its attributes. Given a trajectory $\mathbf{T} = \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_n}\}$ and a set of feature functions ϕ_i, we define the trajectory's feature vector f as

$$f_{\mathbf{T}} = \Phi(\mathbf{T}) = (\phi_1(\mathbf{T}), \phi_2(\mathbf{T}), \ldots, \phi_m(\mathbf{T})) \in \mathbb{R}^m \tag{1}$$

The dimension m of our feature space depends upon our choice of ϕ, but remains constant as we vary $\mathbf{T}$, making ϕ a hash function. This is key, since it allows comparison of trajectories regardless of the original trajectories' lengths.

One advantage of using a feature-based representation is that it allows the large body of machine learning techniques to be directly applied to trajectories. In addition, a reduction in the amount of memory allocated to each trajectory when performing a particular application can be expected. Depending on the application, only relevant features need be calculated and included, generally requiring only a fraction of the memory and resulting in a more efficient implementation of the particular application.

Another advantage of using feature vectors is a decrease in the storage footprint of each trajectory. A trajectory's general shape, for example, can be compactly represented with a feature vector containing just 10 numbers. Therefore, a database containing shape information for millions of tracks need only include 10 numbers per track, discarding the exact coordinates for hundreds, if not thousands of points along the track.

For each application, a different set of features may be considered. Generally, for a given application, features are chosen with the goal of defining "similarity" between trajectories. For some applications one may be interested in features related to a trajectory's shape, while for others, features may include distance features (such as the total distance traveled, and the distance between the start and stop points). Feature selection is an important part of obtaining good results. A poor choice of features can result in redundancy, information loss, or insufficient data for success.

One important potential weakness in the feature vector approach is that they do not represent categorical features well. They can be approximated by having individual features that correspond to each type categorical feature, and having a binary value, such as 0 and 1, corresponding to whether or not this feature is true of false. However, this results in a proliferation of variables and corresponding dimensions to the feature space and is generally best handled in a different way

There are myriad possible features that can be used in feature vector representations of trajectories. Section 3.3 gives a detailed but by no means comprehensive list of features. Most of these features are relatively simple to compute. Given that the computation of these features is necessarily independent across trajectories, they can also be computed in parallel, allowing very fast feature

vector computation, even for large data sets. Because of the independence from each other, we don't have quadratic dependence on the product of the number of points in every pair of trajectories, nor do we have to calculate a total number of trajectory distance measures that is quadratic in the total number of trajectories. Despite the simplicity of some of these features, they can provide a powerful description of a trajectory. For instance, feature vectors consisting of one feature for each coordinate of the start point of each trajectory can enable machine learning techniques to identify both common and anomalous locations of departure for a given set of trajectories.

Additionally, we can combine features of interest either analytically or through concatenation, enabling more complicated analysis. For example, the straightness of a trajectory can be found by dividing the distance between its endpoints by the distance traveled along the trajectory. Taking this one step further, a trajectory can be divided into several segments and the straightness values of these segments concatenated together to form a succinct representation of the shape of the trajectory. These distance geometry feature vectors are discussed in more detail in Section 3.2.2.

There is one more powerful analytic technique that emerges from the combination of the features and a spatial index like an R-Tree that allows objects with spatial extent. We can define features that have a range of values, and place the range in the R-Tree to allow a richer set of searches. For example, one feature could represent the time that a particular airplane was aloft. Or, such a range could be used to represent a value and the errors associated with the value.

One important thing to note that the choice of features by a person has both strengths and weaknesses. In most cases, it's a strength as there is a more direct understanding of how the computational techniques are working, i.e. there is a high-level of explainability. However, there is always the risk that having features that are strictly chosen by analyst will introduce unwanted biases and assumptions that cause the results to be worse than if the computer had the ability to choose its own combinations of features.

Once feature vectors are created, the comparability of feature vector representations enables both supervised and unsupervised machine learning techniques to be applied for prediction, identification of normalcy, pattern detection and anomaly detection. Facilitating the automation of machine learning is especially valuable when working with large trajectory data sets that are difficult, if not impossible, for humans to view, let alone analyze.

3.1 Collaborative Behavior Features

Features can be defined and combined such that the joint behavior of two or more tracks can be identified and analyzed. This section defines and describes some of these collaborative behavior features.

3.1.1 Full Co-travel Feature Vector

Here we construct a feature vector intended to compress a trajectory into a temporospatial representation, enabling automated comparison to find trajectories that *co-travel*; that is, trajectories whose vehicles follow a similar path at approximately the same time.

To create our feature vector, we first select N, the number of points to be sampled from each trajectory. It should be noted that increasing the number of points will increase the accuracy of our compressed temporospatial representation, but it will also increase the storage footprint of each compressed trajectory. Once N has been selected, we define separate feature functions ϕ_i to extract points spaced evenly in time along the trajectory. These feature functions look like this:

$$\phi_i(\mathbf{T}) = s_d\left(\mathbf{T}, \frac{i}{N-1}\right), \quad i = 0, 1, ..., N-1. \tag{2}$$

With a slight abuse of notation, we then have the full co-travel feature vector function:

$$\Phi(\mathbf{T}) = (\phi_0(\mathbf{T}), \ldots, \phi_{N-1}(\mathbf{T})) \tag{3}$$

Since the extracted trajectory points $s_d(\cdot)$ are multidimensional, we typically just append their components (coordinates plus timestamp) to the feature vector one after another.

3.1.2 Rendezvous Feature Vector

It is possible that our interest does not lie in comparing trajectories during their entire voyage. Instead, suppose we wish to identify trajectories that are together only at the start, or the end, or for some interval in the middle of their journeys. To this end, we consider an extension of the hash function in (3). In this extension, we consider N equally-spaced points between p_1 and p_2 in the temporal parameterization. For instance, if $p_1 = 0.3$ and $p_2 = 0.7$, this is equivalent to saying that we will only consider the trajectory after 30% and before 70% of its total travel time has elapsed. We construct the rendezvous feature vector just as we did the full co-travel feature vector but modify the individual ϕ_i as follows:

$$\phi_i(\mathbf{T}) = s_d\left(\mathbf{T}, p_1 + \frac{i}{N-1}(p_2 - p_1)\right)t, \quad i = 0, 1, ..., N-1. \tag{4}$$

As we would expect, Eq. 4 is the same as Eq. 2 when $p_1 = 0$ and $p_2 = 1$.

3.2 Shape Detection Features

3.2.1 Boxes

The goal of this feature, called the "boxiness" feature, is to detect trajectories making repetitive 90° turns, resulting in the formation of boxes or a stair-stepping pattern at any orientation along a trajectory. When analyzing motor vehicle tracks, a high value for "boxiness" could indicate travel in a metropolitan area. In the case of maritime traffic, repetitive 90° turns are unusual and may indicate the vessel is performing a survey or patrol task.

We start by defining $b_i \in [0, 360)$ for $i \in \{0, 1, \ldots, I-1\}$ to be the bearing of the vessel (in degrees) as it travels from $\vec{x}_i$ to $\vec{x}_{i+1}$. Mathematically, b_i is the angle of the vector $\vec{x}_{i+1} - \vec{x}_i$ taken clockwise from the positive y-axis such that 0° is due north and 90° is due east.

We then bin the bearings by $\theta \in \{0°, 1°, \ldots, 359°\}$, such that

$$n(\theta) = \sum_{i=0}^{I-1} \mathbf{1}_{[\theta,\theta+1)}(b_i) \cdot \|x_{i+1} - x_i\|, \tag{5}$$

where $\mathbf{1}_{[\theta,\theta+1)}$ is the indicator function

$$\mathbf{1}_{[\theta,\theta+1)}(x) = \begin{cases} 1 & x \in [\theta, \theta+1) \\ 0 & x \notin [\theta, \theta+1) \end{cases} \tag{6}$$

In (5), instead of adding a single count for each new bearing, we are adding the length of the trajectory between the neighboring points that yielded the bearing θ. By doing this, we give greater weight to bearings that represent a greater distance traveled. To preserve our ability to compare trajectories regardless of their total length $\|\mathbf{T}\|$, we then normalize to get

$$\hat{n}(\theta) = n(\theta) / \|\mathbf{T}\|. \tag{7}$$

One could now take a 90 degree circular convolution of the normalized histogram $\hat{n}(\theta)$ as follows:

$$\sum_{\theta=0}^{359} \hat{n}(\theta) \cdot \hat{n}((\theta + 90) \bmod 360). \tag{8}$$

We can expect that trajectories moving in perfect rectangular patterns would have large values for this convolution. However, consider trajectories that move in all directions, creating a nearly uniform distribution of bearings. Figure 1 shows this for a tug boat traveling the winding path of the Mississippi River. Unfortunately, if we use (8) as a score to detect box-like patterns in trajectories, these types of trajectories can also score high despite not traveling in interesting 90° angular patterns.

Since we are not interested in trajectories that do not form boxes, we will make two alterations to the traditional convolution formula so it will only score

highest for trajectories with four distinct bearing peaks at 90° offsets, e.g. 5°, 95°, 185°, and 275°; or 72°, 162°, 252°, and 342°. Doing so will capture boxes at any orientation. First, we collapse our bins into "quartets" $q_{\hat{\theta}}$ such that

$$q_{\hat{\theta}} = \hat{n}\left(\hat{\theta}\right) \cdot \hat{n}\left(\hat{\theta}+90\right) \cdot \hat{n}\left(\hat{\theta}+180\right) \cdot \hat{n}\left(\hat{\theta}+270\right), \tag{9}$$

where $\hat{\theta} \in \{0°, 1°, \ldots, 89°\}$.

By using a product instead of a sum, we are introducing the stricter requirement that there is travel along all four edges of our box. For example, if there is travel in the direction of 30°, 120°, and 210°, but no travel in the direction of 300°, then our 30° quartet q_{30} will be zero. We especially desire this strictness when dealing with larger data sets so that only the most anomalously box-like patterns will have high scores.

Second, instead of summing all of the quartets $q_{\hat{\theta}}$, we find the maximum:

$$\theta_{\max} = \underset{\hat{\theta}\in\{0°,1°,\ldots,89°\}}{\arg\max} q_{\hat{\theta}}. \tag{10}$$

This will prevent trajectories with uniform bearing distributions from scoring high. To allow for slight deviation from a perfect box, we will sum over a 5° window centered at this maximum. Lastly, we normalize over the largest possible score, $(0.25)^4$. Thus, we have our hash function for the feature we call "boxiness":

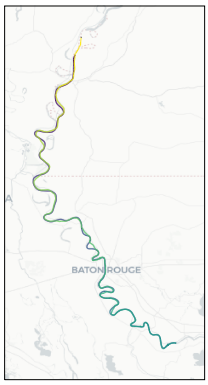
$$\phi(\mathbf{T}) = \frac{1}{0.25^4} \sum_{\hat{\theta}=\theta_{\max}-2}^{\theta_{\max}+2} q_{\hat{\theta}} \in [0,1]. \tag{11}$$

3.2.2 Features Created using Distance Geometry

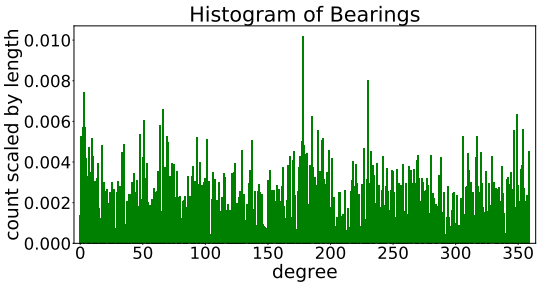
To compare the shapes of trajectories regardless of their orientation, position or size we need a concise representation that is invariant under rigid transformation (rotation, translation, reflection, and uniform scale). We can use distance geometry [9] to construct such a representation. Using the principles of distance geometry, we construct a feature vector describing shape by calculating the "straightness" of smaller and smaller segments of a trajectory. Refer to Algorithm 1 for a full description.

We begin by selecting a depth K. Higher values of K yield a more detailed description of shape at the cost of a longer feature vector. For each value $k \in \{1 \ldots K\}$, we divide a trajectory into k segments of equal distance and compute their straightness. The distance geometry feature vector collects all of these straightness values.

Now we need a definition for "straightness". Intuitively, the straightness of a sub-trajectory is a measure of how closely that sub-trajectory follows a geodesic between its endpoints. A higher straightness value indicates a path that hews more closely to that geodesic. By contrast, a trajectory with positive length



(a)



(b)

Figure 1: (a) *Leonard L. Whittington* (MMSI: 367702250), a tug boat traveling south and then north along the Mississippi River. Its winding path means that the vessel has bearings of all degrees, leading to a somewhat uniform histogram of bearings (see (b)) as calculated by (5). Because a circular 90° convolution of the histogram in (b) will not require that a single set of peaks is separated by 90° intervals, (8) will score in the top 15% of trajectories despite not creating a box shape while traveling. However, if we use the modified quartet convolution in (11), this trajectory falls down to below the 20-th percentile because it lacks four distinct peaks.

that ends where it began will have a straightness value of zero regardless of its total length.

We define straightness using the notions of *straight-line distance* and distance along a trajectory. In the following definitions we use the parameterization of a trajectory with respect to distance.[3] The straight-line distance $d(\cdot)$ for a trajectory is the distance between its endpoints. Since we will be working with sub-trajectories, we express the straight-line distance for a sub-trajectory $\mathbf{T}' \subset \mathbf{T}$ in terms of points on the parent trajectory $\mathbf{T}$:

$$d(\mathbf{T}') = \|s_d(\mathbf{T}, p_1) - s_d(\mathbf{T}, p_2)\| \quad \text{where } \mathbf{T}' = S_d(\mathbf{T}, p_1, p_2) \tag{12}$$

The distance along a trajectory was defined as $\|\mathbf{T}\|$ in Section 1.2.2. Given these two definitions, straightness $r(\mathbf{T})$ (short for *rectitude*) is the ratio between the end-to-end distance and the distance traveled along the trajectory. Observe that since straight-line end-to-end distance cannot exceed the distance traveled along the trajectory, the value of r is always in the interval $[0, 1]$.

$$r(\mathbf{T}') = \frac{d(p_1, p_2)}{\|\mathbf{T}'\|}, \quad \mathbf{T}' = S_d(\mathbf{T}, p_1, p_2) \tag{13}$$

The last piece of our algorithm is the specification of entities for which we compute straightness. These entities depend on the specific depth $k \in 1 \ldots K$. To divide a trajectory into k subsets of equal length, we place $k+1$ values evenly along the interval [0, 1] and extract a sub-trajectory between each point and its successor. This is illustrated for $K = 4$ in Figure 2.

The complete feature vector comprises the straightness values for the segments for each depth level, ordered first by depth and second by position along the trajectory. The distance geometry feature vector for a given depth value K will have $\frac{1}{2}K(K+1)$ elements.

Algorithm 1 contains pseudocode for constructing a full distance geometry feature vector at a given depth K. An example of this feature vector computed on two separate maritime trajectories can be seen in Figure 3.

We note that trajectories that do not move at all lead to division by zero when computing straightness values. We filter these trajectories out. We justify this decision by observing that distance geometry is meant to identify trajectories with similar shape. Trajectories that do not move have trivially similar shapes and do not require complex computation to identify.

[3] If we sample by time, division by zero can occur at points where the vehicle is stationary for long periods of time.

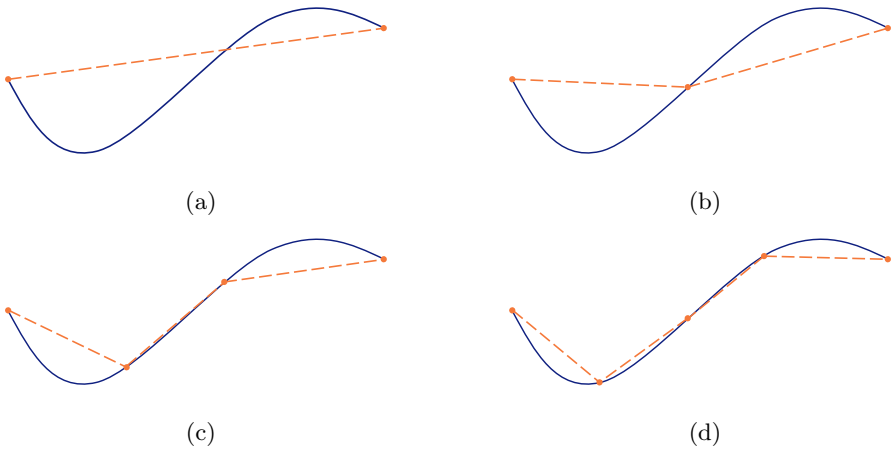


Figure 2: Here we show how our distance geometry method with depth $K = 4$ is used to describe the shape of an arbitrary curve (solid navy blue). In each figure, we find points (which we call *control points*, shown in solid orange) that divide the length of the curve into (a) one part, (b) halves, (c) thirds, and (d) fourths. Neighboring orange points are connected (dashed orange lines) and we divide the length of each dashed orange line by the length of the navy blue sub-trajectory that lies between the endpoints of the dashed orange line. The (a) one, (b) two, (c) three, and (d) four resulting values are then appended to create a 10-dimensional feature vector.

Algorithm 1: Create a feature vector using distance geometry.

Input: Trajectory $\mathbf{T}$ and depth K.

`// See Section 1.2.2 for definitions of` $\|T\|$ `and` S_t`, and Eq. 12 for definition of` d

let `featureVector` = (empty feature vector);

for `numControlPoints` from 2 to $K + 1$ **do**

  let `controlPoints` = (empty list);

  let `controlPointIncrement` = $\frac{1}{\texttt{numControlPoints}-1}$;

  `// Compute fractions for the endpoints of the sub-trajectories at this depth` k

  for `i` from 0 to `numControlPoints` − 1 **do**

    let `controlPointFraction` = `i` × `controlPointIncrement`;

    append `controlPointFraction` to `controlPoints`;

  end

  `// Compute straightness for each sub-trajectory`

  for `j` from 0 to `numControlPoints` − 2 **do**

    let `trajectorySegment` = $S_t(\mathbf{T}, \texttt{controlPoints}[j], \texttt{controlPoints}[j+1])$;

    let `straight` = d(T(`trajectorySegment`, 0), T(`trajectorySegment`, 1));

    let `total` = ‖`trajectorySegment`‖;

    append $\frac{\texttt{straight}}{\texttt{total}}$ to `featureVector`;

  end

end

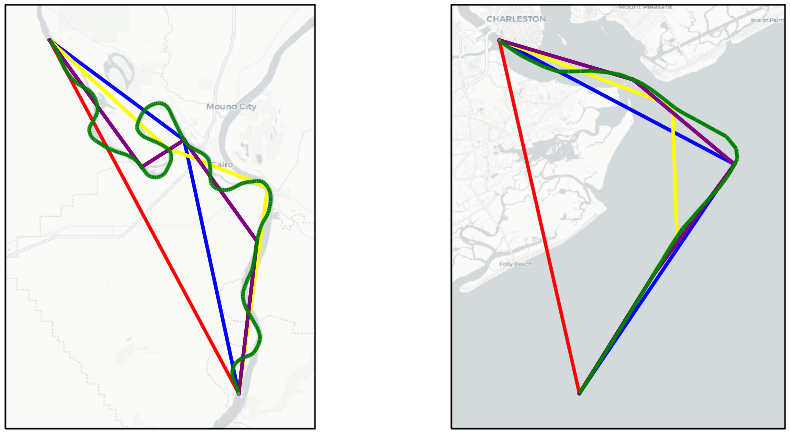
Output: `featureVector`, a $\frac{K(K+1)}{2}$-dimensional feature vector representing the shape of $\mathbf{T}$

3.3 A Rough Taxonomy of Numeric Features

Grouping features according to their characteristics makes it easier to select a relevant set for a specific analysis question. While a full taxonomy is far beyond the scope of this paper, we can enumerate categories for some possible features according to the nature of the description each one provides. Note that there is some overlap between some of the categories so it is possible for some features to arguably belong to multiple categories.

3.3.1 Kinematic Features

Kinematic features describe an object's motion independent of the forces that cause it to move. Statistics of a trajectory's curvature including radius of gyration, total turning, total winding, and the magnitudes of fluctuation in altitude (when available) are kinematic features.



(a) $[0.5784, 0.4657, 0.7597, 0.6630, 0.4505, 0.9172, 0.8944, 0.2790, 0.7130, 0.9011]$

(b) $[0.6636, 0.9976, 0.9291, 0.9999, 0.7223, 0.9644, 0.9999, 0.9965, 0.9339, 0.9665]$

Figure 3: Following the distance geometry method described in Section 3.2.2 with depth $K = 4$, a 10-dimensional feature vector is created for each maritime trajectory above. Each vessel's trajectory as it travels south to north is shown in green, with red/blue/yellow/purple lines connecting the points along the trajectory that divide it into one/two/three/four equal pieces, respectively. Each number in the feature vector represents a straightness calculation for the trajectory between two endpoints of a line segment, as detailed in (13). For instance, the first value of our feature vector was computed using the distance traveled along the green trajectory curve divided by the length of the red line. Next, the trajectory was divided into halves, with the blue line segments connecting the midpoint to the trajectory's endpoints. The distance traveled along the trajectory during the first half of its journey divided by the straight-line distance between the trajectory's start and midpoint (lower blue line) gives us a sense of straightness for the first half of the trajectory. For (a), this is 0.4657, indicating a fair amount of deviation from the most direct path. However, (b) has a straightness of 0.9976, which follows from our figure above: (b) does not deviate much in the first half of its journey from the straight line distance, and therefore has a straightness value very close to one for that portion of its trajectory. An analogous calculation is done for the second half of the trajectory to obtain the third point in our feature vector, and calculations along thirds and fourths yield the rest.

3.3.2 Temporal Features

Temporal features describe *when* some event of interest took place. The timestamps of the first and last points in a trajectory are temporal features, as are the times at which maxima/minima/modes of speed and altitude are attained. Temporal features help us identify trajectories that act simultaneously or in sequence with one another. They also provide clues to the segmentation of trajectories into periods of coherent behavior.

3.3.3 Shape Features

Shape features describe some aspect of the trajectory's geometry. A trajectory's winding number and the numeric properties of its convex hull such as area, perimeter, and eccentricity are shape features. A distance geometry feature vector is a whole group of shape features. Trajectories with similar shape typically exhibit similar behavior.

3.3.4 Geospatial Features

Geospatial features describe *where* a trajectory is observed. A trajectory's origin point and destination point are geospatial features, as is the distance between a trajectory and some point or region of interest. Geospatial features are useful for identifying coordinated behavior including co-travel, rendezvous, and interaction with points of interest.

Features Categorized

Feature Name	Categories
Nearest distance to a fixed point/region	Geospatial
Place where first seen	Geospatial
Place where last seen	Geospatial
Centroid of trajectory	Geospatial
Total distance traveled	Kinematic
End-to-end distance	Kinematic
Straightness	Kinematic
Total curvature	Kinematic
Total amount of turning	Kinematic
Average heading change	Kinematic
Average speed	Kinematic
Maximum altitude	Kinematic
Magnitude of fluctuations in altitude	Kinematic
Most common speed	Kinematic
Most common altitude	Kinematic, Geospatial
Loiter ratio	Kinematic
Shape (distance geometry)	Shape
Convex hull area	Shape
Convex hull eccentricity	Shape
Convex hull perimeter	Shape
Divergence from a given track	Shape
Divergence from a given shape	Shape
Start and End time	Temporal
Duration	Temporal
Time when nearest to a fixed point/region	Temporal
Time where first seen	Temporal
Time where last seen	Temporal

Table 1: Possible features labeled with their categories. As with other machine learning tasks, selecting and combining features to reveal desired characteristics of the data is a critical part of the analysis process.

3.4 Reduced Information Representation

It is worth noting that feature vectors are not just an enabling capability for trajectory analysis. They provide traction to machine learning algorithms by exposing specific characteristics that we care about in a body of trajectories. As such, each feature vector acts as a concise, compact summary of a trajectory. Moreover, once features have been chosen, a feature vector's size is constant.

This drastically reduces memory requirements compared to the original trajectories, whose size depends on sampling rate, metadata, and duration. For any reasonable number of features, a feature vector is likely to occupy the same amount of memory as only one or two points from its corresponding trajectory.

Feature vectors also raise the possibility of offloading some of our work to a database. As a result of the growing influence of geographic information systems (GIS), many production-quality databases now incorporate geometric data types such as points, polylines and polygons as first-class entities along with spatial index structures that accelerate operations on geometry. While many of these indices are limited to two dimensions (reflecting their expected use for geographic data), indices for multidimensional vectors are starting to appear. Notwithstanding this limitation, spatial data types in databases are a powerful way to delegate management of large data sets while we focus on analysis.

4 Applications

4.1 Machine Learning

Supervised learning refers to a machine learning method where patterns and information can be extracted from data by learning from historical data. For some types of trajectory data, such as commercial vehicles and some road networks, historical data can tell us what typical behavior should look like, making it easier to build models to detect anomalies. It can also give us patterns to understand current observed behaviors before the trajectory is complete. For example, in our prediction algorithms we calculate the probability that a vehicle will be in a set of locations. Given a sub-trajectory, the algorithms look at where previous vehicles of the same type that have shared the same or similar sub-trajectory have come from and gone to and calculate a probability based on similarity that the new sub-trajectory will share the same paths. Similarly, neural net approaches, often used to predict the next step in a trajectory, work by processing large amounts of historical trajectory data and creating a model for which it expects future trajectories to follow. While supervised learning can be powerful for prediction and anomaly detection, it requires a large amount of data and a non-trivial amount of time to train an accurate and effective model.

Alternatively, unsupervised learning is a type of machine learning that detects patterns within data without any training data or labels. There are multiple techniques of unsupervised learning that could be used to detect patterns within trajectories. For the work discussed in this paper, SVMs have been considered and PCA was used for dimensionality reduction, but the primary focus is on clustering through unsupervised learning. We will show that this technique excels at pattern and anomaly detection in large datasets with minimal a priori information.

Clustering is the grouping together of similar data points based on their features. Trajectory data sets often contain thousands or even millions of trajec-

tories. Manually identifying uniquely shaped trajectories in all but the smallest data sets would be infeasible. The ability to autonomously cluster trajectories based on their shape enables quick discovery of uniquely shaped trajectories, as well as groups of trajectories that have extremely similar shapes. Since we can't assume trajectory clusters will be spherically shaped, only density-based clustering algorithms are considered. More specifically, box-DBSCAN (a slightly modified version of DBSCAN [10] that will be described shortly) and HDBSCAN [11] are used for our experiments. Our goal is to determine which clustering algorithm will best be able to cluster trajectories based on shape.

DBSCAN is a density-based clustering algorithm that takes in two parameters, *search_radius* and *min_cluster_size*. It defines a cluster as a group of *min_cluster_size* points or more within a radius of *search_radius*. The algorithm begins by picking a random point and examines the number of points within its *search_radius*. If there are *min_cluster_size* points or more a cluster is created and the points are labeled in that cluster, else the point is labeled as noise. We use a modified version we call box-DBSCAN that is identical except that rather than using a *search_radius*, an n dimensional rectangle is used for determining nearby points. This is simpler to use when the scales of the feature space vary greatly in different dimensions and we can choose the size of the n-dimensional rectangle to vary correspondingly in each dimension. The algorithm continues until every point is labeled in a cluster or as noise.

HDBSCAN transforms DBSCAN into a hierarchical algorithm and takes in *min_cluster_size* and *min_samples* as parameters. The algorithm uses a core distance, which is defined as the distance to the *min_samples* nearest neighbor, to create the hierarchy. This generates a dendrogram of clusters based on the minimum core distance between points and the algorithm then selects which clusters to keep based on their stability defined below:

$$\sum_{point \in cluster} (\lambda_{point} - \lambda_{birth})$$

Where λ is the reciprocal of the core distance, λ_{point} is the lambda value when the point falls out of the cluster and λ_{birth} is the lambda value when the cluster is formed. Working from the bottom of the dendrogram up, if the sum of the children's stability is greater than the parent's stability, the children are selected and the parent's stability measure is updated to that of the children. Otherwise, the parent is selected, and all descendants are unselected.

In order to test which clustering algorithm performs best on trajectory data, experiments were first performed on non-trajectory ground truth data. The purpose of this experiment is to work with smaller data sets to quickly simulate many different situations and observe the outcomes. The feature vectors of the data are 10-dimensional and scale between 0 and 1 to resemble our real trajectory data. For well-defined and separated data, a *min_cluster* size of 25 was used for HDBSCAN and DBSCAN, a *search_box* size of 0.2 for was used for DBSCAN, and *min_sample* sizes of 1,5,10, and 20 was used for HDBSCAN. For overlapping and noisy data, a *min_cluster* size of 5 was used for DBSCAN

and HDBSCAN with the same *search_box* and *min_samples* sizes as in the separated data. To assess how well the clustering algorithms preform, the Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) scores were used, comparing the clustering labels to the ground truth labels. In addition, the results were visually compared by graphing the ground truth data and the clustered data in two dimensions through TSNE – a dimensionality reduction algorithm.

In summary, the results showed that both HDBSCAN and DBSCAN generate nearly perfect clusters when the clusters are well defined and separated. As the clusters begin to have a small overlap DBSCAN performs slightly better than HDBSCAN and with a large overlap (large amount of noise) HDBSCAN performs better than DBSCAN. Since features generated from large trajectory datasets are unlikely to be well separated, we suspect HDBSCAN will perform slightly better in general.

What defines the "best" clustering is subjective to the user. For this experiment, we used the ARI and AMI scores to help evaluate the clusterings – the higher the score the better the clustering was assumed to be. However, there existed a few cases where a clustering correctly identified all the clusters but did not receive the highest ARI or AMI score. In one instance, 6 out of 6 clusters were identified but several of the points in each cluster were incorrectly labeled as outliers (noise). In this case, the ARI and AMI scores were lower than for another clustering which only identified 4 clusters out of 6 but labeled fewer outliers. Given situations like this, we leave the user to decide whether it is more important to separate points from the noise or to identify all of the unique clusters.

Many features can be used to cluster trajectories. One way we cluster trajectories in this work is using feature vectors created using distance geometry with depth $K = 4$, as described in Algorithm 1. The depth 4 calculation will return a 10-dimensional feature vector where each value is between 0 and 1. The rest of the experimental process is the same as above for the non-trajectory data. However, creating the ground truth trajectory data sets is a challenge in its own right. We approach this by finding 3-4 reference trajectories within a month of flight trajectories. First, the depth 4 distance geometry feature vector of one unique trajectory is computed as a reference. Then, a month of trajectories are scanned to find all trajectories whose depth 4 distance geometry feature vectors are within an ϵ threshold of the reference feature vector. Each trajectory within the threshold of the unique trajectory is added into its cluster and removed from the original data set to avoid double counting. This is done for each unique trajectory until all clusters are formed. Noise is created by randomly selecting trajectories that are left in the original data set.

While experimenting with flight trajectories, the focus was on two data sets: one with clearly defined clusters and low noise and another with overlapping clusters and a lot of noise. For the well-defined clusters, on average both HDBSCAN and DBSCAN perform close to perfectly. However, for the noisy overlapping clusters HDBSCAN performs slightly better overall and scores higher ARI and AMI. DBSCAN and HDBSCAN both find the same number of clusters,

but HDBSCAN does a better job of identifying the smaller clusters. DBSCAN did recognize the tips of the smaller clusters but as the clusters approach the larger ones the points are labeled as noise. This may be due to the fact that the smaller clusters take on non-uniform shapes rather than that they contain fewer points. It should also be noted that HDBSCAN is more adaptable to changes in data set size. The first data set contains around 10,000 points; the larger data set contains around 4 times as many points. HDBSCAN performs equally well on both data sets using the same parameters, but DBSCAN requires large variation in the $search\_box\_size$ to obtain good results.

4.2 Prediction

If an object's motion is captured only for a short period of time, can we use historical information to fill in the unknown qualities of the past part of the trajectory and predict its future qualities? These are the motivating questions behind trajectory prediction. For example, given an observed trajectory one possible goal is to correctly predict its origin and destination locations. Alternatively, another relevant problem is correctly predicting where an observed trajectory will be in specified amount of time, such as predicting where an aircraft will be 15 minutes in the future. We focus on these problems by using historical data to inform predictions. Namely, we ask what historical trajectories have a path similar to the given observed trajectory. Note that there are two different scenarios that can occur:

1. There is at least one historical trajectory that is suitably aligned to the given observed trajectory. In this case, it is possible to develop weights based on the alignments such that the results can be ranked before they are used.

2. There are no historical trajectories that are similar to the given sub-trajectory. This is an important null result as it indicates the current trajectory is following a path that does not have a historical match.

The output of a trajectory prediction algorithm can take multiple different forms. For example, as a single origin/destination pair with the highest likelihood of correctness. Alternatively, a ranked list can be returned like a typical internet search for evaluation by an expert analyst.

There has been some previous work on the problem of trajectory prediction. The related works mainly fall into two categories based on their approach: Markov models and neural networks. Yet, these solutions are unsatisfactory for a number of reasons. Markov model related solutions work best when the prediction space is discrete (prediction of roads for driving data). However, this discrete approach doesn't apply well to less constrained domains (such as maritime/boat trajectories) where the vast search space must be discretized [12–14]. On the other hand, deep learning approaches (the most successful being Long Short-Term Memory neural networks because of their ability to handle time

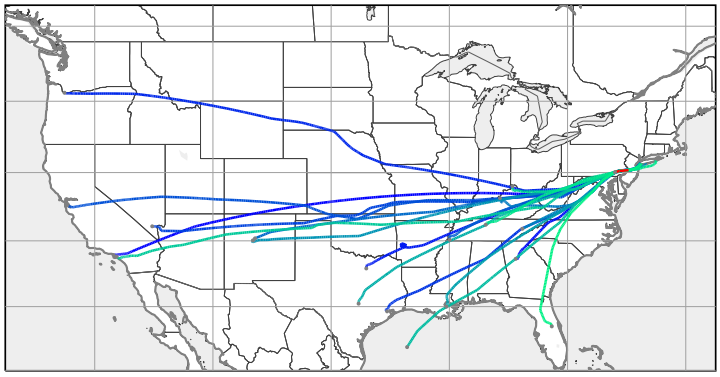


Figure 4: Predict the origin/destination of the red observed trajectory. Visualization of the 19 suggestions for this problem based on our prediction algorithm. We correctly predict the flight is going from JFK - LAX using 5 days of historical data, meaning that the origin destination pair JFK-LAX was given the highest weight and appears first on the list of possible origin destinations. The darker the trajectory the more weight the origin destination pair receives from our algorithm.

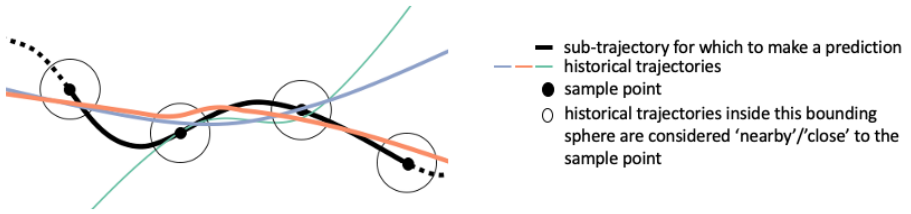


Figure 5: The orange historical trajectory is close to each of the sub-trajectory's sample points, so the orange trajectory's origin/destination appears on the list of predictions for the sub-trajectory.

series data) produce more accurate predictions than Markov modeling methods. However, they have been primarily used for next step prediction where the trajectory in question travels only one of a few routes (and each route requires its own trained model), so it generally cannot work at the scale of thousands or millions of different trajectories unless the search space is discretized [15–18] Additionally, both of these techniques lack explainability, a certain amount of which is required for wide adoption and real-world use by analysts. It is worth noting that there are also some attempts to apply similarity measures to the problem of prediction [19]. Also, it is possible to simply use a polynomial extrapolation or other mathematical techniques that don't take advantage of historical information to predict the future location of the trajectory. But these techniques don't capture the complex human decisions that are often a factor in trajectory behavior.

Our trajectory prediction algorithm is centered around answering the following question: For a given sub-trajectory, what historical trajectories have followed a similar path? To make look ups of this historical Geo-spatial data efficient, we use a spatial index for fast spatial searching. The spatial index allows for efficient queries to find what points are near (within some bounding box around) a given reference point. For this work the R-Tree spatial index is used [20]. Leveraging this structure allows us to store every point of every historical trajectory and maintain a look up time complexity of $O(\ln(n))$, where n is the number of points in the spatial index.

The prediction algorithm works as follows. We represent the observed trajectory as a list of k evenly spaced points in space. For each of those points, we do a query in the R-Tree to find the points of historical trajectories that are nearby (within a certain bounding box). A historical trajectory must be within a specified distance, d, of all k points of the observed trajectory, for the origin/destination of the historical trajectory to be considered as a possible prediction (see Figure 5). It is worth noting that we measure the distance between each point of the sub-trajectory to the trajectory itself (not a point on the trajectory) so that effects of sampling are not as strong. Our experimentation dealt exclusively with flight data, so we set d to be 5km to avoid including multiple flight corridors in one query. This number should change

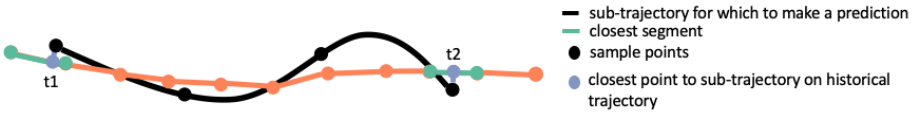


Figure 6: Represent the orange historical trajectory using segments. Find the closest segment of the historical trajectory to the first and last points of the observed trajectory. Interpolate along these two segments to find the closest point on the historical trajectory to the first and last point of the observed trajectory. Let t_1 be the time of the point on the historical trajectory closest to the first point of the observed trajectory. Let t_2 be the time of the point on the historical trajectory closest to the last point of the observed trajectory. If $t_2 - t_1 > 0$ then the historical trajectory and the sub-trajectory are going in the same direction.

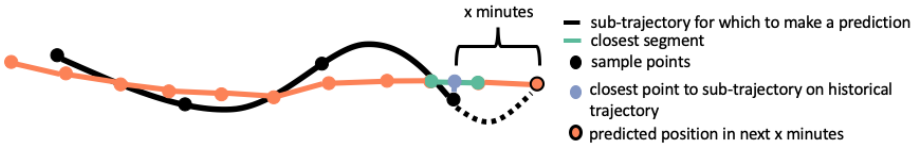


Figure 7: Represent the orange historical trajectory using segments. Find the closest segment of the historical trajectory to the last point of the observed trajectory. Interpolate along this segment to find the closest point on the historical trajectory the the last point on the observed trajectory. Traverse x minutes on the historical trajectory for a prediction of where the observed trajectory will be in x minutes.

to fit the data set. The prediction algorithm also accounts for direction. If a historical trajectory is within d of all k sample points of the observed trajectory, we check to see if the historical trajectory and the observed trajectory are going in the same direction (Figure 6). As shown in the first row of Figure 8, checking for consistent direction does not significantly impact the number of correct origin/destination pairs predicted, but it removes extraneous predictions— the correct origin/destination is predicted with fewer guesses and more confidence. If a historical trajectory passes both the distance and direction requirements, its origin/destination pair is returned as a possible prediction for the observed trajectory's origin/destination.

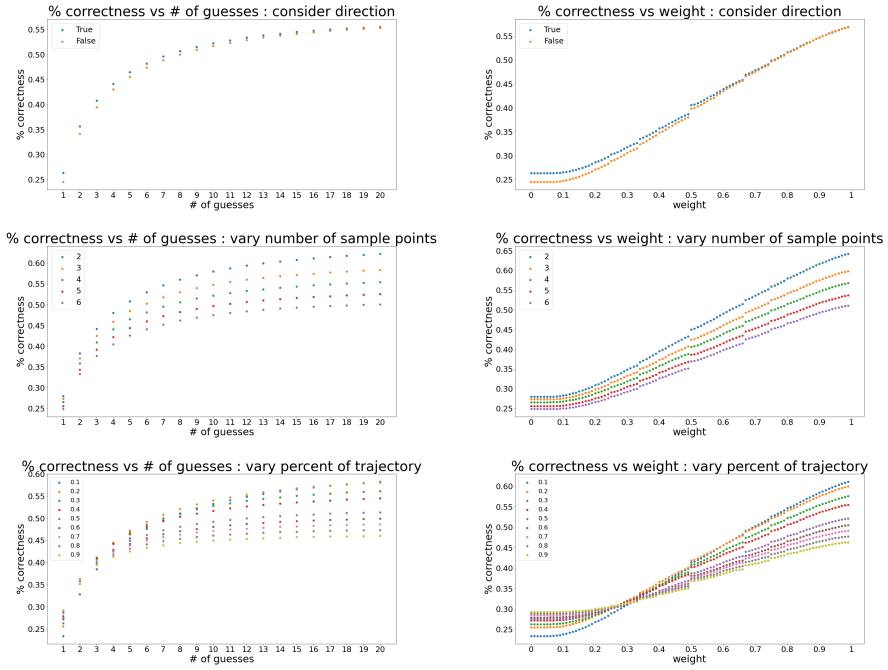


Figure 8: Experiments run on 5 days of historical data from flights in US airspace. For every trajectory in the data set, an origin/destination prediction was made for a randomly selected sub-trajectory of that trajectory.

Given a list of origin/destination predictions, it is important to consider how to order the list to best provide information on the likelihood of each origin/destination pair being the correct origin/destination for the sub-trajectory. We assign each origin/destination pair a weight, which is the number of trajectories with the given origin/destination pair normalized by the number of total trajectories that are close to all k sample points of the sub-trajectory. We experimented with taking into account the distance of each historical trajectory with a given origin/destination from the observed trajectory in the weight of the origin/destination pair. Most of the weighting schemes gave relatively similar

	1 Day	5 Days	1 Month
Number of trajectories	53,700	225,382	803,298
Percent Correct	.31	.57	.88

Table 2: Experiments run on different amounts of historical data from flights in US airspace. For every trajectory in the data set, an origin/destination prediction was made for a randomly selected sub-trajectory of that trajectory. Experiments were run by representing a 100km sub trajectory with 4 sample points.

answers, assuming the were a monotonically decreasing function of the distances between the observed trajectory sample points and the historical trajectories. Simple functions such as equal weights or linearly decreasing weights worked as well as more complex weighting schemes.

We ran experiments on the origin/destination prediction algorithm to determine how different factors influence its performance. 8 details these results. In the experiments, the weight of an origin/destination pair is the integrated weight (sum of the weights) of those before it in the prediction list. Integrated weight is used because the predicted integrated weight of an origin/destination pair is a good analog to predicted position of an origin/destination pair on the predictions list. Additionally, the correct origin/destination pair's position on the prediction list can be thought of as the number of guesses needed to get the correct origin/destination pair. The first row of 8 shows the benefits of checking that a historical trajectory and the observed trajectory are traveling in the same direction. When direction is considered compared to when it is not considered, the position of the correct origin/destination appears observably closer to the top of the prediction list and with less integrated weight. Both improvements get smaller the as position on the list and weight increase. The second and third rows of 8 show how the prediction algorithm performs depending on the quality of the observed trajectory's data. When the number of samples increases we see observable improvement on the correct origin/destination pair's location on the prediction list and its integrated weight. When the sub-trajectory in the experiment is constructed from a larger percent of the historical trajectory it is equivalent to having more data for the observed trajectory in a real life application of this algorithm. When we have more data for the observed trajectory we see observable improvement in the correct origin/destination pair's location on the prediction list and its integrated weight, but only to a point. If too much of the observed trajectory is used for the prediction it may not match with any of the historical trajectories (even ones that have the same origin/destination). Here it is necessary to weigh if it is acceptable for the correct origin/destination pair to appear on the prediction list, but have possibly many other incorrect origin/destination pairs on the list as well or if it must appear above a certain position on the list/with less than a certain integrated weight.

Providing information about weights along with the list of possible pre-

dictions is valuable for an analyst. Without the assistance of the weights an analyst may constrain their analysis to some number of places on the prediction list, maybe 5 origin/destination pairs. However, consider the possibility where only the first origin/destination pair has a significant weight. The analyst can streamline their search. Alternatively, consider the possibility where the top 6 origin/destination pairs all have similar weights. The analyst should not use their standard place cut off heuristic in this case.

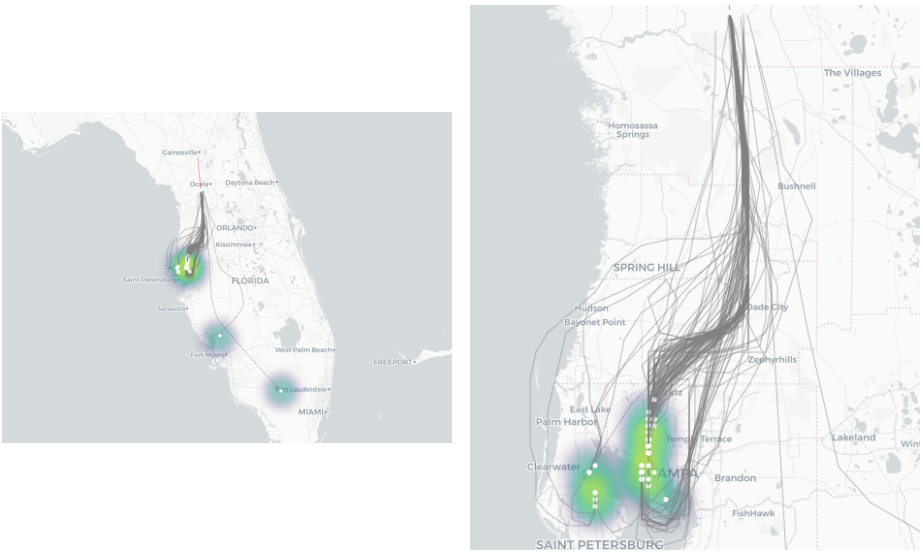


Figure 9: Predict the position of the red observed trajectory in 30 minutes. Visualization of the suggestions for this problem based on our prediction algorithm. The colors of the heat map correspond the weights of the predicted point. Green areas have more points with higher weights, purple areas have less points with lower weights. Historical trajectories that adequately match the observed trajectory appear in grey.

With minor modifications, the prediction algorithm can also be used to predict where an observed trajectory will be in specified amount of time. Consider a historical trajectory that passes the distance and direction requirements of the prediction algorithm for an observed trajectory. Find the closest point on the historical trajectory to the front (most recent) point of the historical trajectory. Move x minutes from this point on the historical trajectory for a prediction of where the observed trajectory will be in x minutes (Figure 7). Weights are then assigned by trajectory, instead of origin destination pair. 9 shows an example of position prediction.

Our algorithm is more adaptable and explainable than other trajectory prediction techniques. In fact, the significant contribution here is the matching algorithm, rather than the origin/destination prediction itself. It allows us to easily extended our prediction algorithm to predict position for the next x min-

utes into the future for given a observed trajectory. In terms of explainability, the list of origin/destination pair predictions suggested by the the algorithm are the origin/destination pairs of historical trajectories who have traveled paths near the observed trajectory in question. And, in the position prediction case, the predicted positions in x minutes are exactly the positions of historical trajectories who have traveled paths near the observed trajectory in question in x minutes.

4.3 Pattern Search

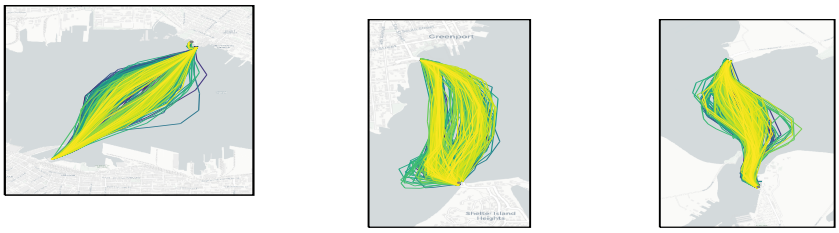
Another commonly asked question in trajectory analytics is, can we identify objects which are performing specific tasks, or tasks similar to one another? Often these tasks require particular movements and behaviors, meaning it may be possible to identify the task being performed using trajectory data. The ease with which these patterns are extracted depends on the representation of the data and methodology employed. We contend that the feature vector approach is an ideal choice to enable detection of meaningful patterns in large datasets, and give a variety of examples below.

4.3.1 Example 1: Ferries

Consider ferries: These vessels transport passengers, vehicles and/or cargo, usually over relatively short distances. Through careful feature selection, we can search (very large) datasets to find ferries and other vessels engaged in ferry-like behavior.

Ferries often travel the same route between two ports, making multiple trips in one day. As a result, their tracks create a back-and-forth pattern over a relatively small area. This means that we can expect the distance traveled to be quite a bit higher than the perimeter of the convex hull created by the trajectory; we will call the ratio between the distance traveled by a trajectory and the length of the perimeter of its convex hull the loiter ratio. Using the loiter ratio as our feature, we can calculate this for each trajectory in a given dataset and find all trajectories with a loiter ratio higher than a given threshold. These are trajectories for ferries and other vessels traveling long distances in small areas, usually back and forth between ports.

Requesting trajectories with the highest loiter ratios from the April 2018 dataset yields the results shown in Figure 10.



(a) (b) (c)

Figure 10: Three passenger ships with high loiter ratios. (a) *Burrard Otter II*, operated by the Coast Mountain Bus Company in Vancouver, Canada. (b) *Mashomack*, a ferry that operates between Greenport and Shelter Island in New York, USA. (c) *Robert H Dedman*, a ferry that operates between Galveston Island and Port Bolivar in Texas, USA.

4.3.2 Example 2: Collaborative Behavior

Using the feature vector defined by (3), we can detect cotravelers, which we define as trajectories that follow a similar path at approximately the same time. We will demonstrate this for a large dataset, namely maritime traffic in U.S. coastal waters during April 2018 obtained from the Bureau of Ocean Energy Management (BOEM) and National Oceanic and Atmospheric Administration (NOAA) via `MarineCadastre.gov/data`. This dataset contains 202,467,487 data points, each with a unique identifier, latitude, longitude and timestamp. The unique identifiers for this dataset are Maritime Mobile Service Identity (MMSI) numbers which are intended to be unique to each vessel.

We begin by converting the data points into 318,022 trajectories using Tracktable with the following parameters: (1) trajectories with fewer than 10 points are discarded, and (2) sequential trajectory points that are separated spatially by at least 20km or temporally by at least 20 minutes are split into separate trajectories. We then apply several additional filters to reduce the size of our dataset by removing uninteresting trajectories. For instance, trajectories that remain in a small area (less than 0.2 km^2) are unlikely to engage in interesting co-travel behavior, as are trajectories that travel a very short distance (less than 5km). Lastly, trajectories with *any* points within the first 20 minutes of April 1 or the last 20 minutes of April 30 were also removed to ensure we are only examining full trajectories, and not partial trajectories due to the time frame cutoff. After this filtering, 158,665 trajectories (approximately 49.9%) remain.

The last preprocessing step that is needed is a trajectory splitter. For this dataset, trajectories can have periods of mobility interspersed with periods of immobility (anchored or docked), without limitation on either. This means, for instance, that we can have a vessel that was docked for 11 days, then traveled

for several hours, then returned to be docked for another day. If we create our cotravel feature vector using (3) with $N = 10$ points equally-spaced in time, our points will all be in dock, meaning we will not be able to detect other vessels that cotraveled during the few hours that the trajectory was moving. Our solution to this is as follows: For each trajectory, if it stayed in a given radius for a given time, delete the idle points and split the trajectory into multiple parts. For this example, we have split each trajectory if it stayed within a radius of 0.2525 km for at least one hour. This resulted in 309,651 total trajectories; however, our storage footprint was reduced from 139,714,711 to 64,685,017 total points over all trajectories.

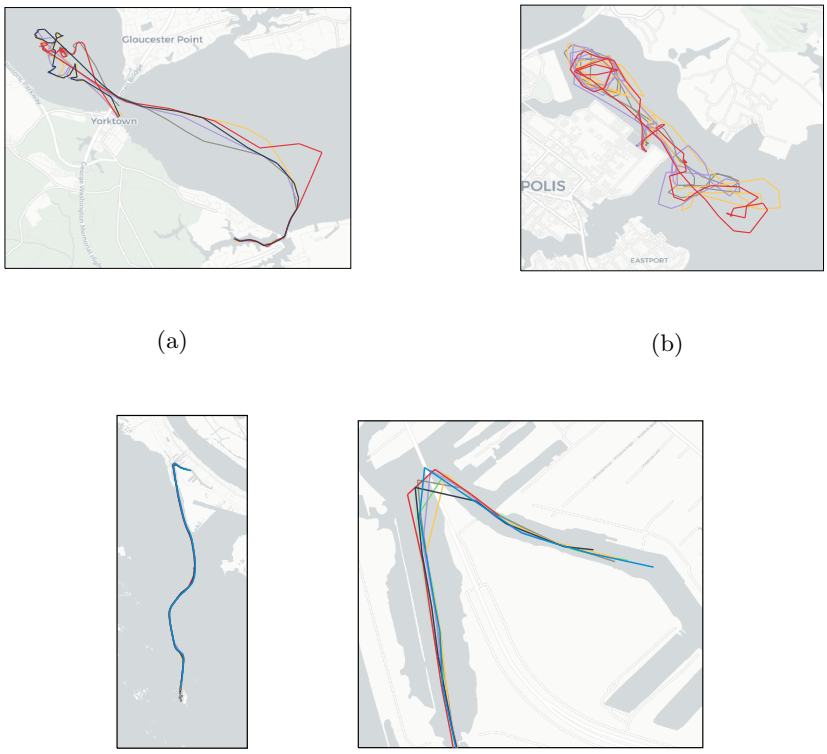
Now that we are assured that our trajectories have no excessively long idle regions, a cotravel feature vector is calculated for each trajectory using (3). Unsupervised machine learning techniques can now be applied to detect cotravelers. As discussed in Section 4.1, box-DBSCAN is an ideal choice as it has no a priori requirement for the number of clusters and can handle arbitrarily-shaped clusters.

We apply box-DBSCAN with $\varepsilon = 0.01$ degrees for each dimension of our hypercube corresponding to latitude and longitude, and $\varepsilon = 15$ minutes for the hypercube's temporal dimensions. This yields 814 cotraveling clusters, and 258,473 outliers (trajectories that are not cotraveling). Without any restrictions or expectations for how these vessels will be traveling together, we have identified the subset of cotraveling trajectories from hundreds of thousands of trajectories. A few of these cotraveling clusters are shown in Figure 11.

While our cotravel feature vector was fairly simple to compute, when coupled with box-DBSCAN it was incredibly effective at identifying cotraveling behavior in a very large dataset. Moreover, we can adapt this method as detailed in Section 3.1.2 to examine only sub-trajectories $S_\tau(\mathbf{T}, p_1, p_2)$ at fixed proportions p_1 and p_2 along trajectory $\mathbf{T}$. When used as feature vectors for box-DBSCAN, this allows us to identify trajectories that are cotraveling for only a portion of their journey, which we will call rendezvous behavior. Figure 12 shows four of the 2220 clusters engaged in rendezvous behavior that were identified using box-DBSCAN. Feature vectors we created using (4) with $p_1 = 0.4$ and $p_2 = 0.6$ (comparing only the middlemost 20% of each trajectory), and parameters for box-DBSCAN were $\varepsilon = 0.005$ degrees for each dimension of our hypercube corresponding to latitude and longitude, and $\varepsilon = 15$ minutes for the hypercube's temporal dimensions and `min_points` was set to 2.

4.3.3 Example 3: Boxes

We will again examine the dataset from April 2018 consisting of maritime traffic in U.S. coastal waters, but now we look for trajectories that move in repetitive 90° turns, forming boxes. First, we convert the 202,467,487 data points into 318,022 trajectories using Tracktable exactly as we did for our collaborative behavior examples. We will also apply the same filters (convex hull area must be at least 0.2 km^2, distance traveled must be at least 5km, and we remove potential partial trajectories in the first/last 20 minutes of the month), with the



(a) (b)

(c)

Figure 11: Three of the clusters found when using box-DBSCAN to cluster over cotravel feature vectors. The gray dot indicates the end of a trajectory. (a) Five search and rescue vessels departing from a dock in Yorktown, VA and traveling together before arriving at the National Search and Rescue School. (b) Four sail boats operated by Warrior Sailing, an outreach program that provides sailing instruction to wounded, ill, and injured service members and veterans [21], sailing together in Annapolis, MD. (c) Eight fishing vessels belonging to Daybrook Fisheries, a leader in Menhaden fishing, departing Port Sulphur, LA (right) and traveling due south together (left).

(a) (b)

(c) (d)

Figure 12: Four of the clusters found when using box-DBSCAN to cluster over rendezvous feature vectors. Color gradients indicate travel from start (dark color) to end (light color), and a gray dot is shown to indicate the end of a trajectory. (a) *Baywatch 18* (MMSI: 338145697, shown in green), a lifeguard/paramedic vessel, travels from and returns to Two Harbors, CA on Santa Catalina Island. Shown in purple, the Catalina Express owned *Catalina Jet* (MMSI: 367199310) transports passengers from Long Beach, CA to Santa Catalina Island and back. Both vessels dock at the island port of Avalon at roughly the same time before returning to their origins. (b) Tug boats *Nancy McAllister* (MMSI: 367693850, shown in green) and GM McAllister (MMSI: 367661170, shown in purple) arrive at Craney Island Creek in Norfork, VA and maneuver around for over an hour before turning back the way they each came. (c) Cargo ship *Corsair* (MMSI: 367706320, shown in green) and passenger vessels *The Hope* (MMSI: 367721960, shown in purple) and *Wyandot* (MMSI: 367721870, shown in orange) depart from separate ports bordering Lake Michigan, then dock at Mackinac Island for approximately one half hour before returning back to their individual ports. (d) Tug boats *Tommy Andrew* (MMSI: 367378180, shown in green) and *Mr. Cass* (MMSI: 366911310, shown in purple) depart from different locations along the Mississippi River, reach Algiers Lock in New Orleans within minutes of each other, then travel along the Intracoastal Waterway together before going their separate ways.

addition of a straightness filter: very "straight" trajectories ($d_{0,1}/\hat{d}_{0,1} > 0.95$, where d for this dataset is calculated as great-circle distance) will be removed, as they account for approximately 13% of our dataset and are not likely to form box shapes. This leaves us with 124,429 (approximately 39.1%) remaining trajectories. For each trajectory, we calculate its "boxiness" using equation (11) and consider the highest scorers to be anomalous. Several of these are shown in Figure 13.

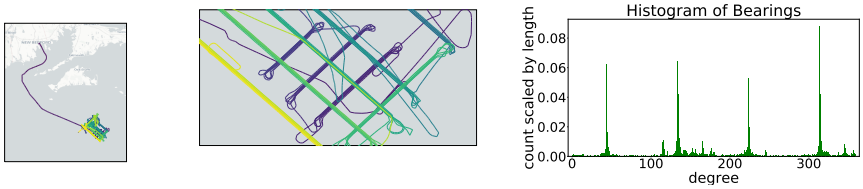
4.3.4 Example 4: Arbitrary Shape Patterns

Suppose, unlike the previous examples, that we do not have a specific behavior or trajectory shape in mind. Instead, we want to take a large dataset of trajectories and sort them into groups of similarly shaped trajectories because, as we will show, trajectories with similar shapes are often performing similar tasks. By combining feature vectors created using distance geometry (as defined in Section 3.2.2) with unsupervised machine learning, we can achieve precisely this. This is especially powerful because we do not need a priori knowledge of the shapes present in our dataset.

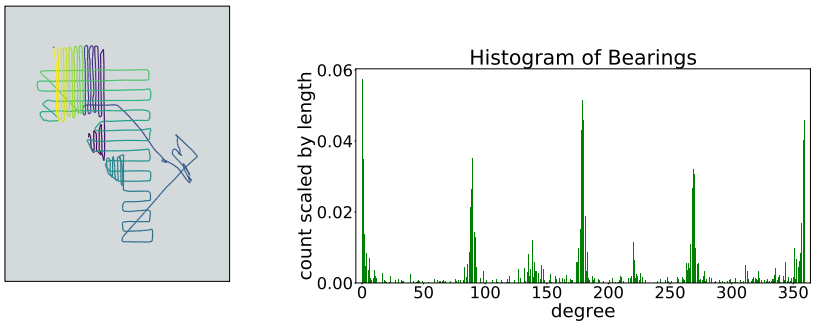
Once feature vectors have been calculated for each trajectory in our dataset using the distance geometry method described in Algorithm 1, unsupervised machine learning techniques can be applied to detect shape patterns. As we did in the collaborative behavior examples, we will use box-DBSCAN due to the advantages discussed in Section 4.1, and we will use the U.S. coastal maritime traffic dataset for April 2018.

We convert our 202,467,487 data points into 318,022 trajectories as we did in the previous examples. We will filter our very "straight" trajectories ($d_{0,1}/\hat{d}_{0,1} > 0.95$) as we did for the Boxes example above; however, our justification for this will differ: These "straight" trajectories account for approximately 13% of our dataset and are unlikely to be interesting; removing them not only increases the speed of DBSCAN's clustering but also allows it to cluster at a finer granularity without generating a large number of clusters associated with these uninterestingly shaped trajectories. After this filtering, 124,429 trajectories (approximately 39.1%) remain, and for each of these we calculate a 10-dimensional feature vector using distance geometry with depth $K = 4$. We apply box-DBSCAN with $\varepsilon = 0.05$ for each dimension of our hypercube and `min_points` $= 10$, resulting in 115 clusters and 61,928 outliers. Figure 14a shows that box-DBSCAN has identified one shape pattern that is unique to vessels traversing rivers, and in Figure 14b we see another that is shared by vessels engaged in ferry-like behavior.

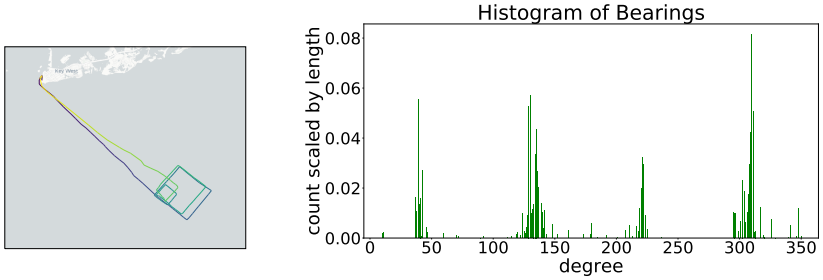
Our parameters for box-DBSCAN ($\varepsilon = 0.05$ for each dimension and `min_points` $= 10$) resulted in a tight threshold for similarity which ensured that the shape clusters were truly alike. However, this also results in a large number of outliers that did not belong to any cluster. For pattern recognition, a large number of outliers can be advantageous, as it allows us to ignore a large portion of our dataset and examine a more manageable subset of trajectories that are of



(a)

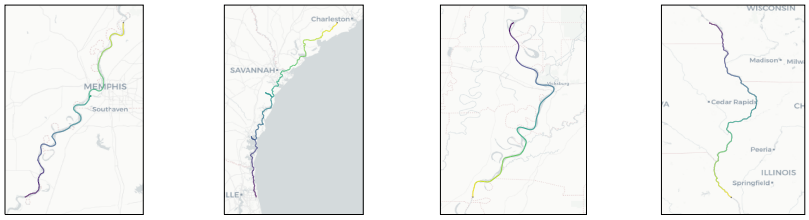


(b)

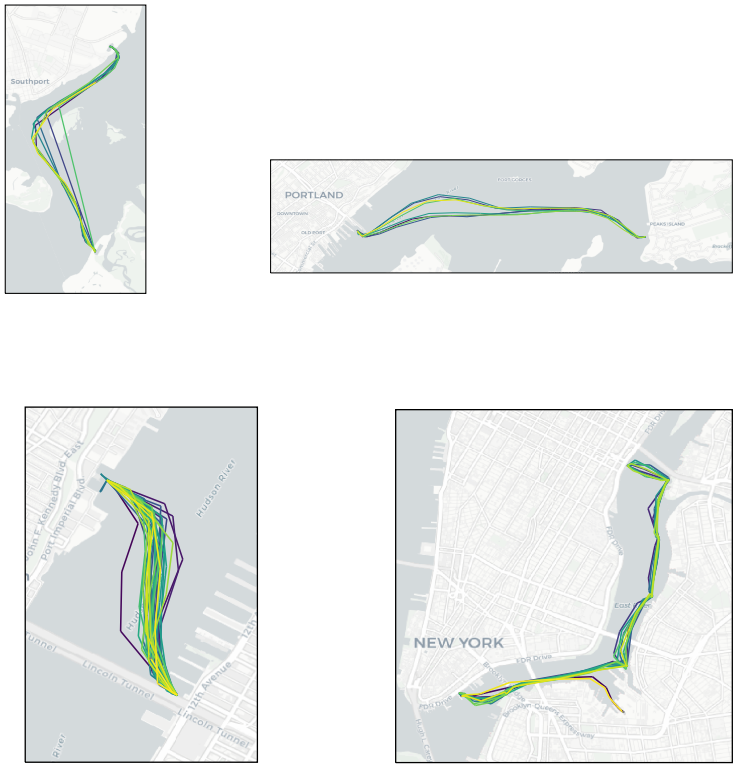


(c)

Figure 13: Three maritime trajectories making repetitive 90° turns detected using the "boxiness" feature defined by equation (11). To the right of each trajectory, the headings histogram is shown, each containing the necessary 90° offset peaks that are required for a high value of (11). (a) *Kommandor Iona* (MMSI: 235003070), a geophysical survey vessel that spent eight days at sea surveying near the coast of Rhode Island (left). It had the highest score for "boxiness" due to the meticulous 90° pattern that it created while surveying (middle). (b) *Ocean Project* (MMSI: 367474360), a survey and inspection vessel equipped with a remotely operated vehicle (ROV) and autonomous underwater vehicle (AUV) for underwater survey. On the left, *Ocean Project* can be seen surveying the Gulf of Mexico more than 100 miles off the coast of New Orleans, LA, forming a trajectory that scored 21st for "boxiness." (c) *JIATF-South* (MMSI: 369970156) is named for the Joint Interagency Task Force South located at Naval Air Station Key West, FL. This task force works to prevent illicit drug trafficking. On the left we can see what appears to be *JIATF-South* patrolling near Key West, forming the 34th "boxiest" trajectory.



(a) **River cluster.** Four of 373 river-traversing trajectories that were clustered together by box-DBSCAN.



(b) **Ferry cluster.** Four of 9,179 trajectories with ferry-like behavior (repeatedly moving back and forth between one or more points) clustered together by box-DBSCAN.

Figure 14: Using distance geometry to describe shape as detailed in Section 3.2.2, we can represent the shape of each of 124,429 maritime trajectories as a 10-dimensional feature vector. Given these feature vectors with no a priori information about the shapes present in the dataset, box-DBSCAN identified 115 prevalent shape patterns, two of which are shown above.

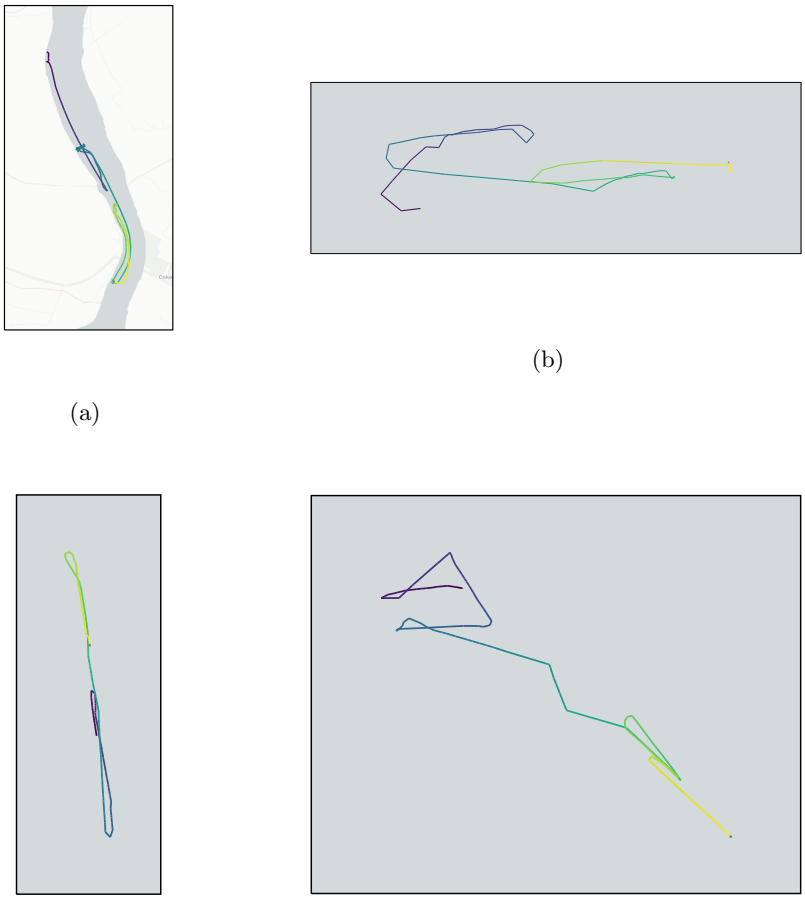
interest. However, if we alter our goal from pattern recognition to anomaly detection, we now desire a small number of outliers, and we can modify our parameters accordingly. For instance, let us cluster again on the same dataset, but with $\varepsilon = 0.213$ for each dimension and `min_points`=2. This decreases our threshold for similarity, resulting in 1 cluster and 4 outliers. Since the incentive to cluster was especially high with this box-DBSCAN parameter choice, we can conclude that these 4 outlying trajectories (shown in Figure 15) are extremely anomalously shaped.

4.4 Aggregate Analysis

Rather then generate a feature vector for each trajectory, feature vectors can be created for a set of trajectories, where the features are aggregate statistics of the trajectories in the set. Using feature vectors to represent sets of trajectories opens a new area of trajectory analysis. As an example of the power of aggregate analysis of trajectory data, a dataset of 10 months of aircraft trajectory data for the contiguous United States was analyzed. The ADS-B [1] dataset was collected from the OpenSky Network [22] for the months of January to October 2020, and included 169 Gigabytes of data containing 875 million aircraft position updates. These position updates were assembled into 13 million trajectories. Any trajectories that spanned midnight (Mountain Time) were split into two trajectories, one for each day. Finally the trajectories were binned by days, and statistics were calculated for each day.

For the first analysis, the following features were calculated for each day: the mean length of the trajectories, the mean convex hull area of the trajectories, and the mean duration of the trajectories. Notice that the features represent only mean statistics about the lengths of, durations of, and area covered by the trajectories. The features contain no direct indication of the number of trajectories each day, or the day of week or date itself.

Given just these simple aggregate statistics, however, it is possible to determine with high accuracy which days came before the Covid-19 pandemic, and which days occurred during the pandemic. We define the first day the pandemic began significantly impacting US flights as March 30, 2020, and use this date to determine pre-Covid and during Covid. Figure 16 shows the ground truth, where each dot represents a day's worth of flights. The orange dots represent days before Covid-19 (before March 30, 2020), and green dots represent days during the pandemic. TSNE has been used to compress the 3-dimensional data into 2-dimensions for better visualization, as such the axes have no direct meaning. Figure 17 shows the results of clustering the data into two clusters using Agglomerative Clustering with the ward linkage criterion. The clustering produces results that are a near perfect match to the truth, with only 9 days mis-classified (6 of which are Fridays and Saturdays pre-Covid, classified as during Covid). Interestingly, one of the two days during Covid were classified as "pre-Covid" was Easter Sunday (April 12). The ARI (Adjusted Rand Index) of the clustering compared with the ground truth is 0.881, and the AMI (Adjusted Mutual Information) is 0.779.



(a) (b) (c) (d)

Figure 15: The four most anomalously shaped trajectories found in U.S. coastal maritime traffic data from April 2018. The start of each trajectory is shown in purple, and the end is yellow and marked with a grey dot. (b)-(d) are each fishing vessels whose automatic identification system (AIS) transceivers do not appear to have been transmitting until they were at sea. (a) *John R Operle* (MMSI: 366961510), a barge traveling the Mississippi River near Columbus, KY. (b) *Max & Emma* (MMSI: 367792770), a fishing vessel operating over 100 km off the coast of Long Island, NY. (c) *Capt C* (MMSI: 367152370), a fishing vessel in the Gulf of Mexico, just off the coast of Corpus Christi, TX. (d) *Elizabeth* (MMSI: 367409750), a fishing vessel roughly 80 km due east of Toms River, NJ.

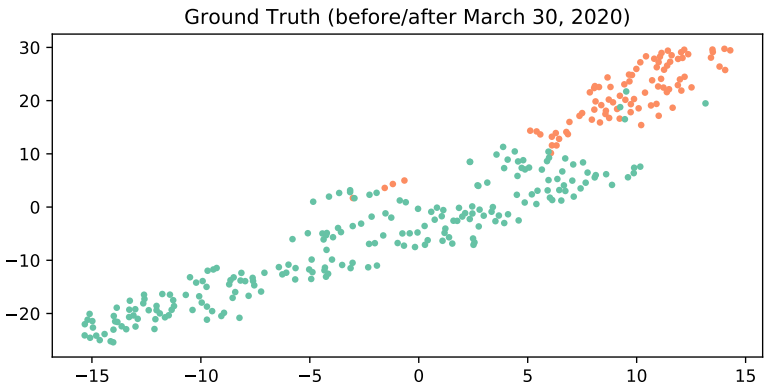


Figure 16: TSNE projection showing the ground truth, blue=pre-Covid, red= during Covid.

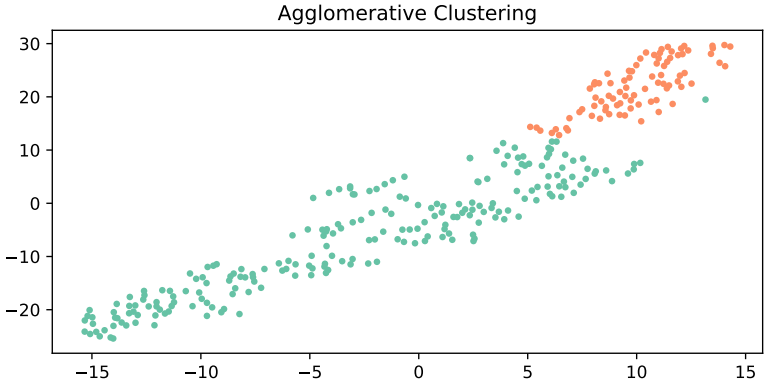


Figure 17: TSNE projection of the results of using Agglomerative Clustering to cluster the data into two clusters.

5 Future Work

The work described in this document represents the early explorations of these techniques. There are a number of important follow-on studies that need to be done. One of the most important would be a systematic study of how to automatically choose which features are most relevant to the analysis at hand. To a large extent, this is a dimensionality-reduction exercise that occurs frequently in machine learning problems. However, the choices of features used in many applications are not just strictly chosen for their mathematical relevance, but also their explainability and how they can connect to the thinking of analysts. Once the relevant features are chosen, understanding the structure of the mathematical feature space in which the objects exist then becomes a key issue for study. Ideally, one would want the clusters of the different type of objects to have a significant separation in this space in order to build robust classifiers. Building these classifiers and understanding where the classification boundaries are robust and where they are fragile is work that needs to be done. A different direction of further research involves the study of how to represent uncertainty in the feature vector representation. Using the approach of the R-Tree, one could represent feature values not just as points in space, but objects with spatial extent. This spatial extent could represent the uncertainty in a value, and be used to build a more robust model of confidence within many different types of results.

Another important area of future study is that of doing a better job in unsupervised learning. Clustering approaches are very natural for this feature space approach, but most clustering approaches have a number of parameters that must be chosen carefully to get meaningful answers. This includes choices of how large a cluster needs to be and how close a points needs to be to another point in *each dimension of the feature space* in order to be considered a neighbor. Automation of this process is needed for real-world applications, especially situations where there is a high dimensionality to the feature space. Finally, understanding the ideal hardware architecture for both storing and working with the data real-time is something that is critical to its real-world usage.

6 Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] United States Code of Federal Regulations, Title 14 - Aeronautics and Space, Chapter I - Federal Aviation Administration, Department of Transportation, Subchapter F - Air Traffic and General Operating Rules, Part 91 - General Operating and Flight Rules, Subpart C - Equipment, Instrument and Certificate Requirements, § 91.225 Automatic Dependent Surveillance-Broadcast (ADS-B) Out equipment and use. `https://ecfr.federalregister.gov/current/title-14/chapter-I/subchapter-F/part-91/subpart-C/section-91.225`.

[2] United States Code of Federal Regulations, Title 14 - Aeronautics and Space, Chapter I - Federal Aviation Administration, Department of Transportation, Subchapter F - Air Traffic and General Operating Rules, Part 91 - General Operating and Flight Rules, Subpart C - Equipment, Instrument and Certificate Requirements, § 91.227 - Automatic Dependent Surveillance-Broadcast (ADS-B) Out equipment performance requirements. `https://ecfr.federalregister.gov/current/title-14/chapter-I/subchapter-F/part-91/subpart-C/section-91.227`.

[3] Pablo Samuel Castro, Daqing Zhang, Chao Chen, Shijian Li, and Gang Pan. From taxi gps traces to social and community dynamics: A survey. *ACM Computing Surveys*, 46(2), December 2013.

[4] Su, Liu, Zheng, Zhou, and Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32, 2020.

[5] P. C. Besse, B. Guillouet, J. Loubes, and F. Royer. Review and perspective for distance-based clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 17(11):3306–3317, 2016.

[6] Eiter and Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

[7] Berndt and Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[8] Vlachos, Kollios, and Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th international conference on data engineering*, pages 673–684. IEEE, 2002.

[9] G.M. Crippen and T.F. Havel. *Distance Geometry and Molecular Conformation*. Chemometrics research studies series. Research Studies Press, 1988.

[10] Ester, Kriegel, Sander, Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Campello, Moulavi, and Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[12] Neto, Souza Baptista, and Campelo. Combining markov model and prediction by partial matching compression technique for route and destination prediction. *Knowledge-Based Systems*, 154:81–92, 2018.

[13] Xing Wang, Xinhua Jiang, Lifei Chen, and Yi Wu. Kvlmm: A trajectory prediction method based on a variable-order markov model with kernel smoothing. *IEEE Access*, 6:25200–25208, 2018.

[14] Yassine Lassoued, Julien Monteil, Yingqi Gu, Giovanni Russo, Robert Shorten, and Martin Mevissen. A hidden markov model for route and destination prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[15] Zhang and Mahadevan. Bayesian neural networks for flight trajectory prediction and safety assessment. *Decision Support Systems*, 131:113–246, 2020.

[16] Ebel, Göl, Lingenfelder, and Vogelsang. Destination prediction based on partial trajectory data. *arXiv preprint arXiv:2004.07473*, 2020.

[17] Endo, Nishida, Toda, and Sawada. Predicting destinations from partial trajectories using recurrent neural network. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2017.

[18] Shi, Xu, Pan, Yan, and Zhang. Lstm-based flight trajectory prediction. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[19] Froehlich and Krumm. Route prediction from trip observations. Technical report, SAE Technical Paper, 2008.

[20] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[21] Dan Dickinson. Warrior sailing, coming to a harbor near you. *Southwinds Magazine*, pages 45–47, December 2019.

[22] Matthias Schäfer, Martin Strohmeier, Vincent Lenders, Ivan Martinovic, and Matthias Wilhelm. Bringing up opensky: A large-scale ads-b sensor network for research. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, page 83–94. IEEE Press, 2014.

[23] Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *Journal of Mathematical Imaging and Vision*, 27:203–216, 2007.

[24] Di Wang, Tomio Miwa, and Takayuki Morikawa. Big trajectory data mining: A survey of methods, applications, and services. *Sensors*, 20(16):4571, Aug 2020.

[25] Almeida, Souza Baptista, Andrade, and Soares. A survey on big data for trajectory analytics. *ISPRS International Journal of Geo-Information*, 9(2), 2020.

[26] Georgiou, Pelekis, Sideridis, Scarlatti, and Theodoridis. Semantic-aware aircraft trajectory prediction using flight plans. *International Journal of Data Science and Analytics*, 9(2):215–228, 2020.

[27] Guo, Shekhar, Xiong, Chen, and Jing. Utsm: A trajectory similarity measure considering uncertainty based on an amended ellipse model. *ISPRS International Journal of Geo-Information*, 8(11):518, 2019.

[28] Bian, Tian, Tang, and Tao. Trajectory data classification: A review. *ACM Trans. Intell. Syst. Technol.*, 10(4), aug 2019.

[29] Wu, Luo, Shao, Tian, and Peng. Location prediction on trajectory data: A review. *Big data mining and analytics*, 1(2):108–127, 2018.

[30] Naveh and Kim. Urban trajectory analytics: day-of-week movement pattern mining using tensor factorization. *IEEE Transactions on Intelligent Transportation Systems*, 20(7):2540–2549, 2018.

[31] Magdy, Abdelkader, and El-Bahnasy. A comparative study of similarity evaluation methods among trajectories of moving objects. *Egyptian Informatics Journal*, 19(3):165–177, 2018.

[32] Georgiou, Karagiorgou, Kontoulis, Pelekis, Petrou, Scarlatti, and Theodoridis. Moving objects analytics: Survey on future location & trajectory prediction methods. *arXiv preprint arXiv:1807.04639*, 2018.

[33] Ding, Chen, Gao, Jensen, and Bao. Ultraman: a unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment*, 11(7):787–799, 2018.

[34] Teimouri, Indahl, Sickel, and Tveite. Deriving animal movement behaviors using movement parameters extracted from location data. *ISPRS International Journal of Geo-Information*, 7(2):78, 2018.

[35] Shang, Li, and Bao. Dita: Distributed in-memory trajectory analytics. In *Proceedings of the 2018 International Conference on Management of Data*, pages 725–740, 2018.

[36] Li, Zhao, Cong, Jensen, and Wei. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 617–628. IEEE, 2018.

[37] Tasnim, Caldas, Pissinou, Iyengar, and Ding. Semantic-aware clustering-based approach of trajectory data stream mining. In *2018 International Conference on Computing, Networking and Communications (ICNC)*, pages 88–92. IEEE, 2018.

[38] Tiwari, Arya, and Chaturvedi. Scalable prediction by partial match (ppm) and its application to route prediction. In *Applied Informatics*, volume 5, page 4. Springer, 2018.

[39] Arboleda, Fernández, and Bogorny. Towards a semantic trajectory similarity measuring. *Indian Journal of Science and Technology*, 10:18, 2017.

[40] Di Ciccio, Aa, Cabanillas, Mendling, and Prescher. Detecting flight trajectory anomalies and predicting diversions in freight transportation. *Decision Support Systems*, 88:1–17, 2016.

[41] Mueller, Perelman, and Veinott. An optimization approach for mapping and measuring the divergence and correspondence between paths. *Behavior Research Methods*, 48(1):53–71, 2016.

[42] Xue, Qi, Xie, Zhang, Huang, and Li. Solving the data sparsity problem in destination prediction. *The VLDB Journal*, 24(2):219–243, 2015.

[43] Ranu, Deepak, Telang, Deshpande, and Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *2015 IEEE 31st International Conference on Data Engineering*, pages 999–1010. IEEE, 2015.

[44] Xue, Zhang, Zheng, Xie, Huang, and Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 254–265. IEEE, 2013.

[45] Yanagisawa, Akahani, and Satoh. Shape-based similarity query for trajectory of mobile objects. In *International Conference on Mobile Data Management*, pages 63–77. Springer, 2003.

[46] Bureau of Ocean Energy Management (BOEM), National Oceanic, and Atmospheric Administration (NOAA). Maritime traffic in us̀coastal waters during april 2018.

[47] Ertöz, Steinbach, and Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 47–58. SIAM, 2003.

[48] Pelekis, Kopanakis, Marketos, Ntoutsi, Andrienko, and Theodoridis. Similarity search in trajectory databases. In *14th International Symposium on Temporal Representation and Reasoning (TIME'07)*, pages 129–140. IEEE, 2007.

[49] Kim and Mahmassani. Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories. *Transportation Research Procedia*, 9:164–184, 2015.

[50] Wilson, Rintoul, and Valicka. Exploratory trajectory clustering with distance geometry. In *International Conference on Augmented Cognition*, pages 263–274. Springer, 2016.