A Benchmark of Existing Tools for Outlier Detection and Cleaning in Trajectories

Mariana Machado Garcez Duarte (mariana.machado.garcez.duarte@ulb.be)  
Université Libre de Bruxelles

Mahmoud Sakr  
Université Libre de Bruxelles

Research Article

**Keywords:** Outlier Detection, Trajectory cleaning, Trajectory Preprocessing, Benchmark, Programming Libraries, Outlier detection algorithms

**Posted Date:** September 18th, 2023

**DOI:** https://doi.org/10.21203/rs.3.rs-3356633/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

**Additional Declarations:** No competing interests reported.
A Benchmark of Existing Tools for Outlier Detection and Cleaning in Trajectories

Mariana M Garcez Duarte¹* and Mahmoud Sakr¹,²*

¹ Université Libre de Bruxelles, Brussels, Belgium.
² Ain Shams University, Cairo, Egypt.

*Corresponding author(s). E-mail(s):
mariana.machado.garcez.duarte@ulb.be; mahmoud.sakr@ulb.be;

Abstract
Outlier detection and cleaning is an essential step in data preprocessing to ensure the integrity and validity of data analyses. This paper focuses on outlier points within an individual trajectory, i.e., points that deviate significantly inside a single trajectory. We benchmark ten open-source libraries to comprehensively evaluate available tools, comparing their efficiency and accuracy in identifying and cleaning outliers. This benchmarking considers the libraries as they are offered to end users, with real-world applicability. We compare existing outlier detection libraries, introduce a method for establishing ground-truth, and aim to guide users in choosing the most appropriate tool for their specific outlier detection needs. Furthermore, we survey the state-of-the-art algorithms for outlier detection and classify them into seven types: Statistic-based methods, Sliding window algorithms, Clustering-based methods, Graph-based methods, Ensemble-based methods, Learning-based methods, and Heuristic-based methods. Our research provides insights into these libraries’ performance and contributes to developing data preprocessing and outlier detection methodologies.

Keywords: Outlier Detection, Trajectory cleaning, Trajectory Preprocessing, Benchmark, Programming Libraries, Outlier detection algorithms.

1 Introduction
Trajectory data, which refers to the path that a moving object follows through space as a function of time, finds extensive applications in various fields such as transportation, logistics, and environmental studies. However, this data type is often plagued by
outliers or noise due to sensor errors and connectivity issues. Outliers are data points that deviate significantly from the expected pattern. In the context of trajectory data, these could represent an abrupt and uncharacteristic change in direction or speed. Such outliers can lead to misleading analysis results and inaccurate decision-making, impacting businesses, governments, and individuals alike. For instance, in transportation, outliers in vehicle trajectory data could lead to incorrect traffic flow analysis, resulting in inefficient traffic management. This makes outlier or noise detection a crucial step in trajectory cleaning Magdy et al. (2017). As such, outlier detection is an essential function in mobility data management systems Zimányi et al. (2020a); Zimányi et al. (2019).

There are two outlier detection categories, one focusing on a collection of trajectories, where a trajectory can be an outlier, and another on points inside one trajectory. In this paper, we focus on the latter.

We benchmark open-source libraries and compare their efficiency and accuracy in detecting and cleaning outliers. The main contributions of this work are:

- An automated method for generating ground-truth in multi-sensor data.
- A survey of the state of the art in outlier detection algorithms.
- A benchmark of libraries, including MovingPandas Graser (2019), Scikit-mobility Pappalardo et al. (2019), Scikit-learn Pedregosa et al. (2011), Ptrail Haidri et al. (2021), PyMove Sanches (2019), movetk Custers et al. (2021), MEOS Zimányi et al. (2020b), Argosfilter Freitas et al. (2008), Stmove Seidel et al. (2019) and MO outlier Filzmoser and Gschwandtner (2017) focusing on outlier detection. The benchmark considers the user’s perspective. That is, the context is to compare the offering of the existing libraries for end users rather than comparing their algorithms and implementation aspects.

Outline. The rest of the paper is structured as follows. In Section 2 we present essential concepts for outlier detection. Section 3 surveys the state of art. A benchmark using real data is the subject of Section 4. In Section 5, we discuss our findings and we conclude.

2 Outlier Detection

There are a variety of outlier detection techniques. In Wang et al. (2019), the authors divide the methods into six categories: Statistical-based methods, Sliding window algorithms, Clustering-based methods, Graph-based methods, Ensemble-based methods, and Learning-based methods.

Statistical-based methods

The essential concept of these techniques for identifying outliers relies on their alignment with the statistical distribution model. These methods include the Kalman filter (KF). KF is a well-established method used to smooth point series. This algorithm estimates missing points based on previously observed values that might have measurement errors. In Urrea and Agramonte (2021), the authors mention the advantage of the KF and its derivatives is its recursive aspect, which can be used in real time. It
is also widely used due to its simplicity and capability to provide accurate estimations and prediction results.

Particle filters (PF) use a set of randomly generated particles to represent the possible states of the system and update the particles based on observed data. In contrast to KF, they are not restricted to the Gaussian distribution of errors, which makes them applicable to a broader range of noisy data. PF can, however, be computationally intensive and thus not commonly implemented in trajectory libraries. Additionally, like KF, PF is sensitive to the first measurement in the trajectory, and their accuracy can be reduced if the first point is an anomaly Zheng (2015); Lee and West (2010); Kotecha and Djuric (2003).

The Hampel Filter (HF) detects and replaces outliers in trajectories with estimates via the Hampel identifier. The HF expresses a conventional heuristic that almost all values lie within three standard deviations of the mean Pearson et al (2016). For each trajectory, the method calculates the median of a sliding window and adjacent points on each side of the trajectory. The HF estimates the standard deviation of each point about its window median using the median absolute deviation. If a measurement differs from the median by more than the threshold, the filter replaces the sample with the median.

**Sliding window algorithms**

The foundation of sliding window algorithms is to calculate the distance between different data points in their neighborhood. An outlier is a data point that is significantly distant from its neighbors. When detecting an outlier inside a trajectory, the method compares each point in the trajectory to its neighbors and selects the points significantly further away than expected. Methods based on the mean or Median Filters replace points compared to the measurements done at preceding points in time. These algorithms are simple and practical for detecting single outliers. Nonetheless, these techniques depend on the number of predecessors compared to the mean. Multiple successive outlier points can affect the accuracy of the outcome trajectory. These algorithms are only sometimes effective at detecting multiple successive outlier points, which can affect the accuracy of the output trajectory. More advanced algorithms or methods may be needed to detect and correct outliers in these cases.

In Knorr et al (2000), the authors review a series of sliding window algorithms. They develop an optimized cell-based algorithm that outperforms the existing methods. The cell-based methodology is built on partitioning the multidimensional data space into a structured grid of cells. The dimensions of these cells are contingent on the distance threshold established for detecting outliers. Subsequently, each cell within the grid is searched to identify potential outliers. In experiments, the method exhibits scalability with large datasets. However, the computational expense escalates significantly as the dimensionality increases. The optimization of the cell-based methodology entails a strategic approach to minimize the number of cells that necessitate examination. This is accomplished by maintaining a register of active cells containing at least one object. Only these active cells are subject to examination in the ensuing passes over the dataset.
**Clustering-based methods**

Clustering-based techniques leverage standard clustering algorithms to distinguish outliers in the data. These techniques leverage the data's inherent structure and density characteristics to differentiate outliers from regular data points. In these methods, data points that do not belong to or lie close to large or dense clusters are considered outliers.

One such algorithm that employs this approach is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) Ester et al (1996). DBSCAN is a density-based clustering algorithm that identifies clusters as high-density regions separated by regions of lower density. It is particularly effective in detecting outliers as it does not require the specification of the number of clusters, and it can discover clusters of arbitrary shape, unlike many other clustering algorithms. Outliers in DBSCAN are identified as points that do not belong to any cluster.

Another algorithm that utilizes clustering-based techniques for outlier detection is CLARANS (Clustering Large Applications based upon RANdomized Search) Ng and Han (1994). CLARANS is an algorithm for medoid (representative objects) based clustering, a partitioning-based clustering variant. The algorithm identifies outliers as points far from the medoids of any clusters. CLARANS has the advantage of being more robust to noise and outliers than other partitioning-based clustering algorithms.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) Zhang et al (1996) is another algorithm that relies on clustering-based techniques. BIRCH is a hierarchical clustering algorithm that can incrementally and dynamically cluster incoming multi-dimensional data points to produce the best quality clustering with the memory available and time constraints. In the context of outlier detection, BIRCH identifies outliers while building the CF (Clustering Feature) tree, a data structure used to summarize the information of data points in the dataset. Data points that do not fit well into the structure of the CF tree are considered outliers.

K-Means Jain (2010) is a partitioning method that divides n observations into k clusters, each belonging to the nearest mean cluster. The algorithm operates iteratively and randomly selects ‘k’ data points from the dataset. These points act as the initial partitions of the clusters. A Euclidean distance to each centroid is computed for the remaining data points, and the point is assigned to its nearest centroid. Once all points have been assigned to clusters, the partitions of the clusters are recalculated. This process continues until the partitions no longer move significantly or a set number of iterations is reached. K-mean is an efficient and straightforward method, making it particularly suitable for large datasets. However, the algorithm has a few limitations with complex geometrically distributed data. It is also sensitive to the initial choice of partitions and can fall into local minima. Furthermore, the number of clusters ‘k’ needs to be specified beforehand, which can be a drawback if the data does not suggest a precise number of clusters.

**Graph-based methods**

Graph-based techniques are a set of methodologies that leverage the structure of graphs to capture the relationships between interconnected data points. These techniques are effective in identifying outliers. Points are constituted as nodes, and their
relationships are represented as edges. The weight of an edge typically represents the similarity or distance between two data points. The graph thus formed encapsulates the data structure, and the properties of this graph are used to identify outliers. Outliers are often identified as nodes that have few connections or are connected to other nodes with weak relationships. In other words, outliers are typically data points that do not fit well into the graph structure.

One common graph-based technique for outlier detection is the Spectral Clustering algorithm Ng et al (2002). This algorithm uses the eigenvalues of the Laplacian matrix of the graph, a matrix representation of the graph, to identify clusters of nodes. Outliers are often found as nodes that do not belong to these clusters.

Another graph-based technique is the Local Outlier Factor (LOF) algorithm Breunig et al (2000). This algorithm calculates a score for each data point based on the density of its local neighborhood compared to its neighbors’ neighborhoods. Data points with a high LOF score are considered outliers.

**Ensemble-based methods**

Ensemble-based approaches focus on merging the outputs from different models to generate more robust outlier detection models. These methods assist in determining whether an outlier identification model should be based on linear models, distance-based models, or other types of models.

In the paper, Salehi et al (2014) review ensemble techniques explicitly for evolving data streams. Instead of continuously modeling and updating the data streams over time, the authors suggest using an ensemble to generate clustering models. The novelty of their approach lies in calculating the outlierness value of an incoming data point, which is determined by utilizing only the applicable set of clustering models. This method provides an adaptable solution for anomaly detection in data streams subject to change over time.

In the paper Lazarevic and Kumar (2005), the authors propose a technique to address the challenges associated with high-dimensional data, where traditional outlier detection methods often struggle because each technique utilizes a small and different set of features. For this, the authors use feature Bagging, creating multiple subsets of elements from the original dataset and applying a chosen outlier detection algorithm to each subgroup. The final outlier score for each data point is then determined by aggregating the scores from all subsets. This approach allows the method to capture a broader range of outlier characteristics, as different subsets of features may reveal various aspects of the data.

**Learning-based methods**

Learning-based approaches, including active and deep learning, aim to train different models to detect outliers. The fundamental idea behind these methods is to leverage the learning techniques to see anomalous observations in the dataset.

In Chalapathy and Chawla (2019), the authors extensively review deep learning methods. The paper discusses the motivation and challenges for deep anomaly detection (DAD) techniques. The authors highlight the limitations of traditional algorithms in detecting outliers, especially in image and sequence datasets, due to their failure
to capture complex structures in the data. They also emphasize the need for large-scale anomaly detection as the volume of data increases. DAD techniques, which learn hierarchical discriminate features from data, are proposed to solve these challenges.

In Fernández-Maimó et al (2018), the authors define a self-adaptive deep learning-based architecture for anomaly detection in 5G networks. The architecture analyzes the network traffic by extracting features from network flows, which are sequences of network packets traveling between source and destination in a network. These features can include various aspects of the network traffic, such as packet size, packet frequency, and other protocol-specific characteristics. By analyzing these features, the deep learning-based system can learn to identify correlations and patterns that are indicative of normal network behavior. Consequently, when the system encounters network traffic that deviates significantly from these known patterns, it is flagged and further investigated.

Kakanakova and Stoyanov (2017) apply a deep learning architecture to solve outlier detection in sensor data when the border’s shape is highly non-linear. The authors also add the requirement of not being computationally intensive. They compare with standard outlier detection methods such as support vector machines and naive Bayes. The authors conclude that there is a trade-off between computational cost and accuracy. Standard detection methods offer estimated resolution with a lower computation cost, while deep learning methods gain in accuracy with a higher cost. In addition, when deep learning is applied to simpler problems, the accuracy obtained is lower than in cases with higher complexity.

Graser et al (2023) reviews deep learning-based methods used to learn from mobility data. The authors differentiate between deep learning models trained using dense, sparse, and aggregated trajectories. It is classified into eight use case categories of deep learning from trajectory data: Location classification, Arrival time prediction, Traffic volume prediction, Trajectory prediction/imputation, (Sub)trajectory classification, Next location / final destination prediction, Anomaly detection, Synthetic data generation. For Anomaly detection, the authors focus on a collection of trajectories, where a trajectory can be an outlier Liatsikou et al (2021); Nguyen et al (2022); Delsolneux et al (2008); Singh et al (2022).

Liatsikou et al (2021) utilizes synthetically generated anomalies and developed a Long Short-Term Memory (LSTM) network. In addition, all trajectories are cropped to nine points since the autoencoder needs a fixed-size input. Nguyen et al (2022) implement a model for maritime trajectory anomaly detection, namely GeoTrackNet. The model is a probabilistic Recurrent Neural Network (RNN) representation of AIS trajectories with a contrarian approach Delsolneux et al (2008). Singh et al (2022) implement an RNN-based anomaly detection system to detect abnormal trajectories.

**Heuristic-based techniques**

Heuristic-based techniques focus more on detection than correction. For instance, Zheng (2015) does not replace outlier points with estimated values but instead removes them from the trajectory. Common heuristics are based on speed with the idea that if the speed change rate is significantly higher than a given threshold and a proportion of the points in the entire trajectory, the point is removed. This approach has the
advantage of not introducing any estimated values into the trajectory but can lead to significant data loss.

In Custers et al (2021), a new method category based on physical movement properties is introduced, such as speed (Optimal Speed-bounded) and acceleration (Optimal Acceleration-bounded). This method defines limits on the minimum and maximum allowed values for these properties and uses them to determine whether a point in the trajectory is consistent with the model. The limits are minimum speed $v_-$ and maximum speed $v_+$, minimum acceleration $a_-$, and maximum acceleration $a_+$. It follows the definition that from one point to its successor, there should always be inside $[v_-, v_+]$ and $[a_-, a+]$. In addition, a trajectory $T = \langle p_1, ..., p_n \rangle$ is consistent with the model if and only if there exists at least one point in a path such that the measurement coincides with the point and the speed and acceleration are inside speed and acceleration bounds. The method defines a reachable region as a cone, i.e., given the physical boundaries, reaching the cone from point $p_i$ to $p_{i+1}$ is possible. In opposition, it is not necessarily possible to construct a trajectory from the concatenation of two consistent sub-trajectories: the concatenation $\langle p_1, ..., p_n = q_1, ..., q_m \rangle$ of two consistent subsequences $T = \langle p_1, ..., p_n \rangle$ and $U = \langle q_1, ..., q_m \rangle$ with $p_n = q_1$ is not necessarily consistent. Joining these sub-trajectories can reproduce inconsistent points—especially when considering an acceleration-bound model. The speed of two points can infer two accelerations for the same position. The model is called concatenable if it is possible to join both sub-trajectories respecting the bounds.

In the next Session, we relate some of these methods to state-of-the-art libraries.

3 State of Technology

This Section will review the available libraries that offer trajectory outlier detection and correction. We also relate some of the algorithms in Section 2 to state-of-the-art libraries.

MovingPandas Graser (2019)\(^1\) is a Python library for trajectories of moving objects. Data can be represented in Pandas Wes McKinney (2010), GeoPandas Yuan et al (2010), HoloViz Yang et al (2022), CSV, GIS file formats, JSON, and geoJSON. MovingPandas implements structures for movement data in Python for interaction and analysis of movement. This library has many trajectory manipulation functions. Focusing on the outlier detection, this library implements KF. For outlier detection, MovingPandas uses the Kalman Filter (KF) algorithm, which is implemented using the Stone Soup software Thomas et al (2017).

Scikit-mobility Pappalardo et al (2019)\(^2\) is a Python library that extends Pandas Wes McKinney (2010). Scikit-mobility offers functions for preprocessing and cleaning trajectory and analysis. The library chosen method for outlier detection is heuristic filtering, based on the speed and a given threshold. This approach can be effective for identifying points in the trajectory that deviate significantly from the expected pattern but may not be as accurate as other methods that use estimation to deal with outlier points.

\(^1\)https://github.com/anitagraser/movingpandas-examples
\(^2\)https://github.com/scikit-mobility/scikit-mobility
Scikit-learn Pedregosa et al (2011)\(^3\) is a Python library focusing on the efficiency and ease of use of machine-learning algorithms. It provides simple and consistent interfaces, making it accessible to practitioners in machine learning. The library supports supervised and unsupervised learning algorithms standard linear regression models to neural networks, and it also includes tools for model fitting, data preprocessing, model selection, and evaluation. It applies the Local Outlier Factor (LOF) algorithm for outlier detection. The process begins by choosing features of interest that are then isolated for further processing. It then utilizes an unsupervised outlier detection method that computes the local density deviation of a data point with its neighbors. Outliers, regarded as samples having a considerably lower density than their neighbors, are determined by fitting the LOF model to the standardized data and identifying instances where the model’s prediction is inaccurate.

Ptrail Haidri et al (2021)\(^4\) is a Python package that uses parallel computation and vectorization, making it suitable for large datasets. It offers several preprocessing steps, such as feature extraction, filtering, interpolation, and outlier detection. Ptrail removes outliers using a Hampel filter (HF) Trofficus (2021) based on the distance and speed of the ships between consecutive points. For each trajectory, the method calculates the median of a sliding window and adjacent points on each side of the trajectory. The HF also estimates the standard deviation of each point about its window median using the median absolute deviation. If a measurement differs from the median by more than the threshold multiplied by the standard deviation, the filter replaces the sample with the median.

PyMove Sanches (2019); Oliveira (2019)\(^5\) is a Python library. It offers a range of operations for data preprocessing and pattern mining. PyMove also provides tools for data visualization, allowing users to explore and understand their data through various techniques and channels. PyMove detects outlier points considering the distance traveled, minimum, and maximum speed.

Movetk Custers et al (2021)\(^6\) is C++ library. It offers tools for constructing, cleaning, and analyzing trajectory data. One of the key features of Movetk is its implementation of the Optimal Speed-bounded and Optimal Acceleration-bounded algorithms for outlier detection. Movetk implements a series of outlier detection methods based on the Optimal Speed-bounded algorithm and the Optimal Acceleration-bounded algorithm. In addition to these algorithms, Movetk implements a range of other methods for outlier detection, including greedy and smart greedy approaches. These methods build on the basic speed and acceleration-bounded algorithms and incorporate additional strategies and techniques to improve their performance and accuracy. For the greedy approach, Movetk greedily builds a consistent subsequence by testing if the new measurement is consistent with the last in the subsequence under the speed-bounded model (GSB) or acceleration-bounded model (GAB). In addition, the authors implement a Smart Greedy Speed/Acceleration-bounded method (SGSB/SGAB). With SGSB and SGAB, multiple subsequences are tracked simultaneously. The next measurement is added to all subsequences that end in a consistent measurement; if there

\(^3\)https://scikit-learn.org/stable/
\(^4\)https://github.com/YakshHaranwala/PTRAIL
\(^5\)https://github.com/InsightLab/PyMove
\(^6\)https://github.com/movetk/movetk
is no such subsequence, a new subsequence starting with the measurement is created. In the end, the longest subsequence is returned. Also, as a baseline, the library implements their interpretation of the method described in Zheng (2015) as Local Greedy Speed-bounded (LGSB). In LGSB, a graph is constructed with a vertex per measurement. Two vertices are connected if their timestamps are successive in the original trajectory and they are consistent to the speed-bound. A measurement is added to the output if and only if its vertex is in a connected component of a given size. LGSB does not guarantee that the complete output is consistent according to the speed bound Zheng (2015).

MEOS \(^7\) is an open-source C library engineered for mobility analytics built upon Zimányi et al. (2020b). It’s a highly versatile solution, providing a single source code that integrates with multiple programming languages. MEOS’s flexibility allows it to operate across diverse computing environments, ranging from edge to cloud-based platforms, and efficiently manage batch and stream processing. The library detects outliers using a heuristic method comparing maximum and minimum speeds.

Argosfilter Freitas et al. (2008) \(^8\) is an R package that offers a set of functions for working with trajectory data. The outlier detection in Argosfilter uses two different methods: one based on speed and the other based on location. The speed-based method is similar to the one provided in Zheng (2015), but the location-based method is based on the algorithm described in the paper Freitas et al. (2008). This method uses a set of spatial constraints, such as a minimum distance between two points or a maximum distance from a reference point, to identify and remove outliers from a trajectory.

Stmove Seidel et al. (2019) \(^9\) is an R package library. It provides construction functions, filter, and outlier detection functions. For the outlier filter, the KF is applied.

MVOOutlier Filzmoser and Gschwandtner (2017) \(^10\) is an R package that is a tool for identifying outliers in multivariate data. One of the primary strategies employed by this package is distance-based outlier detection, in which the distance of each observation from others or a central direction measure is calculated. Observations falling beyond a predefined distance threshold are then tagged as outliers. Among the methods provided by the MVOOutlier package include UniOD, which carries out univariate outlier detection for each variable individually, and DDoutlier, which performs depth-based detection of multivariate outliers. The package also incorporates the Minimum Covariance Determinant (MCD) method and the Mahalanobis distances method, which is adept at accommodating the covariance between variables. Furthermore, the Stahel-Donoho estimates present another robust way for location and scatter estimates, beneficial for outlier detection.

In the next Session, we present experiments performed in the libraries presented above.

\(^7\)https://libmeos.org and https://github.com/MobilityDB/

\(^8\)https://cran.r-project.org/web/packages/argosfilter/argosfilter.pdf

\(^9\)https://tinyurl.com/stmove

\(^10\)https://cran.r-project.org/web/packages/mvoutlier/index.html
<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ptrail</td>
<td>Hampel identifier</td>
<td>Python</td>
</tr>
<tr>
<td>MovingPandas</td>
<td>Kalman filter</td>
<td>Python</td>
</tr>
<tr>
<td>Scikit Mobility</td>
<td>Maximum speed</td>
<td>Python</td>
</tr>
<tr>
<td>Pymove</td>
<td>Maximum/minimal speed</td>
<td>Python</td>
</tr>
<tr>
<td>Scikit-learn</td>
<td>Local Outlier Factor</td>
<td>Python</td>
</tr>
<tr>
<td>Stmove</td>
<td>Kalman</td>
<td>R</td>
</tr>
<tr>
<td>Argosfilter</td>
<td>Maximum/minimal speed</td>
<td>R</td>
</tr>
<tr>
<td>MVOutlier</td>
<td>Minimum Covariance Determinant</td>
<td>R</td>
</tr>
<tr>
<td>MoveTk</td>
<td>Linear speed bound</td>
<td>C++</td>
</tr>
<tr>
<td>MEOS</td>
<td>Maximum/minimal speed</td>
<td>C</td>
</tr>
</tbody>
</table>

Table 1 Name, method and programming language used by each considered library

4 Experiments

This Section presents our method for constructing ground-truth in Subsection 4.1. We present an analysis of the benchmark results in Subsection 4.2.

4.1 Constructing Ground-truth

Constructing a ground-truth for trajectory data analysis is a task that presents several challenges. The difficulties include the cost, accuracy, and scale of the required ground-truth data. Additionally, cleaning the data to remove errors and noise can be time-consuming and may only sometimes produce reliable results.

We present a method that can be applied to the data originating from multi-sensor tracking technologies. Our proposed method offers a different approach to traditional methods for constructing ground-truth, such as manual annotation or data cleaning techniques like those described in Wu et al (2022); Yang et al (2018); Moosavi et al (2017). It also allows for data integration from multiple sensors, increasing the overall accuracy of the ground-truth. We consider speed and bearing to accommodate both errors in direction and speed. The Figure 1 illustrates the different errors one can encounter in a trajectory with respect to speed and bearing. The Algorithm compares the calculated speed and bearing with the recorded values present in the data. The input consists of speed and heading thresholds (tS and tH) and the file path. In lines 6–12, for each point in the file, firstly, we check if the IDs are the same, i.e., the points belong to the same trajectory. Secondly, we calculate the speed and bearing between the point and its successor. Later, we compare the newly calculated speed and bearing with the files’ input in line 1. If the difference in speed or heading is bigger than the inputted thresholds, the point is considered an outlier and added to the output array.

Our method involves cross-referencing the data from multiple sensors to generate a more accurate representation of the actual trajectory. We apply this method to data from modern multi-sensor tracking, such as GPS, AIS, ADS-B, Mode S, TCAS, and FLARM sensors. By combining the data from multiple sources, we can increase the overall accuracy of the ground-truth in the presence of errors or noise in individual sensors. When comparing data from different sources, we can analyze the data to see if there are any discrepancies or inconsistencies between all sources. In addition, statistical tests can determine whether the data is consistent with a particular hypothesis.
Algorithm 1: Ground-truth

Input: thresholdSpeed $tS$, thresholdHeading $tH$, FilePath file
Output: Outliers

1. Outliers ← {};
2. file ← READFILE (FilePath);
3. file ← ORDERBYIDANDTIMESTAMP (file);
4. speed ← 0;
5. bearing ← 0;
6. foreach point ∈ file.length do
7.     if $file[point].id == file[point+1].id$ then
8.         speed ← CALCULATESPEED(point, point + 1);
9.         bearing ← CALCULATEBearing(point, point + 1);
10.    if $|speed - file[point].sensorRecordedSpeed| ≥ tS$ OR $|bearing - file[point].sensorRecordedBearing| ≥ tH$ then
11.        Outliers.append(point);
12.    end
13. end
14. return Outliers

or model. For example, we can check the calculated speed with the speed given in the data.

Our datasets for this benchmark consist of multi-sensor trajectory data. The first source is AIS data: $^{11}$ AIS is the location tracking system for sea vessels. Additionally, the AIS data is collected from various ships, providing diverse trajectories to work with. This can be used for testing the robustness of different algorithms and methods and for evaluating their performance on different types of data. In this paper, we utilize a total of 4.3 GB. The Fig. 2 shows the raw data, and Fig. 3 shows filtering of outliers.

$^{11}$https://dma.dk/safety-at-sea/navigational-information/ais-data
The raw data collected in the OpenSky Network\textsuperscript{12} is stored in a historical database and used by researchers to study and improve air traffic control technologies and processes. The Fig. 4 shows the raw data.

We utilize two datasets from Huang et al (2019)\textsuperscript{13} consisting of GPS trajectory datasets of Southeast Asia from Singapore and Jakarta. Grab-Posisi covers over 1 million kilometers and contains more than 88 million points. The Images 5 and 6 show the raw data.

Table 15 shows the total number of records, the number of outliers detected in the cross-check, the correct points, i.e., points not considered outliers, and the percentage of outliers over total points. The cross-check method was used to detect the outliers.

\textsuperscript{12}https://opensky-network.org  
\textsuperscript{13}https://engineering.grab.com/grab-posisi
<table>
<thead>
<tr>
<th>DataSet</th>
<th>Total Points</th>
<th>Outliers</th>
<th>Normal Points</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jakarta</td>
<td>16798804</td>
<td>85275</td>
<td>16713529</td>
<td>1.523%</td>
</tr>
<tr>
<td>Singapore</td>
<td>9096633</td>
<td>68202</td>
<td>9028431</td>
<td>2.249%</td>
</tr>
<tr>
<td>OpenSky</td>
<td>5417090</td>
<td>2</td>
<td>5417088</td>
<td>0.000%</td>
</tr>
<tr>
<td>AIS</td>
<td>25714294</td>
<td>957</td>
<td>25713337</td>
<td>0.011%</td>
</tr>
</tbody>
</table>

Table 2: Datasets total points, outliers, correct points, and percentage of outlier over total points

OpenSky data does not have a significant amount of outliers. One possible reason for the low number of outliers in the OpenSky data could be the accuracy of the sensors used to collect the data. If the sensors are highly accurate, there may be fewer errors or discrepancies in the data, resulting in more occasional outliers. Additionally, the data may have been adjusted or corrected in some way to account for any errors or noise, further reducing the number of outliers. It is also worth considering the nature of the data itself.

4.2 Benchmark

The benchmark was run in ten libraries composed of MovingPandas, Scikit-mobility, Scikit-learn, Ptrail, PyMove, movetk, MEOS, Argosfilter, Stmove, and MOutlier. These were described in Session 3. Fig. 7. The data used in the benchmark was described in Session 4.1. The benchmark code is publicly available. 14

The European aviation industry Control (2022)15 developed methods to reduce carbon emissions to meet climate targets. An aircraft can fly an optimal flight path and use various technologies and infrastructure to minimize fuel consumption and carbon emissions. This might include using modern flight planning software and meteorological data to plan for a minimal amount of fuel, using green energy at the airport to power the aircraft on the ground, and using electric taxi solutions to minimize ground-based emissions. The aircraft would also fly an optimal climb phase, follow a fuel-efficient cruising level, and use idle thrust descent to reduce fuel consumption during descent, i.e., the aircraft should change its altitude as rarely as possible. Due to this standard, we consider the OpenSky dataset with only 2D rather than 3D data. However, it is essential to note that the assumption may not hold true in all circumstances. It may be necessary to use 3D data or incorporate altitude data in some cases to analyze flight patterns accurately.

In Figure 7, a comparative analysis of the run times for various libraries is presented across four distinct datasets: Jakarta, Singapore, OpenSky, and AIS. Notably, the Pymove library consistently exhibits the highest run times across all datasets, with an increase in computational time for larger datasets such as Jakarta and Singapore.

Libraries developed in R programming language appear to offer comparable run time. Similarly, libraries based on C++ (MoveTk) and C (MEOS) demonstrate run times similar to those of the R libraries. Furthermore, Scikit-learn exhibits competitive performance, likely attributed to its highly optimized Python implementation and algorithm. The MoveTk library displays an anomalous behavior in the execution time

15 https://www.eurocontrol.int/publication/objective-skygreen-2022-2030
with the AIS dataset. This irregularity could be attributed to the underlying process employed for data reading. Given these observations, careful consideration of run times is advised when selecting a library for trajectory data analysis and modification.

In Figures 8 and 9, we see the total run times across libraries developed in Python and R, respectively. What stands out is an association between the language of implementation and the efficiency of the library in question. To summarize, the choice of programming language appears to substantially impact the run time, emphasizing the need to consider language compatibility and efficiency when evaluating libraries for specific tasks.
In order to analyze the correctness results, Fig. 10 and 11 show the recall and F-1 scores of each library, respectively. The scores are based on comparison to the cross-check data and libraries output.

In Figure 10, the metric of interest is recall, which measures the ability of a library to identify true positives correctly, in this context, outliers within the dataset. A high recall score indicates that the library detects a large percentage of the true outliers,
reducing the likelihood of missing significant anomalies. Conversely, a low recall score suggests the library may be susceptible to false negatives, resulting in omitting non-outlier data points. Of particular note is the performance of MoveTK, which displays commendable consistency across various datasets and emerges with the highest recall rate. Argosfilter and Scikit-learn closely follow this. Contrarily, MVOOutlier and Ptrail exhibit significant variability in its recall rates, particularly underperforming in larger datasets. It suggests that both libraries may offer less reliable outlier detection capabilities, a factor worth considering when selecting a library for specific data analysis requirements.

The F-1 score is a metric used for comparing the overall performance of the different libraries. F-1 is a combination of precision and recall. The result is shown in Fig. 11. MoveTK continues to have the highest score across all datasets. This can be related to implementing the optimal Speed-bounded and Optimal Acceleration-bounded algorithms. Scikit-learn trails closely behind, likely benefiting from its Local Outlier Factor algorithm. In contrast, similarly to the recall, MVOOutlier presents a variable F-1 score, suggesting it depends highly on the dataset.

These results suggest that the most reliable library is MoveTk. MoveTk presents a consistently high recall and F-1 score with the lowest run time. Scikit-learn follows closely, presenting strong performance metrics, thus making it another viable option for similar analytical tasks.

5 Conclusion

In this paper, we evaluated the performance of ten libraries for outlier removal in trajectory data to guide data scientists and users on the existing libraries’ offerings. The benchmark includes MovingPandas, Scikit-mobility, Scikit-learn, Ptrail, PyMove, movetk, MEOS, Argosfilter, Stmove and MOutlier. The datasets for this benchmark consist of four massive datasets across multiple mobility domains, namely urban domain, air traffic, and marine. Our results suggest that the most reliable library is MoveTk. MoveTk presents a consistently high recall and F-1 score with the lowest run time. Scikit-learn follows closely, presenting strong performance metrics, thus making it another viable option for similar analytical tasks. Also, we have developed an approach for constructing ground-truth that involves cross-referencing the data from multiple sensors for speed and bearing. We applied this method to data from modern multi-sensor tracking technologies focusing on speed and bearing.

For future work, it would be interesting to explore and benchmark algorithms implemented but not integrated within libraries.

Declarations

This work was funded by the EU’s Horizon Europe research and innovation program under Grant No. 101070279 MobiSpaces.
References


Control E (2022) The economics of aviation decarbonisation towards the 2030 green deal milestone. Euro Control


Troficus M (2021) Hampel filter in python

Urrea C, Agramonte R (2021) Kalman filter: Historical overview and review of its use in robotics 60 years after its creation. Sensors


Yang S, Madsen M, Bednar J (2022) Holoviz: Visualization and interactive dashboards in python. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. SIGKDD International Conference on Knowledge Discovery and Data Mining


