UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Tiit Vaino

Anomaly Detection in CDR-Based Trajectories of the Mobile Cellular Network.

Master's Thesis (30 ECTS)

Supervisor: Amnir Hadachi, PhD

Tartu 2024

Anomaly Detection in CDR-Based Trajectories of the Mobile Cellular Network.

Abstract:

Mobile data is an excellent resource to approximate people's location and movement. This could be used by governments and private companies. For example, emergency services use people's phone locations to know where to send help. The approximate location is derived from combining mobile and cellular network connections information with cellular network cell (mobile mast antenna coverage area) locations. Multiple location events with consecutive timestamps, when put together, make one trajectory. It is crucial to ensure that datasets are clean to prevent costly decisions based on flawed analyses.

The main issue is that cell location in a database and in real life does not match because of human error or unsynchronized databases. This thesis proposes a model for anomaly detection in CDR-based trajectories using a Trajectory Anomaly Detection with Mixed Feature sequence (TAD-FM) approach. The training and testing were done using real data with integrated virtual anomalies, where some of the cell's locations are deliberately modified. Additionally, improvement has been introduced to the model to reduce the time complexity for training and prediction. The proposed model was capable of labelling and detecting 66% of the cells with wrong location data as an outlier.

Keywords:
Machine Learning, Artificial Intelligence, Clustering Algorithm, Neural Network, Autoencoder, TAD-FM, DBSCAN, Cellular Network

CERCS: P176 Artificial intelligence; P170 Computer science, numerical analysis, systems, control.

Anomaaliate tuvastamine mobiilsidevõrgus CDR-põhistes trajektoorides.

Lühikokkuvõte:

Mobiilse kõne- ja andmeside andmestik on suurepärane viis inimeste asukoha ja liikumise ligikaudseks hindamiseks. Seda saaks eetiliselt kasutada nii valitsuste kui ka eraettevõtete poolt. Näiteks kasutavad hädaabiteenused inimeste telefonide asukohti, et teada saada, kuhu saata abi. Ligikaudne asukoht saadakse mobiili ja mobiilivõrgu andmete kombineerimisega mobiilivõrgu kärgede (mobiilimasti antennide katvusalade) asukohtadega. Mitmed asukoha sündmused koos järjestikuste ajatemplitega saab kokku panna üheks trajektooriks. Neid asukohti ja trajektoore võiks kasutada mitmesuguste valitsuse või ärivaldkonna küsimuste analüüsimiseks. On oluline tagada, et andmekogumid oleksid puhtad, et vältida vigaste analüüside põhjal kallite vigade tegemist.

Põhiline probleem on selles, et kärje asukoht andmebaasis ja reaalses elus ei kattu, sest on tehtud inimlik viga või andmebaaside pole sünkroonis. Antud magistritöö pakub välja mudeli anomaaliate tuvastamiseks CDR-põhistes trajektoorides, kasutades *Trajectory Anomaly Detection with Mixed Feature sequence (TAD-FM)* ehk siis trajektoori anomaaliate tuvastamine kombineeritud tunnusjoonte jadaga lähenemisviisi. Mudeli treenimine ja testimine viidi läbi reaalsete andmetega, kuhu olid integreeritud virtuaalsed anomaaliad, kus mõne kärje asukohti oli tahtlikult muudetud. Lisaks on mudelile tehtud täiustusi, et vähendada treenimise ja ennustamise ajalist keerukust. Pakutav mudel suutis märgistada ja tuvastada 66% kärgedest, millel olid valed asukohaandmed, kui kõrvalekalded.

Võtmesõnad:
Masinõpe, Tehisintellekt, Klasterdamisalgoritm, Neuraalvõrk, Autokodeerija, TAD-FM, DBSCAN, Raadiovõrk

CERCS: P176 Tehisintellekt; P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine.

Contents

Terms and notions

CDR Call Detail Record. 2, 3, 7, 8, 19, 22

DBSCAN Density-Based Spatial Clustering of Applications with Noise. 2–4, 11, 15, 16, 20–22, 28–37, 39, 47

GRU Gated Recurrent Unit. 20

HMM Hidden Markov Model. 19

k-NN k-Nearest Neighbors. 18

MLP Multilayer Perceptron. 20

PCA Principal Component Analysis. 19

RNN Recurrent Neural Network. 20

SNN-CAD Similarity-based Nearest Neighbour Conformal Anomaly Detector. 18

ST-RNN Spatial-Temporal Recurrent Neural Network. 20

STL Spatial-Temporal Laws. 19

SVM Support Vector Machines. 19

TAD-FM Trajectory Anomaly Detection with Mixed Feature sequence. 2, 3, 9, 11, 21, 22, 26, 27, 29, 30, 32, 37, 39

Introduction

Overview

A mobile phone is a part of our everyday lives. In 2022, 73% of the world's population had one[1]. They generate a lot of data per device. It was about, on average, 15 GB per month in 2022, which is expected to rise to 46 GB by 2028 [2]. With all this data is also a lot of metadata that could be used for different purposes.

The interesting part of this metadata is spatial-temporal data, which has many use cases. In the Call Detail Record (CDR) is saved events timestamps with a phone's approximate location. These could be put together into one trajectory. Having a lot of CDR data enables to construct a lot of trajectories, which gives many possibilities for research.

Much research has been done to utilize Call Detail Records. It is known that different aspects can affect CDR data, for example political, social and cultural events [3]. Also, people's behaviour like commuting between home and work [4]. It is also possible to estimate population density and its changes [5].

With all those tools, governments could solve a lot of issues. One use case is saving people's lives. Seeing a person's near real-time location gives emergency services the possibility to find people in need quickly. This could be used because of natural or human-caused disasters when there is a threat to people in a given area who need notifying to evacuate. The more everyday situation is that a person needs emergency services like an ambulance or police. With the right methods, the CDR data gives the right tools for epidemiological specialists to monitor disease spreading in the present and future [6, 7, 8].

Population density estimations and commuting patterns could give city planners a hint about which city parts need more attention. For example, infrastructure needs updating because of more traffic. Or there is a need for more public services like schools and hospitals because of rising population density.

A lot of companies could also benefit from processed CDR data. For example, it allows understanding what type of people are surrounding their business [9]. This gives them knowledge on improving their services or products because they understand the target audience better. It also allows companies to make more targeted advertisements, decreasing marketing costs and raising the target audience's visibility. CDR data could also help companies better understand their products and services. For example, taxi

companies could give time estimates for their rides [10].

CDR data is coming from telecommunication companies. So they should be interested in processing this data and helping governments and companies benefit from the knowledge that CDR data provides. There are some problems with the CDRs that bring the impact down.

Problem statement

The problem with the CDR data is that the information about the cell (mobile mast antenna coverage area) location could be incorrect. This impacts all applications that are using this corrupted data. The issue effect on the estimated path and locations can be seen in Figure 1.

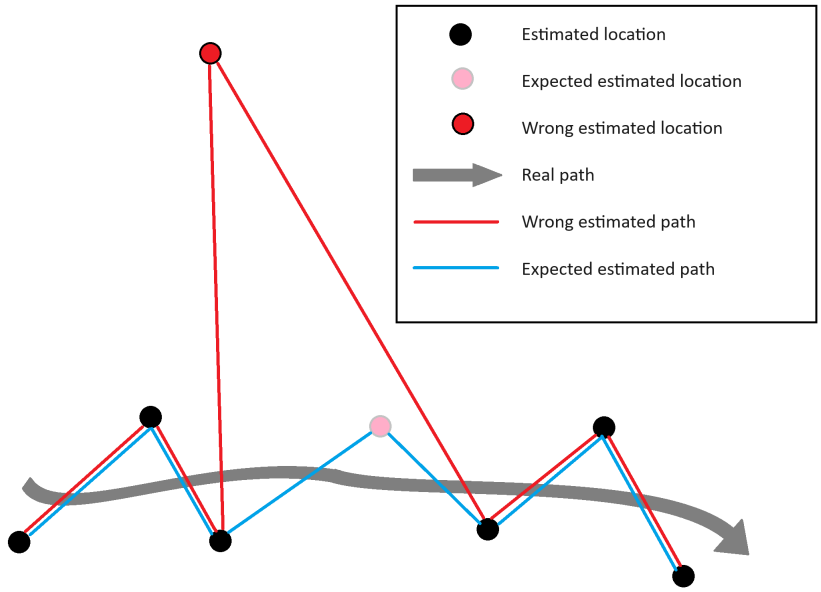

Figure 1. Broken cell affect on location and trajectory estimation.

Those wrong entries about the cellular network mostly come from human error. Some technicians have made a typo in cell coordinates or assembled cell antenna wires in the wrong order. Another problem could be that databases are out of synchronization. This means when some changes are done, then at the same time, in the different databases, the same cell could have a different location.

Wrong location and trajectory estimates could affect the users of CDR data. For

example emergency services input data could be compromised, therefore putting lives in danger and wasting money. Wrong locations affect population density estimations: in the expected area, there are fewer people, and in the wrong area, there are more people. This introduces bias into the data, so governments and businesses could make wrong decisions that can be costly and introduce inconvenience to society. Fixing those errors is crucial for telecommunication companies to offer the best quality data to their customers.

Contribution

This thesis focuses on finding cells with the wrong locations. They affect trajectory patterns; therefore, short segments of people's trajectories are used to find these cells.

Modern artificial intelligence methods are used to solve the problem. The final Trajectory Anomaly Detection with Mixed Feature sequence (TAD-FM) model was inspired by its original article [11]. Updating the preprocessing and clustering algorithm improved the TAD-FM. A half-synthetic dataset was created from the real dataset to train and test the model, where some of the cell's locations were altered.

The main result of this thesis is a promising model that can be used in real-life business processes to detect anomalistic cells. In addition, a list of improvements points was made to the final model.

Roadmap

The thesis is structured into five chapters. They will describe the background, related works, and used methods with final results and discussions.

Those chapters are:

- Chapter 1 teaches the reader how a cellular network works and what kind of data it generates.
- Chapter 2 introduces different researches on how to find anomalies. It is divided into three categories: statistical learning techniques, clustering techniques and deep learning techniques.
- Chapter 3 shows what is inside the TAD-FM and discusses how data was preprocessed and the model was trained in different parts.

- In Chapter 4, the reader can see the model training results and how well it finds the anomalistic cells.
- Chapter 5 will analyze if a developed model is usable in the real world and what can be improved.

At the end of the thesis are the appendices with extra information about the model and development. This research was done in cooperation with Reach-U[1], who provided data and hardware. To make the text more readable, Grammarly was used throughout the thesis [2].

[1] https://www.reach-u.com/

[2] Grammarly was used in the time range from 15.04.2024 to 15.05.2024

1 Background

This section simplistically describes how cellular networks work, what kind of data was used for this thesis and what could be derived from this information. Furthermore, it highlights the potential challenges associated with the data. At the end are described autoencoder and DBSCAN, two important parts of TAD-FM.

1.1 Cellular Network

Cellular networks enable us to call anybody from anywhere. For that, many antennas are needed to cover large areas. Usually, there are three antennas to cover 360° on one mast. An example of this can be seen in Figure 2. Each antenna is servicing one area that is named a cell.

Figure 2. Cellular network mast with antennas [12].

When those antennas are combined, they make a cellular network. In Figure 3, it can be seen how the cellular network is designed. Phone 1 represents the usual use case, where one device in a given cell talks to the antenna that services this area. However, there are challenges with Phone 2 and Phone 3, which are on the border of the two cells, so the connection hops between antennas that can be on the same mast (Phone 2) or a different mast (Phone 3). Based on this information, it could be said approximately where the phone is located.

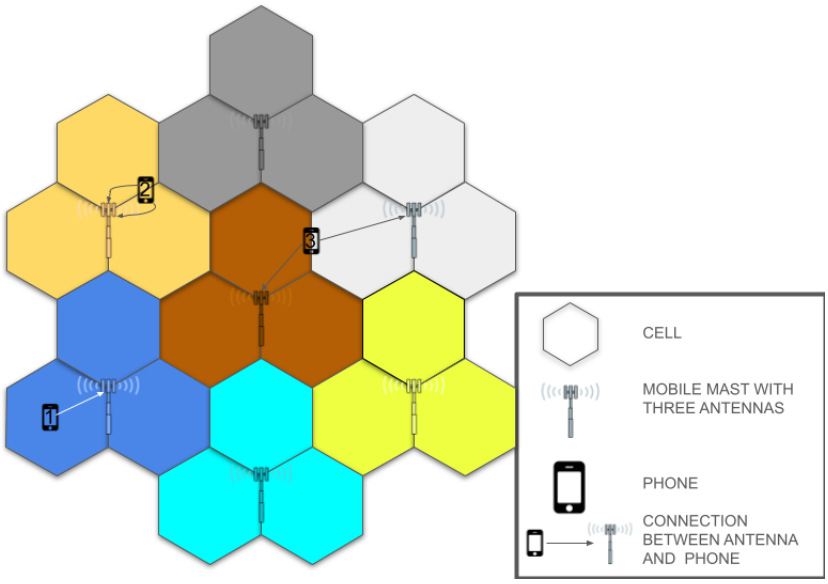

Figure 3. Phones connect to a different antenna in the cellular network.

Localisation can be done in a couple of different ways. The first would be the least accurate but the most straightforward. For example, Phone 1 is connected to the dark blue mast's antenna, which means it is in one of the three dark blue cells, so the mast location is chosen because it is in the middle. The second way would be to take the cell's centroid. Phone 1 is connected to one of the dark blue mast's antenna covering the bottom left dark blue cell, so it can be assumed that the phone location is this cell's centroid. The third way would be more complicated. It would use the ping time between the antenna and the device. It is called timing advance. From that, it can be calculated how far the device is from the antenna, forming a potential sector where it could be located. Those sectors can be seen in Figure 4.

Trajectories with different characteristics could be formed based on those three device localisation methods. When a phone moves through the cellular network, it connects to different antennas, and the distance from the connected antennas also changes. These changes could be used to track phone movements using the localisation methods above. The effect of those localisation methods on a predicted trajectory can be seen in Figure 4.

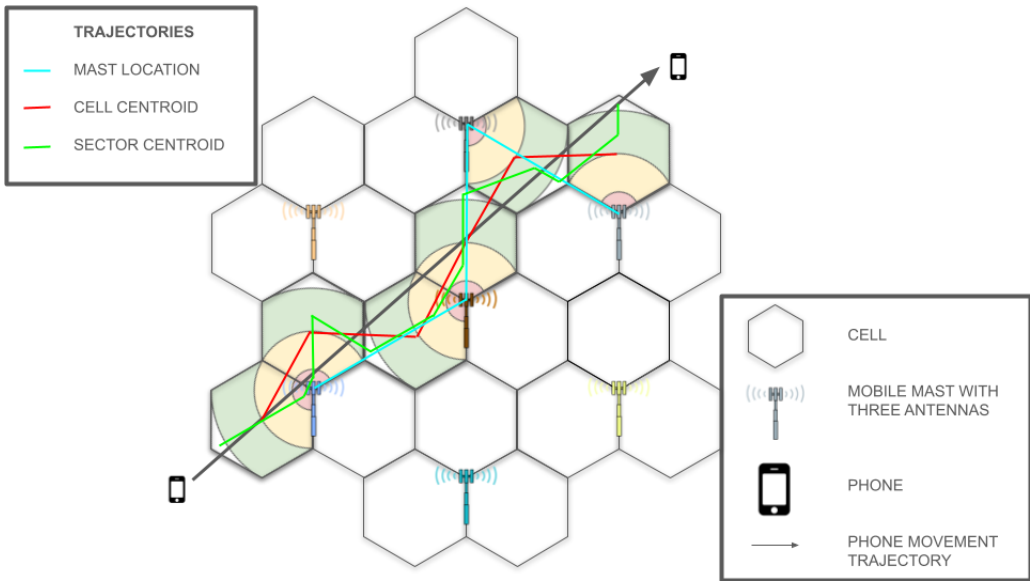

Figure 4. Phone estimated trajectories compared to real trajectory.

All this information about phone activity is stored in two datasets. One dataset includes information about antennas and their characteristics. The other dataset includes information about phone connection characteristics.

1.2 Cellular Network's Cell Data

Each cell in a cellular network has its own characteristics. Any changes in these are manually stored. Human errors can occur when inserting the values or assembling the antenna. Errors can also appear due to network system design because there are time windows where information in different databases is not synced. Each cell has a cell identification number, cell antenna coordinates and address, antenna azimuth and other info about the antenna. It is possible to calculate the area the antenna is covering (a cell), which can be seen as hexagons in Figure 3 and Figure 4. When there are errors in those values for one reason or another, there can be anomalies in phones' trajectories connected to this antenna with problematic values in the database.

In Figure 5 can be seen one example of how an anomalistic cell could affect a phone's trajectory. The problem is that the red cell's antenna location was wrong. Its new location is shown by the white arrow. The result of this can be seen in the estimated trajectories. All three have now "V" shaped segment in their trajectory. This segment causes different anomalistic behaviour in movement characteristics: long distance, really fast movements

and big accelerations.

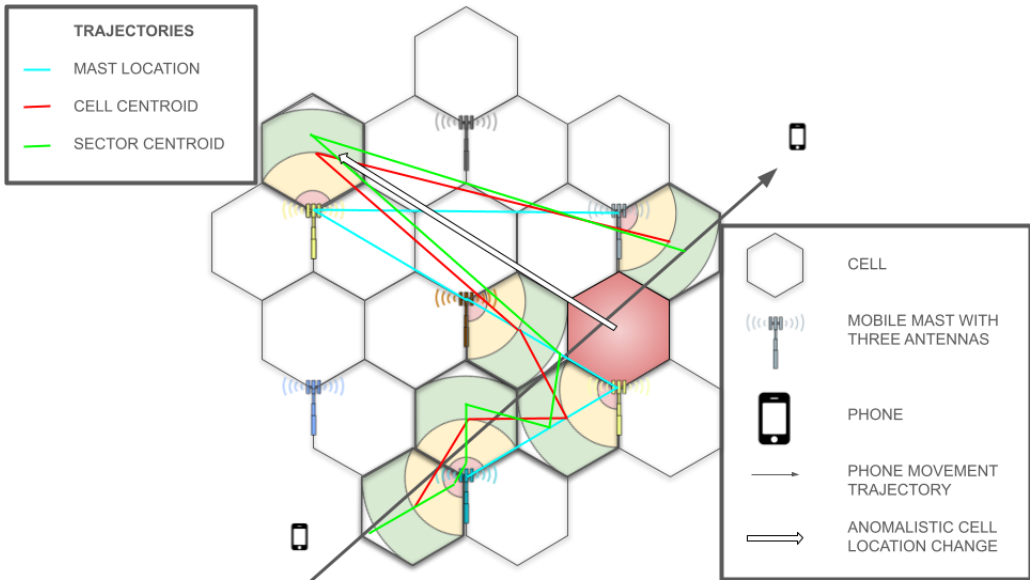

Figure 5. Phone estimated trajectories compared to real trajectory with anomalistic cell.

1.3 Mobile Data

Each time a phone connects to an antenna, an event is created. This event information is automatically saved to a database. Each entry to the database includes metadata about the connection. Essential values are the phone identification number, what kind of event it was, the cell identification number to which the phone was connected, and the timestamp when the event happened. There is also timing advance, which can be used to enhance localisation.

When merging those events with the cell table, it is possible to locate approximately where this event happened. Moreover, we get trajectories when putting merged information in order based on timestamp, as shown in Figure 4. In addition to the phone's trajectories, it is also possible to get its movement characteristics. For example, average speed, acceleration, direction, and changes in all those. However, when there is an error in data, those characteristics are also affected.

1.4 Autoencoder

An autoencoder is an unsupervised neural network where the model tries to output the same result as the input [13]. In Figure 6, it can be seen how the autoencoder is built. The autoencoder effect comes from the fact that the middle layer is smaller, so the autoencoder has to generalise.

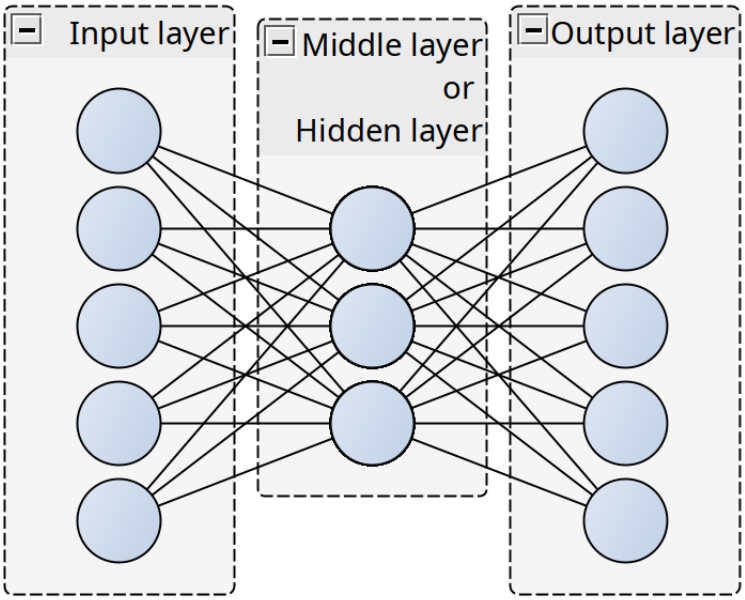

Figure 6. Autoencoder [11].

1.5 DBSCAN

In 1996, was published the article "A density-based algorithm for discovering clusters in large spatial databases with noise" [14] where was introduced a well-known clustering algorithm, Density-Based Spatial Clustering of Applications with Noise (DBSCAN). Its design already included noise detection helping to filter outliers out. Its ability to detect arbitrary shapes has attracted a lot of other researchers to use DBSCAN as part of their methods, as seen in Section 2.

The logic of DBSCAN is described in Figure 7.

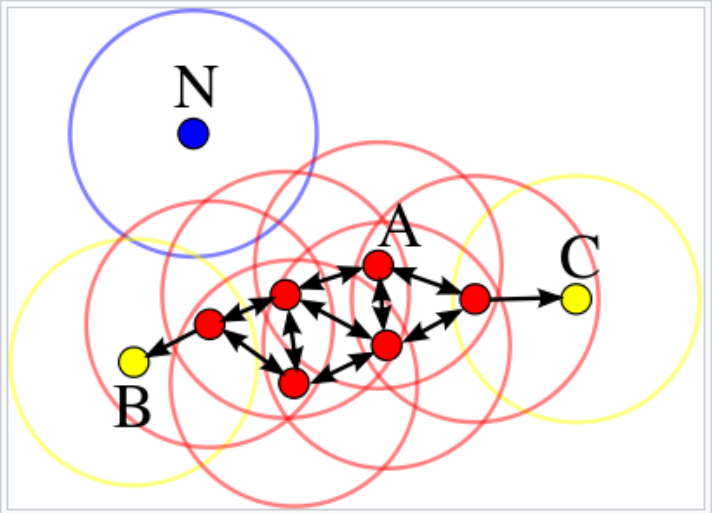

Figure 7. DBSCAN logic chooses core (A) and border points(B,C) and leaves out noise (N)[15].

A more detailed algorithm can be seen in Algorithm 1. For DBSCAN is necessary two variables: max distance to the neighbouring point (ϵ or eps) and the minimal number of neighbours (*MinPts*). In data, DBSCAN has three different types of points (colours in Figure 7): core points (red), border points (yellow) and noise (blue). Core points are points with at least *MinPts* of neighbouring points under epsilon distance (marked as a circle). Border points have under *MinPts* of neighbours, but at least one of the neighbours is a core point. Points marked as noise have fewer than *MinPts* of neighbours, and none is a core point. Clusters are assigned based on the core points. Core points are points in each other's neighbours and belong to the same cluster. Border points belong to the same cluster as their neighbouring core point. Points that are noise do not belong to any cluster.

Algorithm 1: DBSCAN Algorithm

Input: DB: Database
Input: ϵ: Radius
Input: MinPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined

```
foreach point p in database DB ;                    // Iterate over every point
do
    if label(p) = undefined then
        continue ;                                  // Skip processed points
    Neighbors N ← RangeQuery(DB, dist, p, ϵ) ;      // Find initial
    ;                                               // neighbors
    if |N| < MinPts then
        label(p) ← Noise ;                          // Non-core points are noise
        continue;
    c ← next cluster label ;                        // Start a new cluster
    label(p) ← c;
    Seed set S ← N \ {p} ;                          // Expand neighborhood
    foreach point q in S do
        if label(q) = Noise then
            label(q) ← c;
        if label(q) ≠ undefined then
            continue ;                              // Previously processed
        label(q) ← c;
        Neighbors N ← RangeQuery(DB, dist, q, ϵ);
        if |N| ≥ MinPts ;                           // Core-point check
         then
            continue
        S ← S ∪ N
```

2 Literature overview

This section introduces current state-of-the-art methods for finding anomalies that could be used to solve finding anomalistic cells from cellular network. It is done in three categories: statistical learning techniques, clustering techniques, and deep learning techniques. In each category, a variety of methods from different articles is introduced. Some of them find anomalies in a set of trajectories, and some of them from a set of data points.

2.1 Statistical Learning Techniques

Statistical algorithms are one of the most intuitively understandable ways to find anomalies. They could be used wholly or as one step in the whole process. There has been much research in this field, and this subsection will introduce some of them.

In 2012, a paper, "An online algorithm for anomaly detection in trajectory data" [16], was published. The task was to identify unusual sea vessel routes. Their solution was short and computationally easy. Authors even say it could be used as a one-line solution. Based on initial data, a typical route was calculated. Now, they take each route and calculate the abnormality score for each route utilizing a statistical test. They used simulated data and ship trajectories travelling through the English Channel for testing, and the test results were promising.

Similarly to the previous paper [16], an article "Anomaly detection in maritime data based on geometrical analysis of trajectories" [17] dealt with anomalous ship trajectories. Their proposed unsupervised method takes trajectories and gives them an abnormality score. They first find the shortest path using the A* algorithm to calculate the score. Then, key features, such as trajectory length, were extracted for each trajectory under observation. Using those features, patterns were compared with the shortest path to get the abnormality score.

The following paper, "Anomaly detection in trajectory data for surveillance applications" [18] by Rikard Laxhammer, also automates anomaly detection of sea vessel trajectories. Rikard introduced Similarity-based Nearest Neighbour Conformal Anomaly Detector (SNN-CAD) algorithm. This algorithm takes data points, compares them to known points (similarly to how in the k-Nearest Neighbors (k-NN) algorithm) and then, based on a given threshold, says if it is abnormal.

The paper "Trajectory-Based Anomalous Event Detection" [19] introduced the pos-

sibility of using a single-class Support Vector Machines (SVM) to find outliers. The approach was based on SVM clustering and they used a single-class variant for it. They used generated data and vehicle and people trajectories extracted from the video for testing. In both cases, it was able to find outliers. A good thing about this model is that it can be known how many outliers there are.

Unlike previous methods described above in this section of the paper "STL: Online Detection of Taxi Trajectory Anomaly Based on Spatial-Temporal Laws," [20] time info is also used in addition to spatial data. For this, an algorithm called Spatial-Temporal Laws (STL) was introduced. STL looks at how given trajectory characteristics differ (time, distance and displacement vector) from previous trajectories. When the observable trajectory is out of the range of expected values, it is considered abnormal. For training and testing, taxi movements were used to spot fraudly behaving taxis making too long trips.

To find anomalies, it is also possible to use a Hidden Markov Model (HMM), which has been shown in the article "Anomaly Detection on Collective Moving Patterns: A Hidden Markov Model Based Solution" [21]. They took a path and the probability that this sequence is possible was calculated using the HMM. This path was marked as an outlier if this value was lower than the threshold. They used people's movements for training and testing as generated and from real-world data. For the last one, they used Principal Component Analysis (PCA) to reduce dimensionality.

In all the discussed statistical methods, it can be seen that some type of abnormality score is calculated. These abnormality scores were used to find the outliers. With a given threshold, outliers can be filtered out.

2.2 Clustering Techniques

Different clustering methods could be used to find outliers in some datasets. Outliers are points that do not have similar points in their proximity. In this subsection, some methods from the research done in the field will be introduced.

The article "CDR Based Temporal-Spatial Analysis of Anomalous Mobile Users" [22] introduces how hierarchical clustering could be used for anomaly detection. In their approach, they had to define several clusters and then look into those clusters. Their data was CDR, in which they detected anomalous users.

K-means++ was used in the method presented in the article "Local Outlier Detection for Multi-type Spatio-temporal Trajectories" [23]. They used different trajectory datasets

from the solar astronomy domain, and their goal was to find anomalous subsegments. In the first stage, the trajectory was split into segments, and then features were extracted. In the second stage, clustering was done, and now they have template trajectory segments. In the last step, they calculated the abnormality score and the segment distance from the template.

The article "A Framework for Spatial-Temporal Trajectory Cluster Analysis Based on Dynamic Relationships" [24] also used spatial-temporal data (truck and human trajectories). Clustering was done using DBSCAN; they looked at each event's location and the haversine distance between those events.

DBSCAN was also used in another research described in the paper "DensityBased Clustering Based on Hierarchical Density Estimates" [25]. They updated the DBSCAN algorithm to find the dataset hierarchical structure.

As seen, different clustering methods are usable. The aforementioned methods use the distance between points for clustering. This means between the two points, a distance is calculated, and clusters are formed between those points that are close to each other.

2.3 Deep Learning Techniques

The most novel topic in anomaly detection is deep learning. Different neural models could be used for noise detection. This subsection will describe some of them with examples from different papers.

Recurrent Neural Network (RNN) is usually used with time-series data, so it seems to be a logical choice when dealing with sequence-based data as trajectory. In the article "Spatial-Temporal Recurrent Neural Network for Anomalous Trajectories Detection" [26], the model Spatial-Temporal Recurrent Neural Network (ST-RNN). Their model used Gated Recurrent Unit (GRU), attention units and simple Multilayer Perceptron (MLP). They used their model on taxi trajectories and assigned each the probability of being anomalous.

A similar problem was solved in the article "A deep encoder-decoder network for anomaly detection in driving trajectory behaviour under spatiotemporal context" [27]. There, they segmented road networks into spatial-temporal units. For each unit, characteristics like speed, acceleration and direction were calculated. Then, each unit's characteristics were given an autoencoder, and it was calculated later if it was possible to reconstruct the same output as the initial input.

An interesting approach was chosen in the paper, "Unsupervised learning trajectory anomaly detection algorithm based on deep representation" [11]. Researchers used a mixed approach to detect anomalistic trajectories in a dataset. They used two autoencoders in sequence and DBSCAN on top of it. It is called TAD-FM and uses unsupervised learning. Autoencoders were used to make a summary of each trajectory, reducing dimensions. Also, there is preprocessing before the autoencoders, where each trajectory is segmented, and characteristics like variations of velocity, distance, angle, and acceleration are calculated for each trajectory.

All described models used neural network models to find summarising features. There were end-to-end models, but they also incorporated into other methods. Also, movement was described similarly in preprocessing with several values.

3 Methods

This section will describe TAD-FM architecture, preprocessing data, and how the final model was trained. The method was inspired by the paper [11], which addressed anomaly detection on GPS traces. Since the CDR location data is almost similar to GPS traces but it has significant geolocation inaccuracy and sparseness in time and space. It was decided to build the model for this thesis following the same philosophy. Adjustments and modifications were necessary in the application of techniques to ensure suitability for CDR-based trajectories.

3.1 Model

This subsection will describe TAD-FM. A short overview is given of a high-level view of the architecture with examples of input data. The preprocessing part includes data cleaning, making trajectory segments, calculating metrics for each cell and normalizing. The autoencoder part includes two autoencoders with mixed feature fusion. The last part is clustering, where DBSCAN is used.

3.1.1 High-Level Architecture

On a high level, the TAD-FM is separated into three layers: preprocessor, autoencoders, and clustering. Each of them has a unique part to do in solving the problem. Model architecture can be seen in Figure 8.

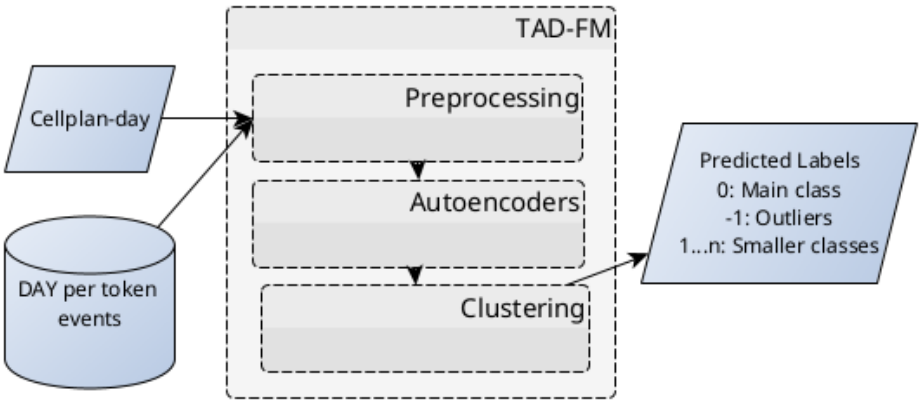

Figure 8. TAD-FM high-level architecture.

The model takes two datasets, as shown in Figure 8. Phone users create mobile events dataset rows. Each connection between the user's phone and cell antenna is recorded as one row. It contains info about the user, connection, and cell. Table 1 shows an example of the mobile events dataset. Cell plan data contains info about mobile masts' locations and antenna characteristics. The cell data example is in Table 2. In both tables only relevant columns are shown, unnecessary columns are dropped for both datasets in the preprocessing part.

Table 1. Example of a mobile events.

user_id	**timestamp**	**cell_id**
1	1715075250000	100-200-30000-40004
2	1715075251000	100-200-30000-40000
1	1715075252000	100-200-30000-40001
1	1715075253000	100-200-30000-40003
4	1715075254000	100-200-30000-40002

Table 2. Example of a cell plan.

cell_id	**latitude**	**longitude**
100-200-30000-40000	12.123456	21.654321
100-200-30000-40001	12.223456	21.654321
100-200-30000-40002	12.323456	21.754321
100-200-30000-40003	12.423456	21.654321
100-200-30000-40004	12.534567	21.654321

Those datasets are cleaned and merged. Unimportant events are removed and five points trajectories are formed. Based on these trajectories, each cell's characteristics are calculated. In the end, these characteristics are normalized. All this was done in the preprocessing layer.

Normalized characteristics are then fed through an autoencoder layer, from which a short summary is generated for each cell. This summary is then given to a clustering algorithm. After the algorithm, each cell is assigned a class number. -1 indicates the value is an outlier, and 0 and above indicate some cluster. Now, those outliers could be given a closer look by a human who could fix this issue.

3.1.2 Preprocessing

There are two inputs: mobile events and the cell plan. In Figure 9, the simplified flow can be seen. A more detailed flow can be seen in Appendix I. In the original article [11], researchers took trajectories and calculated metrics for each trajectory. For this thesis, relevant trajectories were summarized into one set of metrics for each cell.

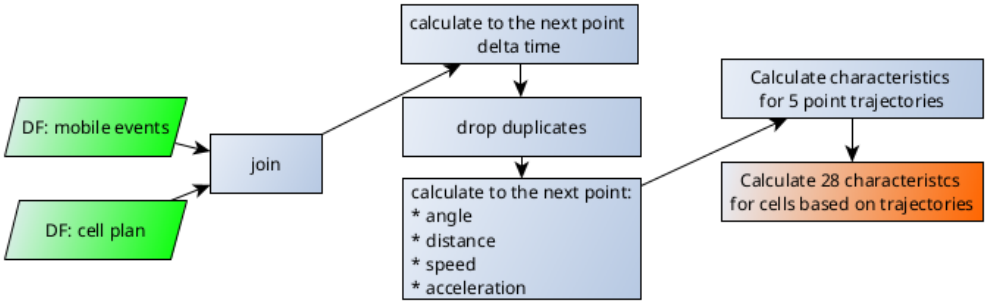

Figure 9. Simplified datasets preprocessing flow.

As can be seen, two datasets are merged to add a location to each event. The cell site location was chosen because this is the value in which the error is searched, and it does not need any more computation. To reduce the need for computation even more, irrelevant points are dropped. Those are the repetitive middle points where the cell does not change. Then, the time difference and distance with the next point are calculated. Angle change is calculated between the last, current and following points. Speed is calculated using time and distance, and acceleration is calculated using the change in velocity at the current point. Thus, for each point, there is a quantification of distance, angle, speed, and acceleration.

The five-point trajectories' characteristics are calculated using those four values for each point. An example of a five-point trajectory with chosen metrics can be seen in Figure 10. Vectors in this figure how values like distance and speed angle are measured. The movement characteristics are measured between the current and the next point, and the angle and acceleration is measured between the last, the current and the next point. Each trajectory comes from one phone movement by looking at two points before and two after the current point. Only relevant metrics that give information about the middle cell are selected at each point. About each trajectory, there are four metrics describing them:

- Sum of distances of cells C-2, C-1, C0, C1, C2.

- Sum of velocities of cells C-1, C0.
- Angle of the C0 cell between the cells C-1, C1.
- Acceleration difference in C-1, C1.

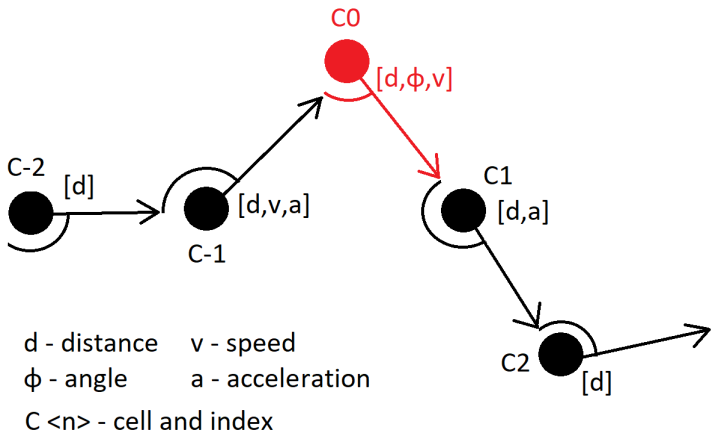

Figure 10. Example trajectory with used metrics for each cell.

For each cell, trajectories are taken where the middle point is the given cell. Using those trajectories, four metrics and 28 cell metrics are calculated. Number 28 comes from seven statistics being calculated over four metrics of the given cell's trajectory. Those seven statistics are:

- minimal value,
- maximal value,
- standard deviation value,
- 25 percentile value,
- median value,
- 75 percentile value.

To make the autoencoder part perform even better, those metrics are normalized. It was needed because each of the 28 values are on its own range. To solve this a Min-Max scaler were chosen, because it compresses the range between the values 0 and 1.

Its calculation can be seen in Equation (1).

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{1}$$

- X_{scaled}: Scaled value of the feature X
- X: Original feature value
- X_{min}: Minimum value of the feature X in the training dataset
- X_{max}: Maximum value of the feature X in the training dataset

3.1.3 Autoencoders

Autoencoder layer architecture can be seen in Figure 11. In TAD-FM, it is used as a summarizer that reduces dimensionality, therefore reducing computation needed in the clustering layer.

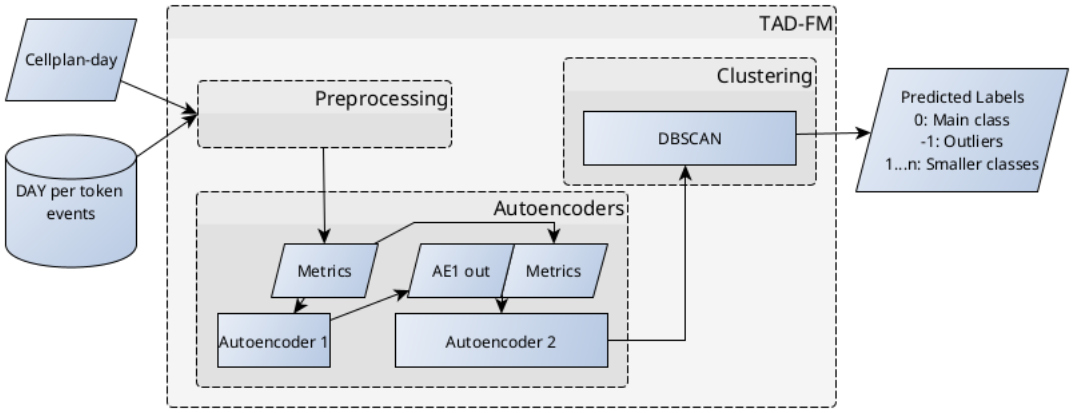

Figure 11. TAD-FM autoencoders and clustering.

TAD-FM has two autoencoders, as shown in Figure 11 and Figure 12. The TAD-FM autoencoders will generalise 28 values to 15 values. The first autoencoder takes those 28 normalised metrics as input and tries to learn them. Because the middle layer has 10 neurons, the model has to generalise. The second autoencoder does the same process as the first one with different inputs and a different size hidden layer. The second

autoencoder takes the same input as the first layer and concatenates the first layer's middle-layer output. So, in total, the second autoencoder has 38 values as input. Its middle layer has 15 neurons. Later, after training both autoencoders, the output is taken from the second autoencoder's middle layer.

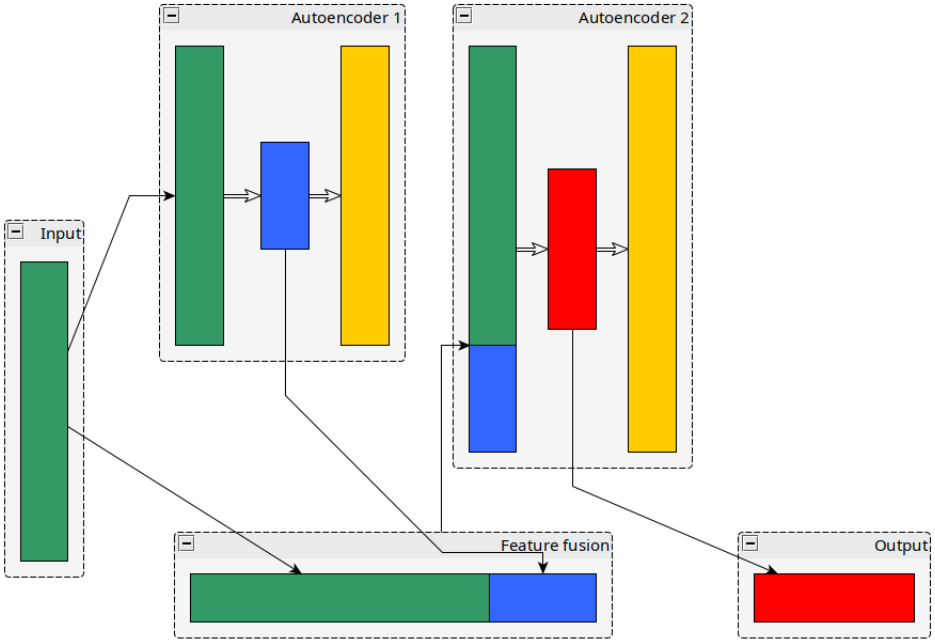

Figure 12. TAD-FM autoencoders fusing features. The size of the boxes represents the real layers size

To train both models, a loss function and optimizer are needed. Equation (2) shows a loss function used for the original paper's first and second autoencoder. x represents input, and y represents output of a autoencoder. ADAM [28] optimizer was used as the optimizer for both autoencoders, because it is adaptive and converges fast.

$$L(x, y) = -\sum_{i=1}^{n} (x_i \log(y_i) + (1 - x_i) \log(1 - y_i)) \tag{2}$$

3.1.4 Clustering

Those 15 summarised values from the second autoencoder middle layer was given to the clustering algorithm to find cells that form clusters and outlying cells. For this, a density-based clustering algorithm DBSCAN was used.

The goal of the DBSCAN clustering was aimed for broken cells to come out as noise and other points to belong to the main cluster.

3.2 Training and test dataset

The autoencoder and the clustering part need data to learn and predict something. To evaluate the models, this data needs to include broken cells. This subsection describes how a training and test dataset was created.

Firstly, finding the ratio between normal and broken cells was necessary. It was a balancing between the two arguments. The first one was that not too many normal cells should not be affected by broken cells. It is a problem, because normal cells that are near the broken cells, their trajectories are affected also. This will result in changes in representing 28 metrics. The second one was that when there were very few broken cells, it was hard to evaluate how well the model performed.

In the data, there were not enough known errors in the cell plan, so it was decided that creating a set of broken cells was necessary. A possibility would have been to find those real-life broken cells, but it would have taken a lot of resources.

Cells' locations were broken in three different ways, which could also happen in real life. The first way is for two cells to switch their locations. The second way would be for latitude and longitude to be switched. The third way would be that the latitude or longitude is wrong. Those three changes were done in the cell plan dataset to a small amount of cells, so those altered cells were mixed with real-life data.

Cell location change affects the trajectories of those mobile phones connected to broken cells. Therefore, cell 28 representative metrics are affected. Now, there are three types of cells in the cell plan. Broken cells, where location data is wrong. Affected cells, where there is a broken cell, one or two jumps away in the trajectory. Lastly, normal cells, which have not been altered nor affected.

Two different days were chosen to make a training and test set. On the first day data, the model was fine-tuned and trained. On another day, the model was only evaluated.

Choosing two different days ensured that there wass at least some variability between the test and training set.

3.3 Tuning the Model

Initially, the TAD-FM autoencoder layer was tested on a small dataset to get initial results fast. Then, the model was fine-tuned. Lastly, the model was trained on two weeks of real-life data, where no modifications were made and saved. After that, DBSCAN was fine-tuned to find the best parameters, and then the clustering algorithm was usable on necessary datasets.

3.3.1 Tuning Autoencoders

For the model, fine-tuning was tested by changing layers, the number of neurons and learning parameters. It was found that the base model's architecture is better than variants where some changes were made. So, only learning parameters needed to be tuned.

There were two learning parameters for both autoencoder layers separately: the number of epochs and the learning rate. It is a challenge to find the best parameters. An evolutionary algorithm was used to find those parameters. The paper "An Overview of Evolutionary Algorithms for Parameter Optimization" [29] describes different Evolutionary algorithms. In this thesis, a genetic algorithm was used, which was implemented in Python by Daniel Tucker in the *evolutionary-algorithm* Python library [30].

This genetic algorithm implementation at first generates a population of random pairs of epoch and learning rate values in a given range. Then, those pairs are given a score based on how well the model trains described by the TAD-FM custom loss function. The best values are chosen as parents of the new population. Lastly, new offspring are derived from those parents by mixing values and mutating those. Now, there is a new population, and a new cycle starts. This is done until there are no improvements or a maximum number of cycles has been reached.

After the genetic algorithm had found the best parameters, the autoencoder's loss functions graphs were evaluated, and based on that, necessary adjustments were made with parameters to avoid over-fitting. In the final step, the first autoencoder is trained on two weeks of data, and then the second autoencoder is trained on concatenated first autoencoder output and two weeks of data. After that, the model is saved for later usage to avoid unnecessary retraining.

3.3.2 Tuning Clustering

To get the best results out of DBSCAN, the best parameters combination needs to be found. For this, two parameters *MinPts* and ϵ were fine-tuned. Also, the most suitable distance metric was looked for.

Firstly, the best distance metric was found. The tricky thing about it was that cluster labels and actual labels may be different, so there is a need for a specific metric that handles this well. For this, ten synthetic datasets were made that represent different distributions of clusters. Ten metrics were chosen to evaluate those datasets. Metrics and results can be seen in Table 3. The best of the given results was the Chi-Square test. It gave good results on *df_real_01* dataset, which is closest to the real case.

Table 3. Evaluating different metrics on synthetic datasets.

	dataset	real_score	Adjusted Rand Index (ARI)	Adjusted Mutual Information (AMI)	Fowlkes-Mallows Index (FMI)	Balance Accuracy (BA)	Accuracy score (ACC)	Completeness (COM)	Calculate Homogenity (HOM)	V-measure (VMES)	Chi-root (CHI2)	Silhouette (SIL)
0	df_perfect	1.00	0.999992	0.998088	1.000000	0.500000	0.982059	1.000000	0.992387	0.998468	0.750000	0.998290
1	df_good_c5	0.90	1.000000	0.999633	1.000000	0.500000	0.998876	1.000000	0.998534	0.999706	1.000000	0.999972
2	df_good_c3	0.80	0.999159	0.955949	0.999967	0.166667	0.960823	1.000000	0.838120	0.962807	0.750000	0.997229
3	df_good_c2	0.70	0.508812	0.513532	0.990115	0.000000	0.951310	1.000000	0.013766	0.065236	0.250000	0.992718
4	df_real_01	0.87	0.611295	0.598009	0.990437	0.566196	0.958223	0.424798	0.128104	0.290320	0.849237	0.002390
5	df_real_1	0.83	0.564878	0.563133	0.981576	0.564696	0.949660	0.125886	0.127980	0.126299	0.849236	0.524519
6	df_real_10	0.80	0.512559	0.521157	0.897890	0.549645	0.864005	0.025550	0.128021	0.030420	0.849173	0.558804
7	df_real_noisy	0.10	0.567588	0.554727	0.904391	-0.016667	0.856179	0.065521	0.334576	0.078078	0.020623	0.896194
8	df_bad_v1	0.00	0.500000	0.500000	0.990026	-0.166667	0.000000	1.000000	0.000000	0.000000	0.000000	NaN
9	df_bad_v2	0.00	0.500366	0.500226	0.896581	-0.016667	0.856657	0.000354	0.001718	0.000421	0.000112	0.517021

Initially, cosine distance was used to measure the distance between two points, as in the initial TAD-FM paper [11]. However, due to its slow implementation in scikit-learn [3], Euclidean distance was considered. To evaluate if it is suitable, it was compared with cosine distance in a grid search; they gave similar patterns as can be seen in Figure 13, so Euclidean was chosen.

[3] https://scikit-learn.org/stable/

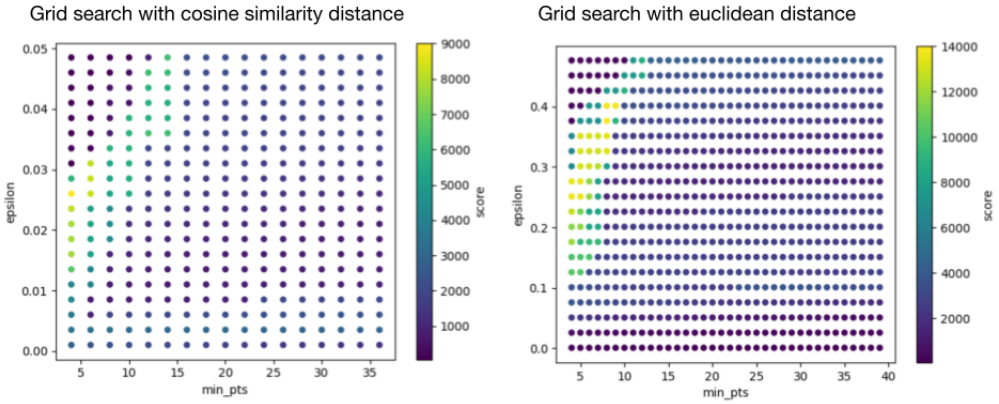

Figure 13. Comparison of cosine similarity with Euclidean metric grid search.

When looking into the hot spot's cluster distribution, the results were good but needed to improve. There were a lot of different small clusters, but it would have been hard to say if a small cluster was normal or abnormal. However, when the number of *MinPts* was raised, the results got better. Most of the values were in one big cluster and others that were marked as outliers mainly were broken or affected cells. So, it was chosen as the best parameter for DBSCAN.

4 Results

This Section will be about autoencoder layer learning, DBSCAN learning and the final results of the training and test set. Information about hardware and software can be seen in Appendix II.

4.1 Preprocessing Results

The steps done in the preprocessing helped to reduce the input for the autoencoder. In the initial paper [11] each trajectory was segmented and for each segment was calculated summarizing metrics. This means the number of summarizing metrics depends on how many events there is. In the current thesis the size of the autoencoder input depends only how many cells there is, what is considerably less than the number of events or even trajectories. It is this way because all the trajectories that are relevant for a given cell, their metrics are merged into one.

4.2 Autoencoders Results

The TAD-FM autoencoder layer was trained on two weeks of data. Preprocessing and training the neural network was done on two different computers. One was a laptop, and the other was a powerful remote server.

Preprocessing was done on the more powerful server. Data and preprocessing were split into three to four-day patches because there were many entries in the mobile events dataset, and the server halted with a longer time window. In total, it took 16h 39m 54s to calculate 28 metrics for each day for each cell.

Training the autoencoder went rather quickly. For two weeks of data, which was a couple of millions of rows, on a laptop, it took 133 seconds. Both autoencoders' loss graphs can be seen in Figure 14. From the graphs, it can be seen that the model is learning quite fast when compared to the original paper [11].

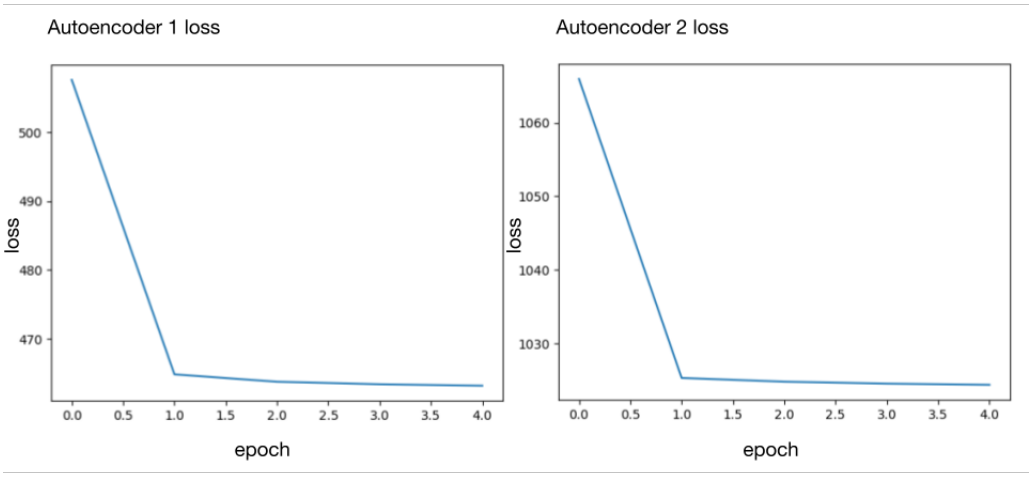

Figure 14. A custom loss function has been used for the first and second autoencoders' loss graphs.

4.3 Clustering Results

DBSCAN was already trained with one day of data that was under observation. At first, it was trained on a modified train dataset and later at a modified test dataset. Both datasets were run through the already trained autoencoder layer.

Using the original distance metric, cosine similarity, caused problems. Model training took a long time, and sometimes, it even halted. This problem was caused by the fact that scikit-learn does not support using tree-based algorithms with cosine similarity. This raised DBSCAN computational and memory complexity. Then, as described earlier and can be seen in Figure 13, cosine similarity was compared to Euclidean distance. Euclidean distance was chosen because scikit-learn supports using it in the tree-based algorithms, and the results are similar to cosine similarity.

In Table 4 and Table 6 are the results of the tests. Outliers summary of those tables can be seen in Table 5 and Table 7. In the results were three distinct cell types, marked in column *Type*. Here are listed their labels and meanings:

- "Norm" - cells which did not have any synthetically anomalous cells nearby,
- "Broken" - cells whose locations were synthetically changed in three different

ways,

- "Affected" - cells which had "broken" cells in at least one shared five-point trajectory.

The column *broken_type* with *type* gives information on how the cell or its neighbour is synthetically affected. Those types were chosen based on how humans can make an error when inserting location values. Here are listed those errors and their labels:

- "1" - two cell's locations are switched,
- "2" - latitude and longitude are switched,
- "3" - one of the two, latitude or longitude, location is random.

Training DBSCAN on training data took 741 seconds. It clustered data into two clusters plus outliers, as seen in Table 4. As expected, most of the cells are in the main cluster "0". Then, a small amount is marked as cluster "1" and noise. Chi-Square result was 48737.31.

Table 4. Train dataset clustering.

Type	**broken_type**	**labels**	**Precent by type**
Norm	0	0	99.72
		1	0.28
Affected	1	0	23.52
	2	-1	0.48
		0	57.11
	3	-1	0.13
		0	18.75
Broken	1	0	24.00
		1	4.00
	2	-1	40.00
	3	-1	16.00
		0	16.00

In Table 5, it could be seen that the model found over half of the broken cells. Also, there is only a little noise from other cells. None of the normal cells are an outlier, and a small percentage of affected cells are marked as outliers.

Table 5. Train dataset outliers percentage per type.

Type	Percent
Norm	0
Affected	0.61
Broken	56.00

Training DBSCAN on test data took 270 seconds. The results were also clustered into two clusters plus outliers, as seen in Table 6. The cluster distribution is somewhat similar. One difference is that there is a small amount of outliers in the type "Norm" cells. Also, cells with the type "Affected" have more outliers. Chi-Square result was 24972.21.

Table 6. Test dataset clustering

Type	broken_type	labels	Precentage by type
Norm	0	0	99.03
		1	0.96
		-1	0.01
Affected	1	0	40.44
	2	0	35.60
		-1	3.48
	3	0	19.00
		-1	1.49
Broken	1	0	27.78
		-1	11.11
	2	-1	22.22
	3	0	5.56
		-1	33.33

Table 7 is similar to the train results. It could be seen that the model was able to find over half of the broken cells. Also, there is only a little noise from other cells. A small percentage of the normal cells are an outlier, and a more significant percentage of affected cells are marked as outliers.

Table 7. Test dataset outliers percentage per type.

Type	Percent
Norm	0.01
Affected	4.97
Broken	66.66

5 Discussion

This Section will discuss the results from Section 4. A good thing about the model's performance is the rooms with improvements. Also how current implementation is usable and what needs to be done to make it perform even better. Lastly are brought out possible ways to improve the model and the input data.

5.1 Model performance

There were good things about results. The model found over half of the broken cells. A lot of processing was done in the preprocessing part. This reduces autoencoders' training time a lot. Also, there was not much noise, which could affect finding broken cells. It is even good when some affected cells are present. They help to find the broken cell's original location.

There is also room for improvements. It would have been better if all the "broken" cells were found. With current setup, when changing the DBSCAN parameters to achieve this, it would have come with a cost of more noise from normal cells, making it harder to distinct affected and broken cells from noisy normal cells.

5.2 Usability

With a couple of improvements, the model could be usable in business to find anomalistic cells. The code needs to be integrated with other business pipeline. This way a model's preprocessing and autoencoder part could be done weekly or over two weeks, depending on the needs. Training DBSCAN could be done daily. Then the outcome would be an error raised with an info which cells are outliers and needs to be checked. Finding problematic cells quickly gives the possibility to fix issues fast therefore affecting less end users' analysis.

There is also room to make the implementation better by reducing computation needed. Model training process could be done fewer times. Maybe monthly or even quarterly. The costly part currently is preprocessing. It is worth looking into reducing this part's computation time because currently, it takes almost 17 hours, but there were other business-critical processes running on the same server.

5.3 Possible Future Improvements

There are ways to make enhancements to both the model and the overall process. Mainly, those improvements are ways to evaluate mode, better preprocessing, and improve neural network or clustering.

5.3.1 Model

One way is to improve preprocessing. When looking into data, it takes a full day at a time. This window could be reduced to avoid long jumps because the phone was turned off, but the person moved. These jumps are causing a hard time for clustering because it will make the distance metrics longer, but usually, the anomalitic cells also have a large value in the distance metrics. Making adjustments could help make a better distinction between the two.

Another way to make preprocessing better is to reduce the number of features. Currently, there are 28 metrics for each cell. This is a lot, but when removing less necessary ones, all the other processes down the pipeline take less time.

Autoencoders could also be made lighter. It would be worth reducing the number of neurons in the hidden layer. This reduces the computation needed for training and predicting, making the model cheaper to run and is also greener. Reducing the number of neurons forces the model to generalise more therefore making it more robust.

Clustering could be enhanced in a couple of ways. One could be saving only the core points or the cluster areas. This could reduce computation power because new points could be measured against existing core points. Finding new clusters is unnecessary every time because the behaviour of the cell's should not change a lot.

Currently, the scikit-learn's DBSCAN implementation is slow, especially with cosine distance. So, the scikit-learn could be replaced by something else. In the current case, it would be good to do it in Apache Spark. It would be good because preprocessing is already done using Pyspark, and it could better handle a large number of data points better. A new implementation in Spark could help try out cosine similarity as in the initial TAD-FM paper.

The TAD-FM gave promising results but, it would be good to look into other methods. They could be used as comparison models to evaluate if the results are good.

It is also worth to look over the autoencoders' training process. It could be done by

trying out other loss functions and optimizers. This could help to train the model better making the loss graphs a little better looking than it is currently seen in Figure 14.

5.3.2 Data

Clustering results could also give a better understanding of the data. It would be worth looking into what type of data is in different clusters. This could also give some new insights about the data and maybe even some new hypotheses which could be worth researching.

Better input data is always a good way to improve the results. Currently the anomalous cells were generated synthetically, but it would be better to find more known real-life errors. Also, enhancing the synthetic anomalous data generation could be developed further by introducing new possible error types. Better anomalous data points give a possibility to train and evaluate models better.

Conclusion

This thesis gives an overview to a reader simplistic overview of how a cellular network works. It is described, that wrong cell location entry in a database could cause many problems, for example, how it affects the trajectories of mobile phones. Therefore, it affects end users, like emergency services.

Then a reader could learn different ways in which anomalistic trajectories could be found. A little summary of a variety of methods from different articles was done in three categories: statistical learning, clustering and deep learning methods. One of them was also TAD-FM [11], which had parts of clustering and deep learning methods.

The thesis described how a TAD-FM was used to solve one real-world problem: finding the cell with the wrong location data. Compared to the original article about TAD-FM [11], there were updates to preprocessing and clustering. Changes in preprocessing helped to make the joined mobile events and cell plan datasets usable for TAD-FM and reduce the computation power for autoencoders. Also, a distance metric was changed in the clustering part from cosine similarity to Euclidean distance to reduce computation and memory complexity for DBSCAN.

A considerable amount of real-life data was used to train the autoencoders, and a half-synthetic dataset was created to train and test the whole model. The results were promising using this improved model with the given dataset. The model was able to find over half of the "broken" cells without a lot of noise. With a couple of extra steps, the finished model is usable in business to solve the issue.

In the future there are a couple of ways to make improvements for this thesis method. In preprocessing, it could reduce the number of features and trajectory time window. In the neural network layer, it could reduce the number of neurons and enhance the training process. Also, improving the clustering could be done by separating training and predicting processes. All this could reduce the computation burden and enhance the performance. It is also worth looking into other methods to find anomalistic trajectories.

Acknowledgments

This research was done in Reach-U in the University of Tartu Industrial Master's Programme in IT. Reach-U provided access to the data and provided necessary working equipment and environment.

I would like to thank my supervisor, Amnir Hadachi, who helped me with working on and writing my thesis. Also, a lot of support came from Reach-U and its ETL team. So big thanks to Joosep, Robert, Tanel and also Elis and Teet.

References

[1] International Telecommunication Union. *Facts and Figures 2022*. 2024. URL: https://www.itu.int/itu-d/reports/statistics/2022/11/24/ff22-mobile-phone-ownership/ (visited on 04/03/2024).

[2] Petroc Taylor. *Average monthly usage of mobile data per smartphone in 2022 and 2028*, by region(in gigabytes)*. 2024. URL: https://www.statista.com/statistics/1100854/global-mobile-data-usage-2024/ (visited on 04/15/2024).

[3] Hendrik Hiir et al. "Impact of Natural and Social Events on Mobile Call Data Records – An Estonian Case Study". In: Jan. 2020, pp. 415–426. ISBN: 978-3-030-36682-7. DOI: 10.1007/978-3-030-36683-4_34.

[4] Kevin S. Kung et al. "Exploring Universal Patterns in Human Home-Work Commuting from Mobile Phone Data". In: *PLOS ONE* 9.6 (June 2014), pp. 1–15. DOI: 10.1371/journal.pone.0096180. URL: https://doi.org/10.1371/journal.pone.0096180.

[5] Guangyuan Zhang et al. "A Method for the Estimation of Finely-Grained Temporal Spatial Human Population Density Distributions Based on Cell Phone Call Detail Records". In: *Remote Sensing* 12.16 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12162572. URL: https://www.mdpi.com/2072-4292/12/16/2572.

[6] Kyra H. Grantz et al. "The use of mobile phone data to inform analysis of COVID-19 pandemic epidemiology". In: *Nature Communications* 11 (2020). URL: https://api.semanticscholar.org/CorpusID:222162112.

[7] Nishant Kishore et al. "Flying, phones and flu: Anonymized call records suggest that Keflavik International Airport introduced pandemic H1N1 into Iceland in 2009". In: *Influenza and Other Respiratory Viruses* 14 (Nov. 2019). DOI: 10.1111/irv.12690.

[8] Patty Kostkova et al. "Data and Digital Solutions to Support Surveillance Strategies in the Context of the COVID-19 Pandemic". In: *Frontiers in Digital Health* 3 (2021). ISSN: 2673-253X. DOI: 10.3389/fdgth.2021.707902. URL: https://www.frontiersin.org/journals/digital-health/articles/10.3389/fdgth.2021.707902.

[9] Filippo Maria Bianchi et al. "Identifying user habits through data mining on call data records". In: *Engineering Applications of Artificial Intelligence* 54 (2016), pp. 49–61. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2016.05.007. URL: https://www.sciencedirect.com/science/article/pii/S0952197616300975.

[10] Hongjian Wang et al. “A Simple Baseline for Travel Time Estimation using Large-scale Trip Data”. In: *ACM Trans. Intell. Syst. Technol.* 10.2 (Jan. 2019). ISSN: 2157-6904. DOI: 10.1145/3293317. URL: https://doi.org/10.1145/3293317.

[11] Zhongqiu Wang et al. “Unsupervised learning trajectory anomaly detection algorithm based on deep representation”. In: *International Journal of Distributed Sensor Networks* 16.12 (2020), p. 1550147720971504. DOI: 10.1177/1550147720971504. eprint: https://doi.org/10.1177/1550147720971504. URL: https://doi.org/10.1177/1550147720971504.

[12] Vyacheslav Shatskiy. *black and white electric post under blue sky during daytime*. 2021. URL: https://unsplash.com/photos/black-and-white-electric-post-under-blue-sky-during-daytime-31JqyCVndUM (visited on 04/17/2024).

[13] Andrew Ng et al. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[14] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[15] Erich Schubert et al. “DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN”. In: *ACM Trans. Database Syst.* 42.3 (July 2017). ISSN: 0362-5915. DOI: 10.1145/3068335. URL: https://doi.org/10.1145/3068335.

[16] Olov Rosén and Alexander Medvedev. “An on-line algorithm for anomaly detection in trajectory data”. In: *2012 American Control Conference (ACC)*. 2012, pp. 1117–1122. DOI: 10.1109/ACC.2012.6315346.

[17] Behrouz Haji Soleimani et al. “Anomaly detection in maritime data based on geometrical analysis of trajectories”. In: *2015 18th International Conference on Information Fusion (Fusion)*. 2015, pp. 1100–1105.

[18] Rikard Laxhammar. “Anomaly detection in trajectory data for surveillance applications”. PhD thesis. Örebro universitet, 2011. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:oru:diva-17235.

[19] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. “Trajectory-Based Anomalous Event Detection”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 18.11 (2008), pp. 1544–1554. DOI: 10.1109/TCSVT.2008.2005599.

[20] Bin Cheng et al. “STL: Online Detection of Taxi Trajectory Anomaly Based on Spatial-Temporal Laws”. In: *Database Systems for Advanced Applications*. Ed. by Guoliang Li et al. Cham: Springer International Publishing, 2019, pp. 764–779. ISBN: 978-3-030-18579-4.

[21] Su Yang and Weihua Liu. “Anomaly Detection on Collective Moving Patterns: A Hidden Markov Model Based Solution”. In: *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. 2011, pp. 291–296. DOI: 10.1109/iThings/CPSCom.2011.25.

[22] Zhen Wang and Sihai Zhang. “CDR Based Temporal-Spatial Analysis of Anomalous Mobile Users”. In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2016, pp. 710–714. DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.126.

[23] Xumin Cai et al. “Local Outlier Detection for Multi-type Spatio-temporal Trajectories”. In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 4509–4518. DOI: 10.1109/BigData50022.2020.9377801.

[24] Ivens Portugal, Paulo Alencar, and Donald Cowan. “A Framework for Spatial-Temporal Trajectory Cluster Analysis Based on Dynamic Relationships”. In: *IEEE Access* 8 (2020), pp. 169775–169793. DOI: 10.1109/ACCESS.2020.3023376.

[25] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. “Density-Based Clustering Based on Hierarchical Density Estimates”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 2013. URL: https://api.semanticscholar.org/CorpusID:32384865.

[26] Yunyao Cheng et al. “Spatial-Temporal Recurrent Neural Network for Anomalous Trajectories Detection”. In: *Advanced Data Mining and Applications*. Ed. by Jianxin Li et al. Cham: Springer International Publishing, 2019, pp. 565–578. ISBN: 978-3-030-35231-8.

[27] Wenhao Yu and Qinghong Huang. “A deep encoder-decoder network for anomaly detection in driving trajectory behavior under spatio-temporal context”. In: *International Journal of Applied Earth Observation and Geoinformation* 115 (2022), p. 103115. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2022.103115. URL: https://www.sciencedirect.com/science/article/pii/S156984322200303X.

[28] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[29] Thomas Bäck and Hans-Paul Schwefel. “An overview of evolutionary algorithms for parameter optimization”. In: *Evolutionary computation* 1.1 (1993), pp. 1–23.

[30] Daniel Tucker. *evolutionary-algorithm 0.0.2*. 2020. URL: https://pypi.org/project/evolutionary-algorithm/ (visited on 05/07/2024).

Appendices

I Detailed Preprocessing

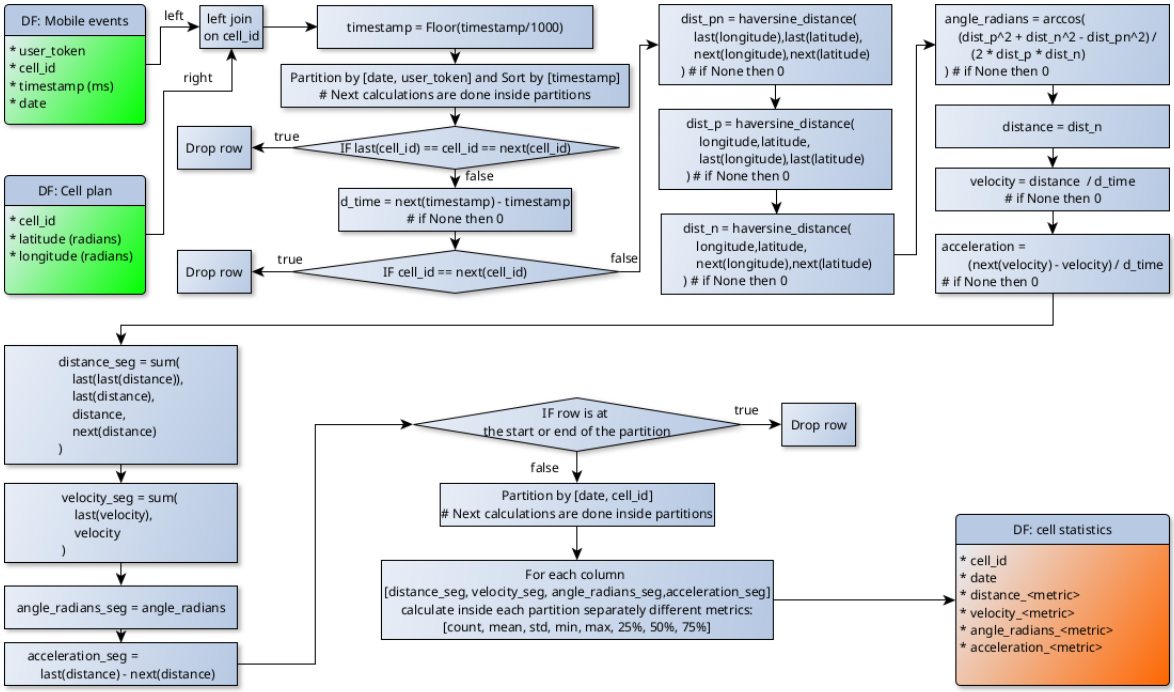

Figure 15. Detailed overview of the preprocessing.

II Used Soft- and Hardware

Calculating metrics and training DBSCAN was done on a more powerful server.

- OS: Red Hat Enterprise Linux Server release 7.9 (Maipo)
- RAM: 503GB
- CPU: Intel (R) Xeon (R) CPU E5-2667 v4 @ 3.20GHz

Most important software and versions used in a server:

- python 3.9, pip 24.0
- jupyterlab==4.0.12
- pandas==2.1.4
- numpy==1.24.4
- pyspark==3.5.0
- scikit-learn==1.4.1.post1
- scipy==1.9.1

For development and training the autoencoders was used Lenovo ThinkPad P14s Gen 4 21K5000BMX.

- OS: Ubuntu 23.10
- RAM: 27GiB
- CPU: AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics

Most important software and versions used in a laptop:

- python 3.11, pip 23.0.1
- notebook==7.0.6

- pandas==2.1.2
- numpy==1.26.1
- pyspark==3.5.0
- scikit-learn==1.3.2
- torch==2.1.0
- evolutionary-algorithm==0.0.2
- scipy==1.11.3

III Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Tiit Vaino**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Anomaly Detection in CDR-Based Trajectories of the Mobile Cellular Network.

   supervised by Amnir Hadachi.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Tiit Vaino
15/05/2024