Development and performance of a transport mode classification algorithm for smart surveys

Jonas Klingwort, Yvonne Gootzen, Jurgens Fourie

31.03.2025

Contents

1 Introduction

An integral component of smart time-use, travel, and mobility surveys is the ability to predict respondents' modes of transportation, thereby minimizing the necessity for manual data labeling and reducing the response burden. This report documents the development of a transport mode prediction algorithm specifically designed for integration with smart surveys. The algorithm is based on a decision tree, using smartphone GPS data and infrastructure information from OpenStreetMap (OSM). The development of the algorithm was based on data collected by Statistics Netherlands (Schouten et al. 2024). The algorithm was also evaluated on open geo-data that is publicly available in the SSI Git-repository (https://github.com/essnet-ssi/geoservice-ssi) and was collected within the scope of the SSI project. This document provides a comprehensive overview of the algorithm's development, describing the development procedures, the datasets utilized, the underlying methodology, and the resulting outcomes.

2 Background

Transport mode prediction currently lacks a universally established algorithm. Existing methods predominantly rely on manual rule-based approaches, decision trees, or machine-learning techniques. Within this project's scope, an extensive review of existing methodologies and algorithms was conducted by Fourie (2025). For developing a transport mode classification algorithm within the SSI project, it was decided to base it on a decision tree due to its simplicity and interpretability. We briefly discuss their advantages and disadvantages and compare manual rule-based approaches, decision trees, and machine-learning models. Some of the benefits of decision-tree models are listed below:

1. Interpretability: Decision trees provide a transparent, easy-to-understand decision-making process, making them ideal for explaining predictions.
2. Non-linearity: They can model complex relationships between input features without requiring linear assumptions.
3. Feature importance: Decision trees naturally rank features based on their importance, helping understand key factors affecting transport mode choices.
4. Categorical & numerical data: They can process different data types (e.g., numeric, timestamps, categorical travel modes) without complex preprocessing.
5. Computational efficiency: Training and prediction are relatively fast, making them suitable for real-time transport mode prediction in smart surveys.
6. Missing data: Decision trees can handle missing values better than some machine learning models using surrogate splits.

Some of the disadvantages of decision-tree models are listed below:

1. Overfitting: Decision trees can overfit the training data, leading to poor generalization unless pruning techniques are applied.

2. Sensitivity to noisy data: Small variations in input data can lead to different splits, making the model unstable.

3. Limited expressiveness: Decision trees can handle complex patterns but may struggle with highly complex relationships between features compared to deep learning models.

4. Bias in splitting criteria: Splitting criteria like the Gini index or Information Gain tend to favor features with more levels, which might lead to biased predictions.

We also give a brief comparison of decision trees with rule-based approaches and with machine-learning approaches. Decision trees are more flexible and scalable than rule-based approaches but may overfit the data. Rule-based methods are static and rely heavily on expert knowledge, which may not generalize well. Decision trees can be trained automatically while rule-based approaches are handcrafted. This comparison is summarized in Table 1.

Theme	**Decision Trees**	**Manual Rule-based Approaches**
Flexibility	Adapts to patterns in data automatically	Requires manually defined rules
Interpretability	Easy to understand	Easy to understand
Scalability	Scales well with data size	Becomes complex with increasing rules
Handling New Data	Can retrain to adjust	Needs manual updates
Accuracy	Higher with enough data	Limited by predefined rules

Table 1: Comparison of decision trees and rule-based approaches

While decision trees offer simplicity and interpretability, they may not be as accurate as ensemble methods (like Random Forests) or deep learning models. Table 2 gives a brief summary:

Theme	**Decision Trees**	**Random Forest**	**Neural Networks**
Interpretability	High	Medium (ensemble of trees)	Low (black box model)
Accuracy	Moderate	High	Very High
Computational Cost	Low	Medium	High
Handling Overfitting	Pruning needed	Less prone (ensemble effect)	Requires regularization
Training Speed	Fast	Slower than single tree	Slowest

Table 2: Comparison of decision trees with machine learning models

Alongside this project, and as an integral component of its development, a rule-based algorithm for transport mode classification was also developed by Fourie (2025). The results obtained during the development of this algorithm contributed to the advancement of this project. The algorithm with results is included in the Appendix D. The findings derived from the work by Fourie (2025) are expected to be published soon by Fourie et al. (expected 2025). This simple rule-based algorithm performed reasonably well but had the drawback of resulting in multiple classifications. The developed rule-based approach is a non-nested system that evaluates conditions independently and selects the best match without a strict hierarchy. This fact can lead to overlapping or conflicting conditions requiring priority handling. No priority handling is needed using a decision tree because each track will be assigned one predicted transport mode.

3 Data

Two datasets were used to develop and evaluate the transport mode prediction algorithm. First, data from a large-scale field study conducted by Statistics Netherlands was used to develop and test the algorithm's internal validity. This dataset is not publicly available. Second, data was collected within the SSI project to create an open, publicly available geo-dataset with error-free labels. This dataset was used to evaluate the generalizability of the algorithm and establish its external validity.

3.1 Development data

The dataset is based on a Dutch general population sample collected from 2022 to 2023. Data from 255 participants were used for the development. The dataset contains 4,298 tracks and a total of about 20 million observations. An observation consists of a timestamp and geo-location (longitude and latitude). We refer to Schouten et al. 2024 for general details about the dataset.

3.1.1 Data processing

The dataset underwent preprocessing steps to ensure quality and consistency. Gootzen et al. expected 2025 describes the initial and general data cleaning procedures and informs on the data quality. For the specific development of the transport mode classification algorithm, tracks exceeding 10 hours were excluded. Second, tracks containing fewer than ten GPS observations were removed. Third, labels for similar transport modes were grouped: 'car (driver)' and 'car (passenger)' were merged into a single transport mode. The categories 'bike' and 'e-bike' were also merged into one transport mode. After preprocessing, the average number of GPS observations per track was 843, although the median was notably lower at 402, indicating a skewed distribution. Similarly, the average track duration was 53 minutes, but the median was 12 minutes, reflecting the skewness. Regarding track length, the mean was 15 km, while the median was considerably shorter at 2.8 km, again indicating a skewed distribution in the data.

3.1.2 Transport modes

The target variable of the classification task is the transport mode used during a track. The distribution of track labels across transport modes in the entire dataset is presented in Table 3. The labels indicate that the developed algorithm will only classify a single mode. Multi-modal tracks cannot be classified. This fact is not a shortcoming caused by the algorithm, but the labels to develop/train the algorithm do not contain multi-modal tracks. The target variable also contains the label 'Other'. This label is expected to reduce the overall quality of the algorithm as it is not a defined mode of transport and can contain a wide variation of transport modes. This label was excluded to only classify well-defined transport modes in the study by Fourie (2025).

Mode	Count	Percentage
Car	1,892	44.02
Walk	1,002	23.31
Bike	946	22.01
Train	161	3.75
Other	111	2.58
Bus	96	2.23
Metro	58	1.35
Tram	32	0.74
Total	4,298	100

Table 3: Distribution of transport modes in development data. Rows ordered by count.

3.1.3 Train and test splits

The dataset was partitioned at the user level, ensuring that each user was assigned exclusively to the training or testing set, but not both. This approach was chosen to evaluate the model's ability to generalize to entirely unseen users, providing a strict and realistic assessment of generalization in scenarios where new users are encountered in future surveys. By separating users in this manner, the risk of overfitting is mitigated, as the model is prevented from learning user-specific patterns from the training set that could influence predictions in the test set. However, this approach introduces specific challenges. The number of users in the dataset is limited, and some users contribute disproportionately, with a large number of tracks attributed to a single individual. As a result, the train-test splits may become imbalanced across labels, potentially affecting the robustness and generalizability of the algorithm. Therefore, it was decided to split the dataset by partitioning users into separate subsets for training (70%) and testing (30%), ensuring that no user appeared in both sets. Stratification was applied based on each user's dominant mode of transport to maintain a balanced representation of transport modes. The public transport modes bus, metro, and tram were grouped for the train and test splits (they were used as individual classes for the remainder of the development). This practical solution prevented the case that tram was once only the most prominent mode, and therefore, no split could have been applied because the stratification would require at least two occurrences of a transport mode. This approach preserved the variation and distribution of transport modes across both subsets, ensuring that the test set accurately reflected the training data's characteristics while preventing user overlap between the two sets. Tables 4 and 5 show the train and test splits.

3.2 Open geo-data

This dataset was reserved exclusively for testing the developed algorithm, with no portion used during the development or training phases, ensuring an unbiased evaluation of the algorithm's generalization capabilities. The dataset was collected in the

Mode	Count	Percentage
Car	1,333	44.43
Walk	684	22.80
Bike	643	21.43
Other	103	3.43
Train	99	3.30
Bus	75	2.50
Metro	45	1.50
Tram	18	0.60
Total	3,000	100

Table 4: Train set. Rows ordered by count.

Mode	Count	Percentage
Car	559	43.07
Walk	318	24.50
Bike	303	23.34
Train	62	4.78
Bus	21	1.62
Tram	14	1.08
Metro	13	1.00
Other	8	0.62
Total	1,298	100

Table 5: Test set. Rows ordered by count.

summer of 2024 using the most recent version of the CBS smartphone app available at that time. This app version employed a revised sensor configuration compared to the app used to collect the development data. The updated configuration reduced the number of sensors used in the smartphone to collect GPS but prioritized collecting more detailed data from a single sensor type. The data was collected to obtain data with high-quality labels without errors for the transport mode. This data was collected by a small group of CBS staff and staff from the University of Utrecht. Furthermore, the data contains tracks within the Netherlands and Germany. Accordingly, this test set will inform how well the algorithm generalizes to a different app version/sensor configuration and data collected in a different country. Data from 5 users with 137 tracks are available. The transport mode distribution is shown in table 6.

Mode	Count	Percentage
Walk	78	0.61
Tram	27	0.21
Bike	10	0.08
Train	9	0.07
Bus	5	0.04
Metro	5	0.04
Ferry	3	0.02
Total	137	100

Table 6: Transport modes in open geo-data. Rows ordered by count.

Note that the decision tree was not trained on data containing the 'ferry' label. Thus, the algorithm will fail to predict this label. However, it was collected to evaluate the algorithm's decision for this label. Lastly, the most prominent mode in the development data, 'car', is not included. This dataset is publicly available in the SSI Git repository (https://github.com/essnet-ssi/geoservice-ssi).

4 Methods

The methods section contains the construction of GPS features (Section 4.1), the construction of OSM features (Section 4.2), the pre-processing of GPS and OSM feature (Section 4.3), development of the decision tree (Section 4.4), and the transport mode prediction algorithm (Section 4.4).

4.1 Feature construction GPS

This section describes the required GPS features for the algorithm. In particular, features from five themes were created and evaluated: speed (speed, acceleration, jerk, snap), GPS (accuracy, frequency), direction (bearing, altitude), trip (length, duration), and time (weekday, weekend indicator). For most features, several variants were created based on different statistics. A list of all GPS features is given in Appendix A. Even more GPS features were evaluated by Fourie et al. (expected 2025). However, as they have been found not to be relevant, they were not implemented here. The required Python code can be found in the accompanying script **gps_feature.py**. A list with a short description of all Python scripts can be found in Appendix C.

4.2 Feature construction OSM

This section describes the required OSM features for the algorithm. A list of all used OSM features is given in Appendix B. The features are based on publicly available OpenStreetMap (OSM) data obtained from the official Geofabrik download portal (https://download.geofabrik.de/). A documentation of all OSM infrastructure contained in the database can be found at: https://wiki.openstreetmap.org/wiki/Map_features. OSM data complements the GPS features described in Section 4.1 by providing details on infrastructure such as road networks, transit routes, and stations. Integrating this data improves usually the quality of the transport mode classifications (Fourie 2025; Gong et al. 2012; Sadeghian et al. 2022; Smeets et al. 2019). Fourie (2025) systematically studied which OSM features have the most considerable potential to improve the transport mode classifications. The main findings by Fourie (2025) were, a) that OSM did not help to improve the classification quality for the transport modes walk, bike, and car. Based on these findings, it was decided not to create features using OSM-specific information for these three transport modes, and b) that although OSM provides a variety of data on transportation and travel infrastructure – including features such as roundabouts, traffic junctions, stop signs, speed cameras, and street lamps, these did not help improve the classification performance. Another reason to limit the number of OSM features is computational efficiency. Accordingly, in the development of the algorithm, only OSM features about bus, metro, train, and tram stops and/or routes were created. The required Python code can be found in the accompanying script **osm_features.py**. Even more OSM features were evaluated by Fourie et al. (expected 2025). However, as they have found not to be relevant, they were not implemented here.

Track buffering

Buffering a GPS track when calculating features using OSM data is beneficial because it helps include relevant spatial context around the track, improving feature extraction and accuracy. In the following, we explain why this step is helpful. First, it accounts for GPS inaccuracies and noise. GPS tracks often have errors due to signal loss, multipath effects, or device inaccuracies. A buffer helps compensate for small deviations and ensures relevant OSM features are included even if the track is slightly misaligned. Second, it captures nearby infrastructure and context. Many transport mode features depend on proximity to roads, paths, and transit stops. A buffer ensures that all relevant OSM data is considered within a reasonable distance of the track. This is especially important in urban environments where GPS can jump between nearby roads. Third, it enables more robust feature engineering. By including OSM features within the buffer, one can calculate more informative features such as, for example, pathway availability (e.g., bike lanes, sidewalks, pedestrian zones) or transit accessibility (e.g., nearest bus/tram stops). Fourth, it handles transport mode variability. A narrow track-only approach may miss important context (e.g., a train passenger may be slightly off the designated rail network). To conclude, buffering the GPS track will likely improve spatial accuracy, feature information, and mode classification robustness when integrating OSM data.

The buffering process takes the GPS coordinates representing the track and generates a buffer zone around it. This buffer is defined by a specified radius or distance, which determines how far the area extends from the track's centerline. For instance, a buffer with a radius of 25 meters would create a region 25 meters wide on either side of the track. This is the radius so that the diameter will be 50m. An example of this procedure is shown in Figure 1.

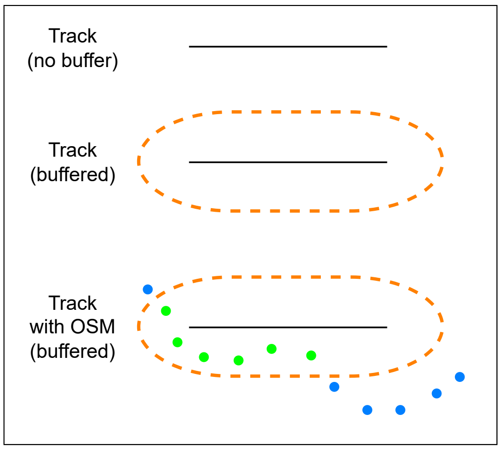

Figure 1: Simplified example of track buffering: a single track (black solid line), a buffered track (black solid line with surrounding orange dashed line), and a buffered track with mapped OSM infrastructure (black solid line with surrounding orange dashed line and mapped OSM infrastructure. Green points in the buffer are considered for feature construction, blue points outside the buffer are not.)

Once the buffer is constructed, spatial operations are performed to identify which OSM coordinates or features lie within the buffered area. This is achieved using spatial indexing and intersection techniques, which compare the locations of OSM features to the buffer's boundaries. The accompanying Python script **osm_features.py** contains the code to apply the buffering. Representing the track as a linestring object instead of considering the individual measurements enormously increased computational efficiency. Points-of-interest that fall within or intersect the buffer are retained for further analysis. For the OSM count features, the total buffer was also used to normalize features for a fair comparison between shorter and longer tracks. In contrast to Fourie (2025), who used a 10 meter buffer, a buffer of 25 meters was used. It was tested whether different buffer sizes (10, 20, 50, 75, and 100 meters) were having an effect. No noticeable changes in results were observed (Fourie et al. expected 2025). The lack of an impact might be due to the fixed thresholds used in the rule-based algorithm.

4.3 Pre-processing of GPS and OSM features

Some GPS calculations did not result in reasonable numeric values. If the calculation of a feature resulted in an infinite value, the infinite value was replaced with twice the maximum value (inf → 2 * max). A negative infinity value was set to zero (-inf → 0). Missing values remained unchanged since a decision tree can handle missing data. String variables were factorized for the decision tree. For the OSM features, some variables contained missing values. This occurs when there is no OSM infrastructure in the buffer of a track. Here, the missing data was replaced with a zero count, reflecting this feature's actual absence.

4.4 Decision tree development

A decision tree is a supervised learning algorithm used for classification and regression tasks. It is a tree-like model where each internal node represents a decision based on a feature, each branch represents an outcome of that decision, and each leaf node represents a final prediction. In the following, we briefly explain the components of the tree. The tree starts with the root node, the topmost node of the tree. It represents the entire dataset and the first decision point based on a selected feature. The decision nodes are intermediate nodes that split data based on a condition. Each decision node applies a rule (e.g., speed > 30km/h?) and branches accordingly. The branches (or edges) represent possible outcomes of a decision. They connect nodes and direct the data down the tree. The leaf nodes (or terminal nodes) represent the final outcome/classification (e.g., 'Car' or 'Bike'). The process of dividing a node into two or more sub-nodes is based on feature conditions.

Optimizing hyperparameter space of decision tree

Grid search was done which is a hyperparameter tuning technique used to find the best combination of parameters that optimize the model's performance. It systematically searches through a predefined set of hyperparameters by testing all possible combinations and selecting the best one based on a scoring metric. The hyperparameter search was conducted using the following space:

$$\begin{aligned}
\text{Maximum depth:} \quad & d \in \{3, 5, 7\}, \\
\text{Minimum samples split:} \quad & m_{\text{split}} \in \{10, 25, 50\}, \\
\text{Minimum samples leaf:} \quad & m_{\text{leaf}} \in \{5, 10, 15, 20, 25\}, \\
\text{Criterion:} \quad & c \in \{\text{gini}, \text{entropy}\}, \\
\text{Maximum features:} \quad & f \in \{1, 3, 5, 7\}.
\end{aligned}$$

The output for the optimal decision tree and the transport mode prediction algorithm respectively is shown below. The required Python code for the algorithm can be found in the accompanying script **train_decision_tree.py**.
The file **decision_tree_ssi.pickle** contains the trained decision-tree model.

Transport mode prediction algorithm

```

node=0 is a split node with value=[anders: 0.034, auto: 0.444, bus: 0.025, fiets: 0.214, metro: 0.015, tram: 0.006, trein: 0.033, voet: 0.228]: go to node 1 if bus_route_mean_distance <= 1287.4715576171875 else to node 64.
    node=1 is a split node with value=[anders: 0.022, auto: 0.288, bus: 0.016, fiets: 0.297, metro: 0.005, tram: 0.007, trein: 0.004, voet: 0.36]: go to node 2 if speed_percentile_10 <= 4.045245885848999 else to node 39.
        node=2 is a split node with value=[anders: 0.029, auto: 0.255, bus: 0.013, fiets: 0.235, metro: 0.007, tram: 0.005, trein: 0.003, voet: 0.453]: go to node 3 if speed_iqr_value <= 5.45417857170105 else to node 16.
            node=3 is a split node with value=[anders: 0.014, auto: 0.062, bus: 0.003, fiets: 0.089, metro: 0.006, tram: 0.002, voet: 0.824]: go to node 4 if speed_percentile_80 <= 10.346889019012451 else to node 15.
                node=4 is a split node with value=[anders: 0.013, auto: 0.066, bus: 0.003, fiets: 0.048, metro: 0.007, tram: 0.002, voet: 0.862]: go to node 5 if jerk_percentile_85 <= 7493698.0 else to node 10.
                    node=5 is a split node with value=[anders: 0.022, auto: 0.231, fiets: 0.088, metro: 0.011, tram: 0.011, voet: 0.637]: go to node 6 if altitude_percentile_85 <= 2.445638060569763 else to node 9.
                        node=6 is a split node with value=[anders: 0.027, auto: 0.178, fiets: 0.055, metro: 0.014, voet: 0.726]: go to node 7 if altitude_percentile_85 <= 0.14158499240875244 else to node 8.
                            node=7 is a leaf node with values=[auto: 0.303, fiets: 0.121, metro: 0.03, voet: 0.545].
                            node=8 is a leaf node with values=[anders: 0.05, auto: 0.075, voet: 0.875].
                        node=9 is a leaf node with values=[auto: 0.444, fiets: 0.222, tram: 0.056, voet: 0.278].
                    node=10 is a split node with value=[anders: 0.012, auto: 0.037, bus: 0.004, fiets: 0.041, metro: 0.006, voet: 0.902]: go to node 11 if metro_route_min_distance <= 0.024851143825799227 else to node 14.
                        node=11 is a split node with value=[anders: 0.012, auto: 0.038, bus: 0.004, fiets: 0.042, voet: 0.903]: go to node 12 if speed_percentile_5 <= 0.2018592208623886 else to node 13.
                            node=12 is a leaf node with values=[anders: 0.028, auto: 0.099, bus: 0.014, fiets: 0.057, voet: 0.801].
                            node=13 is a leaf node with values=[anders: 0.006, auto: 0.014, fiets: 0.037, voet: 0.944].
                        node=14 is a leaf node with values=[metro: 0.143, voet: 0.857].
                node=15 is a leaf node with values=[anders: 0.029, fiets: 0.824, voet: 0.147].
            node=16 is a split node with value=[anders: 0.042, auto: 0.429, bus: 0.021, fiets: 0.367, metro: 0.008, tram: 0.008, trein: 0.006, voet: 0.118]: go to node 17 if speed_stddev <= 10.804237842559814 else to node 32.
                node=17 is a split node with value=[anders: 0.052, auto: 0.297, bus: 0.018, fiets: 0.487, metro: 0.008, tram: 0.006, trein: 0.004, voet: 0.128]: go to node 18 if bus_route_max_distance <= 144.82131958007812 else to node 25.
                    node=18 is a split node with value=[anders: 0.114, auto: 0.217, fiets: 0.408, metro: 0.005, tram: 0.005, trein: 0.005, voet: 0.245]: go to node 19 if accuracy_percentile_85 <= 14.302633285522461 else to node 22.
                        node=19 is a split node with value=[anders: 0.18, auto: 0.261, fiets: 0.486, voet: 0.072]: go to node 20 if proportion_10_30 <= 0.280659481883049 else to node 21.
                            node=20 is a leaf node with values=[anders: 0.179, auto: 0.464, fiets: 0.179, voet: 0.179].
                            node=21 is a leaf node with values=[anders: 0.181, auto: 0.193, fiets: 0.59, voet: 0.036].
                        node=22 is a split node with value=[anders: 0.014, auto: 0.151, fiets: 0.288, metro: 0.014, tram: 0.014, trein: 0.014, voet: 0.507]: go to node 23 if proportion_15_30 <= 0.2102891132235527 else to node 24.
                            node=23 is a leaf node with values=[anders: 0.018, auto: 0.123, fiets: 0.193, metro: 0.018, trein: 0.018, voet: 0.632].
                            node=24 is a leaf node with values=[auto: 0.25, fiets: 0.625, tram: 0.062, voet: 0.062].
                    node=25 is a split node with value=[anders: 0.016, auto: 0.344, bus: 0.028, fiets: 0.533, metro: 0.009, tram: 0.006, trein: 0.003, voet: 0.06]: go to node 26 if acc_kurt <= inf else to node 29.
                        node=26 is a split node with value=[anders: 0.029, auto: 0.076, fiets: 0.829, voet: 0.067]: go to node 27 if sd_time_diff_s <= 1.9156306385993958 else to node 28.
                            node=27 is a leaf node with values=[anders: 0.03, fiets: 0.925, voet: 0.045].
```

```
                                                        node=28 is a leaf node with values=[anders: 0.026, auto: 0.211, fiets: 0.658, voet: 0.105].
                                                node=29 is a split node with value=[anders: 0.009, auto: 0.476, bus: 0.042, fiets: 0.387, metro: 0.014, tram: 0.009, trein: 0.005, voet: 0.057]: go to node 30 if speed_iqr_value <= 20.525142669677734 else to node 31.
                                                        node=30 is a leaf node with values=[anders: 0.01, auto: 0.171, bus: 0.01, fiets: 0.733, metro: 0.01, tram: 0.01, voet: 0.057].
                                                        node=31 is a leaf node with values=[anders: 0.009, auto: 0.776, bus: 0.075, fiets: 0.047, metro: 0.019, tram: 0.009, trein: 0.009, voet: 0.056].
                                node=32 is a split node with value=[anders: 0.019, auto: 0.741, bus: 0.028, fiets: 0.085, metro: 0.009, tram: 0.014, trein: 0.009, voet: 0.094]: go to node 33 if speed_percentile_85 <= 30.46554946899414 else to node 34.
                                        node=33 is a leaf node with values=[anders: 0.061, auto: 0.306, fiets: 0.327, tram: 0.02, voet: 0.286].
                                        node=34 is a split node with value=[anders: 0.006, auto: 0.871, bus: 0.037, fiets: 0.012, metro: 0.012, tram: 0.012, trein: 0.012, voet: 0.037]: go to node 35 if bus_route_std_distance <= 595.4812316894531 else to node 38.
                                                node=35 is a split node with value=[anders: 0.009, auto: 0.897, fiets: 0.017, metro: 0.009, trein: 0.017, voet: 0.052]: go to node 36 if speed_percentile_20 <= 0.006204613484442234 else to node 37.
                                                        node=36 is a leaf node with values=[anders: 0.077, auto: 0.615, fiets: 0.077, trein: 0.077, voet: 0.154].
                                                        node=37 is a leaf node with values=[auto: 0.932, fiets: 0.01, metro: 0.01, trein: 0.01, voet: 0.039].
                                                node=38 is a leaf node with values=[auto: 0.809, bus: 0.128, metro: 0.021, tram: 0.043].
                node=39 is a split node with value=[anders: 0.004, auto: 0.38, bus: 0.027, fiets: 0.47, tram: 0.01, trein: 0.006, voet: 0.103]: go to node 40 if proportion_45_80 <= 0.024996007792651653 else to node 49.
                        node=40 is a split node with value=[auto: 0.078, bus: 0.01, fiets: 0.757, tram: 0.007, trein: 0.003, voet: 0.145]: go to node 41 if proportion_5_15 <= 0.6839917302131653 else to node 48.
```

```
        node=41 is a split node with value=[auto: 0.091, bus: 0.012, fiets: 0.825, tram:
            0.008, trein: 0.004, voet: 0.06]: go to node 42 if proportion_15_30 <=
            0.04568206053227186 else to node 43.
                node=42 is a leaf node with values=[voet: 1.0].
                node=43 is a split node with value=[auto: 0.095, bus: 0.012, fiets: 0.863,
                    tram: 0.008, trein: 0.004, voet: 0.017]: go to node 44 if
                    avg_time_diff_s <= 1.8849532008171082 else to node 47.
                        node=44 is a split node with value=[auto: 0.058, bus: 0.01, fiets:
                            0.903, tram: 0.005, trein: 0.005, voet: 0.019]: go to node 45 if
                             accuracy_percentile_20 <= 5.499175310134888 else to node 46.
                                node=45 is a leaf node with values=[auto: 0.03, bus: 0.012,
                                     fiets: 0.929, tram: 0.006, trein: 0.006, voet: 0.018].
                                node=46 is a leaf node with values=[auto: 0.184, fiets:
                                    0.789, voet: 0.026].
                        node=47 is a leaf node with values=[auto: 0.324, bus: 0.029, fiets:
                             0.618, tram: 0.029].
        node=48 is a leaf node with values=[fiets: 0.364, voet: 0.636].
node=49 is a split node with value=[anders: 0.01, auto: 0.848, bus: 0.052, fiets: 0.026,
    tram: 0.016, trein: 0.01, voet: 0.037]: go to node 50 if altitude_percentile_80 <=
    0.8099796772003174 else to node 57.
        node=50 is a split node with value=[anders: 0.019, auto: 0.837, bus: 0.058, trein:
            0.019, voet: 0.067]: go to node 51 if jerk_skew <= -1.5975200533866882 else to
            node 56.
                node=51 is a split node with value=[anders: 0.011, auto: 0.832, bus: 0.063,
                     trein: 0.021, voet: 0.074]: go to node 52 if jerk_percentile_90 <=
                    21673985.0 else to node 53.
                        node=52 is a leaf node with values=[auto: 0.765, bus: 0.235].
                        node=53 is a split node with value=[anders: 0.013, auto: 0.846, bus
                            : 0.026, trein: 0.026, voet: 0.09]: go to node 54 if
                            busway_normcount <= 3.5516588923201198e-06 else to node 55.
                                node=54 is a leaf node with values=[auto: 0.778, bus:
                                    0.222].
                                node=55 is a leaf node with values=[anders: 0.014, auto:
                                    0.855, trein: 0.029, voet: 0.101].
                node=56 is a leaf node with values=[anders: 0.111, auto: 0.889].
        node=57 is a split node with value=[auto: 0.862, bus: 0.046, fiets: 0.057, tram:
            0.034]: go to node 58 if train_route_mean_distance <= 1428.7440795898438 else to
             node 63.
```

```
                    node=58 is a split node with value=[auto: 0.89, bus: 0.012, fiets: 0.061, tram: 0.037]: go to node 59 if snap_min_value <= -2761052520448.0 else to node 60.
                        node=59 is a leaf node with values=[auto: 0.429, fiets: 0.571].
                        node=60 is a split node with value=[auto: 0.933, bus: 0.013, fiets: 0.013, tram: 0.04]: go to node 61 if max_time_diff_s <= 32.5 else to node 62.
                            node=61 is a leaf node with values=[auto: 0.979, fiets: 0.021].
                            node=62 is a leaf node with values=[auto: 0.857, bus: 0.036, tram: 0.107].
                    node=63 is a leaf node with values=[auto: 0.4, bus: 0.6].
    node=64 is a split node with value=[anders: 0.054, auto: 0.693, bus: 0.039, fiets: 0.082, metro: 0.03, tram: 0.005, trein: 0.08, voet: 0.017]: go to node 65 if railway_station_normcount <= inf else to node 72.
        node=65 is a split node with value=[anders: 0.03, auto: 0.154, bus: 0.053, fiets: 0.036, metro: 0.207, tram: 0.024, trein: 0.485, voet: 0.012]: go to node 66 if bus_route_std_distance <= 8919.11767578125 else to node 69.
            node=66 is a split node with value=[anders: 0.023, auto: 0.138, bus: 0.092, fiets: 0.046, metro: 0.402, tram: 0.034, trein: 0.241, voet: 0.023]: go to node 67 if proportion_80_120 <= 0.017564226873219013 else to node 68.
                node=67 is a leaf node with values=[auto: 0.114, bus: 0.136, fiets: 0.045, metro: 0.614, tram: 0.068, trein: 0.023].
                node=68 is a leaf node with values=[anders: 0.047, auto: 0.163, bus: 0.047, fiets: 0.047, metro: 0.186, trein: 0.465, voet: 0.047].
            node=69 is a split node with value=[anders: 0.037, auto: 0.171, bus: 0.012, fiets: 0.024, tram: 0.012, trein: 0.744]: go to node 70 if jerk_iqr_value <= 47285052.0 else to node 71.
                node=70 is a leaf node with values=[anders: 0.086, auto: 0.343, trein: 0.571].
                node=71 is a leaf node with values=[auto: 0.043, bus: 0.021, fiets: 0.043, tram: 0.021, trein: 0.872].
        node=72 is a split node with value=[anders: 0.058, auto: 0.785, bus: 0.036, fiets: 0.09, tram: 0.002, trein: 0.01, voet: 0.018]: go to node 73 if speed_average <= 23.857415199279785 else to node 76.
            node=73 is a split node with value=[anders: 0.014, auto: 0.274, bus: 0.041, fiets: 0.589, trein: 0.027, voet: 0.055]: go to node 74 if accuracy_percentile_95 <= 16.21677875518799 else to node 75.
                node=74 is a leaf node with values=[anders: 0.021, auto: 0.106, bus: 0.043, fiets: 0.745, voet: 0.085].
                node=75 is a leaf node with values=[auto: 0.577, bus: 0.038, fiets: 0.308, trein: 0.077].
            node=76 is a split node with value=[anders: 0.061, auto: 0.826, bus: 0.036, fiets: 0.05, tram: 0.002, trein: 0.009, voet: 0.015]: go to node 77 if proportion_5_15 <= 0.12189747020602226 else to node 86.
                node=77 is a split node with value=[anders: 0.062, auto: 0.877, bus: 0.021, fiets: 0.025, trein: 0.005, voet: 0.01]: go to node 78 if speed_percentile_90 <= 43.59307289123535 else to node 79.
                    node=78 is a leaf node with values=[auto: 0.286, fiets: 0.714].
                    node=79 is a split node with value=[anders: 0.064, auto: 0.892, bus: 0.022, fiets: 0.006, trein: 0.005, voet: 0.01]: go to node 80 if bus_route_max_distance <= 45133.2890625 else to node 83.
                        node=80 is a split node with value=[anders: 0.037, auto: 0.916, bus: 0.029, fiets: 0.007, trein: 0.003, voet: 0.008]: go to node 81 if proportion_30_50 <= 0.0958034060895443 else to node 82.
                            node=81 is a leaf node with values=[auto: 0.838, bus: 0.068, fiets: 0.054, trein: 0.027, voet: 0.014].
                            node=82 is a leaf node with values=[anders: 0.042, auto: 0.927, bus: 0.023, voet: 0.008].
                        node=83 is a split node with value=[anders: 0.151, auto: 0.816, fiets: 0.005, trein: 0.011, voet: 0.016]: go to node 84 if speed_median_value <= 83.79621505737305 else to node 85.
                            node=84 is a leaf node with values=[anders: 0.29, auto: 0.699, voet: 0.011].
                            node=85 is a leaf node with values=[anders: 0.011, auto: 0.935, fiets: 0.011, trein: 0.022, voet: 0.022].
                node=86 is a split node with value=[anders: 0.053, auto: 0.469, bus: 0.142, fiets: 0.23, tram: 0.018, trein: 0.035, voet: 0.053]: go to node 87 if tram_route_std_distance <= 25382.951171875 else to node 88.
                    node=87 is a leaf node with values=[anders: 0.034, auto: 0.138, bus: 0.276, fiets: 0.345, tram: 0.069, trein: 0.138].
                    node=88 is a split node with value=[anders: 0.06, auto: 0.583, bus: 0.095, fiets: 0.19, voet: 0.071]: go to node 89 if speed_percentile_90 <= 34.240671157836914 else to node 90.
                        node=89 is a leaf node with values=[fiets: 0.75, voet: 0.25].
                        node=90 is a split node with value=[anders: 0.069, auto: 0.681, bus: 0.111, fiets: 0.097, voet: 0.042]: go to node 91 if jerk_percentile_15 <= -90295644.0 else to node 92.
```

```
                                                            node=91 is a leaf node with values=[auto: 0.333, bus:
                                                                0.083, fiets: 0.5, voet: 0.083].
                                                            node=92 is a leaf node with values=[anders: 0.083, auto:
                                                                0.75, bus: 0.117, fiets: 0.017, voet: 0.033].
```

Listing 1: Python code for decision-tree based transport mode prediction

4.5 Evaluation metrics

The algorithm's performance will be assessed using precision, recall, F1-score, accuracy, and balanced accuracy, metrics commonly used in transport mode classification. Precision evaluates the model's ability to minimize false positives, while recall measures its ability to capture true positives. The F1-score combines both metrics to provide a balanced evaluation, particularly useful for imbalanced datasets. Accuracy represents the overall correctness of predictions but can be misleading in imbalanced data, where balanced accuracy offers an evaluation by averaging recall across all classes. Key definitions include true positive (TP), when the model correctly predicts the actual class (e.g., predicting 'walking' when correct), false positive (FP), where an incorrect class is predicted (e.g., predicting 'car' instead of 'bike'), false negative (FN), when the correct class is missed, and true negative (TN), when incorrect classes are correctly excluded. The formulas for each metric are:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

5 Results

First, the results based on the development data (see Section 3.1) are described. Second, the results based on the open geo-data (see Section 3.2) are described.

5.1 Evaluation on development set

The results from the confusion matrix in Table 7 and classification report in Table 8 for the training data indicate the following key findings: The confusion matrix reveals that the model performs well in predicting car and walking but struggles with categories like bus and tram, where most instances are misclassified. Bike also shows a strong prediction rate, though some misclassifications occur with cars and walking. The ‘Other’ category is highly misclassified, with many instances incorrectly labeled as car or bike, indicating potential difficulties distinguishing less common transport modes.

Table 7: Confusion matrix of training data

Observed\Predicted	**Other**	**Car**	**Bus**	**Bike**	**Metro**	**Tram**	**Train**	**Walking**
Other	0	64	0	25	0	0	5	9
Car	0	1161	2	106	5	0	20	39
Bus	0	46	3	15	6	0	3	2
Bike	0	31	0	554	2	0	4	52
Metro	0	4	0	1	27	0	8	5
Tram	0	7	0	7	3	0	1	0
Train	0	10	0	5	1	0	82	1
Walking	0	39	0	48	0	0	2	595

The classification report reveals that the model performs well for high-frequency classes like car and walking, achieving precision, recall, and F1-scores around 0.85–0.87, indicating strong and balanced performance for these categories. Bike and train also show relatively high F1 scores (0.79 and 0.73, respectively), with the train having a high recall (0.83) despite moderate precision. However, the model struggles with underrepresented classes. Other and tram have 0.00 F1-scores, as the model fails to classify these instances correctly. Buses have low performance, with an F1 score of 0.07, mainly due to extremely low recall, meaning most buses are misclassified as other categories (especially cars). Metro performs better, with precision and recall around 0.60–0.61, but still shows room for improvement.

Table 8: Classification report for training data

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Other	0.00	0.00	0.00	103
Car	0.85	0.87	0.86	1,333
Bus	0.60	0.04	0.07	75
Bike	0.73	0.86	0.79	643
Metro	0.61	0.60	0.61	45
Tram	0.00	0.00	0.00	18
Train	0.66	0.83	0.73	99
Walk	0.85	0.87	0.86	684
Accuracy			0.81	3,000
Macro avg.	0.54	0.51	0.49	3,000
Weighted avg.	0.77	0.81	0.78	3,000

Although the overall accuracy is relatively high at 81%, the macro average F1-score of 0.49 and balanced accuracy of 0.51 indicate poor performance in less common classes. The weighted average F1-score of 0.78 is boosted by the well-classified dominant classes, masking the severe misclassification of minority classes. This suggests the model may be biased towards common classes, struggling to capture the nuances of less common transport modes.

The results from the confusion matrix in Table 9 and classification report in Table 10 for the test data indicate the following key findings: The model performs well for high-frequency classes like car, bike, train, and walking, with relatively high precision, recall, and F1-scores. For instance, car has an F1-score of 0.85, and walking achieves 0.84, reflecting consistent performance compared to the training set, where these classes also had high scores. Train maintains strong recall (0.87) and a high F1-score (0.82), showing that the model reliably identifies most train instances. Similarly, bike achieves an F1-score of 0.76, demonstrating the model's ability to generalize reasonably well to this class.

Table 9: Confusion matrix of test data

Observed\Predicted	**Other**	**Car**	**Bus**	**Bike**	**Metro**	**Tram**	**Train**	**Walking**
Other	0	3	0	1	0	0	0	4
Car	0	459	0	67	6	0	9	18
Bus	0	15	0	4	1	0	0	1
Bike	0	12	0	249	6	0	1	35
Metro	0	2	0	1	4	0	5	1
Tram	0	3	0	3	7	0	0	1
Train	0	5	0	2	0	0	54	1
Walking	0	18	0	24	2	0	0	274

However, the confusion matrix highlights various misclassifications, especially for underrepresented classes. For example, 'Other' is never classified correctly, with instances being mistaken for car, bike, or walking – mirroring the training set where 'Other' had an F1-score of 0.00. Bus also performs poorly, with all instances misclassified, mostly as car or bike, leading to a 0.00 F1-score, just like in training data. This suggests the model struggles to learn meaningful patterns for rare classes, likely because of class imbalance and overlapping features.

The metro and tram classes continue to be problematic. Metro shows a slight improvement over the training set, with a 0.21 F1-score on the test set, but remains low, with many instances misclassified as train. Tram remains entirely misclassified, with an F1-score of 0.00, indicating the model failed to generalize this class from the training set to the test set. These results indicate that minority classes are poorly represented in the decision boundaries, possibly because the model is biased toward more common classes like car and walking.

Table 10: Classification report for test data

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Other	0.00	0.00	0.00	8
Car	0.89	0.82	0.85	559
Bus	0.00	0.00	0.00	21
Bike	0.71	0.82	0.76	303
Metro	0.15	0.31	0.21	13
Tram	0.00	0.00	0.00	14
Train	0.78	0.87	0.82	62
Walk	0.82	0.86	0.84	318
Accuracy			0.80	1,298
Macro avg.	0.42	0.46	0.44	1,298
Weighted avg.	0.79	0.80	0.79	1,298

Despite an overall accuracy of 80%, a macro average F1-score of 0.44 and a balanced accuracy of 0.46 reveal that performance varies widely across classes, with the model performing well on frequent categories but failing on rare ones. The weighted average F1-score of 0.79 is heavily influenced by the well-classified majority classes, masking the poor recognition of smaller classes. The test results confirm the patterns observed in training: the model captures dominant class features well but struggles with minority classes, leading to repeated misclassification patterns across both datasets.

Feature Importance

Table 11 shows the feature importance of the top 20 selected features. In total, 39 were selected. Although the hyperparameter space should have limited the number of features to seven, the final decision tree contains 39 features. This is, because the hyperparameters are not strictly enforced. The feature importance results reveal that the model relies heavily on a few key features, with ‘bus route mean distance’ as the most influential feature, contributing 22.3% to the decision-making process. This suggests that distance patterns along bus routes play a critical role in distinguishing transport modes. Speed-related features also dominate the model’s decisions, with metrics like ‘speed IQR’ (17.7%), ‘speed percentile 10’ (5.1%), and ‘speed standard deviation’ (3.1%) collectively contributing a large share of the importance. This heavy reliance on speed variation could explain the model’s struggles with modes with overlapping speed ranges (e.g., bus vs. car or metro vs. train). Interestingly, railway station count (10,4%) is another essential feature, likely helping the model identify train and metro trips. Proportion-based speed features (e.g., proportion 45–80 km/h, 7.9%) also influence predictions, possibly helping differentiate slower modes like walking from faster ones like cycling or driving. Lower-ranked features, like 'jerk percentile 85' (0.9%) and 'tram route standard distance' (0.75%), contribute minimally. Overall, the model leans heavily on speed and distance metrics, explaining its success with frequent modes like cars and bikes and its failures with underrepresented classes. Strengthening the model with more contextual features (see discussion) or refining route-based features for specific transport modes could help improve classification performance, especially for minority classes.

Table 11: Top 20 decision tree feature importance

Feature	Importance
bus_route_mean_distance	0.22
speed_iqr_value	0.18
railway_station_normcount	0.10
proportion_45_80	0.08
speed_percentile_10	0.05
proportion_5_15	0.04
speed_stddev	0.03
speed_average	0.03
speed_percentile_90	0.03
speed_percentile_80	0.02
bus_route_std_distance	0.02
bus_route_max_distance	0.02
proportion_15_30	0.02
speed_percentile_85	0.02
acc_kurt	0.02
accuracy_percentile_85	0.01
jerk_percentile_85	0.01
proportion_80_120	0.01
tram_route_std_distance	0.01
speed_median_value	0.01

5.2 Evaluation on open geo-data

The results from the confusion matrix in Table 12 and classification report in Table 13 for the test on the open geo-data show the following key findings: The confusion matrix reveals considerable misclassification patterns, particularly among specific transport modes. Walking is the most accurately predicted class, with 52 correct classifications, though it is still confused with bike (16) and car (9). Tram shows the highest misclassification rate, frequently being predicted as car (14), bike (7), or metro (5). Bus is rarely identified correctly and is often confused with car, bike, metro, and train. Similarly, the ferry is misclassified as the bike. The label 'ferry' did not appear in the observed labels in the development data. However, this label was kept to study what prediction would result for this label. This is especially interesting, because GPS signals usually get noisy when the smartphone is close to or on the water. Train and metro show moderate accuracy, but car and other categories are never correctly predicted. These results highlight challenges in distinguishing between modes with similar speed profiles and infrastructure characteristics.

Table 12: Confusion matrix of open geo-data

Observed\Predicted	Other	Car	Bus	Ferry	Bike	Metro	Tram	Train	Walking
Other	0	0	0	0	0	0	0	0	0
Car	0	0	0	0	0	0	0	0	0
Bus	0	2	0	0	1	1	0	1	0
Ferry	0	0	0	0	3	0	0	0	0
Bike	0	2	0	0	7	1	0	0	0
Metro	0	3	0	0	0	1	0	0	1
Tram	0	14	0	0	7	5	0	0	1
Train	0	3	0	0	0	0	0	6	0
Walking	0	9	0	0	16	1	0	0	52

The classification report highlights performance variations across transport modes. Walking and the train achieve the highest F1 scores (0.79 and 0.75, respectively), indicating relatively good performance. The bike also shows moderate recall (0.70) but low precision (0.21), leading to a modest F1-score of 0.32. In contrast, bus, ferry, tram, car, and other categories are never correctly identified, resulting in F1-scores of 0.00. Metro has a low F1-score (0.14) due to poor precision and recall. The overall accuracy of 0.48, the balanced accuracy of 0.32, and the macro F1-score of 0.22 reflect substantial class imbalances and misclassification issues, particularly for underrepresented classes.

Table 13: Classification report for test open geo-data

Class	Precision	Recall	F1-Score	Support
Other	0.00	0.00	0.00	0
Car	0.00	0.00	0.00	0
Bus	0.00	0.00	0.00	5
Ferry	0.00	0.00	0.00	3
Bike	0.21	0.70	0.32	10
Metro	0.11	0.20	0.14	5
Tram	0.00	0.00	0.00	27
Train	0.86	0.67	0.75	9
Walking	0.96	0.67	0.79	78
Accuracy			0.48	137
Macro avg.	0.24	0.25	0.22	137
Weighted avg.	0.62	0.48	0.53	137

6 Discussion

This report describes the development and performance of a transport mode classification algorithm for smart surveys. A decision-tree-based algorithm was developed. The results show that the decision tree model achieves a reasonable overall accuracy of 81% (balanced accuracy 51%) on the training data and 80% on the test set from the development data (balanced accuracy 46%), with a weighted average F1-score of 0.78

and 0.79, respectively. These results indicate that the model generalizes relatively well, but deeper analysis shows considerable class-specific imbalances and misclassification patterns. The model performs well in identifying high-frequency classes like car, bike, and walk, with consistently high F1 scores. In contrast, classes like bus and tram are poorly classified, with F1 scores of 0.00, suggesting that these classes are either rarely predicted or consistently confused with more dominant classes. When tested on the open geo-data, exclusively reserved for model evaluation, the results indicate that the model struggles with generalizability, particularly for less frequent transport modes. While it performs reasonably well for walking and trains, its failure to correctly classify buses, trams, and cars suggests poor generalization to unseen data. The low macro F1-score and imbalanced precision-recall values highlight a strong bias toward dominant classes, leading to frequent misclassifications. This result suggests the developed algorithm may overfit to patterns in the training data rather than learning robust, generalizable decision rules. The feature importance table highlights that OSM-based distance metrics and GPS-based speed features predominantly drive the decision-making process. This heavy reliance on distance and speed could explain the model's difficulty distinguishing between modes with similar speed ranges and infrastructure patterns. The confusion matrices reveal some systematic misclassifications, such as walking and biking. The other class instances are spread across multiple categories, reflecting this group's lack of distinctive patterns.

A non-nested rule-based algorithm was also developed as part of creating the decision-tree-based algorithm (Fourie et al. expected 2025). This rule-based algorithm achieved an overall accuracy of 85%, a balanced accuracy of 70% when evaluated on the test set from the development data. When tested on the open geo-data collected within the SSI project, an accuracy of 80% and a balanced accuracy of 83% were achieved. These results suggest that the manual rule-based algorithm generalizes better unseen data than the decision-tree-based algorithm developed in this paper. The main differences between the decision tree and the rule-based algorithm are a) single vs. multiple transport mode predictions and b) the number of predicted classes (excluding the class 'Other' and 'Ferry').

Limitations and future work

GPS signal

There are commonly known issues with GPS signals, which vary depending on smartphone and sensor configuration (Gootzen et al. expected 2025). Features about the different GPS measurements had predictive power to classify the transport mode. However, the GPS signals are only helpful to a certain extent by capturing variations in speed, but nearly no other GPS-based feature is important. This is a shortcoming of smartphone sensors. GPS-loggers, for example, are typically more accurate, especially high-end devices with better antennas. The influence of GPS signals on features and prediction quality needs more research in the future.

Features

About 200 features based on GPS and OSM were considered in the development. However, in the current version of the decision tree, only 39 features are important, which are predominantly speed (GPS) and distance metrics (OSM). The decision tree prioritized OSM distance metrics over OSM count variables. This means that counting OSM infrastructure is less sufficient for transport mode prediction. This was also shown by Fourie et al. (expected 2025), that OSM-based proximity features have more predictive power. More research is required on the quality and coverage of the OSM database. The fact that counts were not chosen could also be due to low and incomplete coverage.

Transport mode labels
The class 'Other' complicated the model training as this might contain a huge variability in transport modes. Without additional features, such a category will always remain challenging to predict. Moreover, there are errors in the labels of the development data as explained by Fourie (2025). Future studies must ensure to collect high-quality labels, especially for the public transport modes as these are most challenging to predict.

Machine learning
The decision tree and rule-based model are straightforward algorithms. Machine learning could also be considered to predict the transport mode. However, the issues encountered during development include, for example, the imbalance of data, rare classes, errors in training labels, and the absence of potentially more relevant features that will remain with this dataset. Especially contextual features, for example, owning a car or (e)bike, subscription for public transportation, the smartphone being logged in to WiFi from public transport, or the smartphone using services such as Apple Carplay or Android Auto, or other smartphone background services, are considered to have potential to improve the performance of the algorithm. No machine learning algorithm alone will solve these problems.

Performance
In the literature, sometimes better algorithm performance is reported. However, several papers only report accuracy, not the F1 or balanced accuracy. This report shows that accuracy alone can lead to over-optimistic conclusions about the algorithm's performance.

Comparability and generalization
For a fair comparison of general algorithm performance, a reference public dataset should be utilized, such as the open geo-dataset collected within the scope of this project. Otherwise, there will always be tailored solutions that do not generalize or are not comparable. The rule-based algorithm shown in Appendix D showed that it generalizes to two countries (The Netherlands and Germany) and to different sensor configurations of the smartphone app, see also Fourie et al. (expected 2025).

7 Conclusion

Transport mode prediction is central to improving the functionalities for smart time-use, travel, and mobility surveys. The decision tree model shows some promising results, but also shows issues of not being able to distinguish between transport modes with similar movement and infrastructure patterns. The high performance of dominant classes (like car and walk) comes at the expense of minority classes (bus, metro, tram), leading to poor recall and precision scores. The developed algorithm lays the groundwork for automatic transport mode prediction, while full automation of this smart feature remains a future goal.

By establishing this algorithm as a benchmark, the development provides a practical starting point for refining future models. The algorithm's simplicity and speed make it a viable option for real-world implementation, potentially alongside user prompts, to reduce respondent burden without compromising accuracy.

References

Fourie, J. J. (2025). Rules for transport mode determination in smart travel surveys [Master Thesis – Statistics and Data Science University Leiden].

Fourie, J. J., Klingwort, J., & Gootzen, Y. (expected 2025). Rule-based transport mode classification in a smartphone-based travel survey [Discussion paper, Statistics Netherlands].

Gong, H., Chen, C., Bialostozky, E., & Lawson, C. T. (2012). A GPS/GIS method for travel mode detection in New York City. *Computers Environment and Urban Systems*, *36*(2), 131–139.

Gootzen, Y., Klingwort, J., & Schouten, B. (expected 2025). Data quality aspects for location-tracking in smart travel and mobility surveys [Discussion paper, Statistics Netherlands].

Sadeghian, P., Zhao, X., Golshan, A., & Håkansson, J. (2022). A stepwise methodology for transport mode detection in gps tracking data. *Travel Behaviour and Society*, *26*, 159–167.

Schouten, B., Remmerswaal, D., Elevelt, A., de Groot, J., Klingwort, J., Schijvenaars, T., Schulte, M., & Vollebregt, M. (2024). A smart travel survey: Results of a push-to-smart field experiment in the netherlands [Discussion paper, Statistics Netherlands].

Smeets, L., Lugtig, P., & Schouten, B. (2019). Automatic travel mode prediction in a national travel survey.

Appendix A GPS features

We provide some information on features that might not be known to a general audience. Speed is the rate at which an object covers distance. It tells how fast an object is moving but does not specify direction. Acceleration is the rate at which velocity changes over time. It describes how quickly an object's speed or direction of motion changes. Acceleration is a vector quantity, meaning it has both magnitude and direction. Jerk is the rate at which acceleration changes over time. Jerk is also a vector quantity that describes how abruptly an object's acceleration changes. Snap is the rate at which jerk changes over time. Snap is used less frequently but can be important when analyzing systems where smooth motion is essential.

Bearing refers to the direction or angle from one point to another, typically measured clockwise from a reference direction (often true north) to the line connecting two points on the Earth's surface. In GPS applications, bearing is used to specify the direction in which an object or location lies relative to another point.

Altitude refers to the vertical position or height of a point above a reference surface, typically mean sea level.

- **Speed, Acceleration, Jerk, and Snap**
  - Shared features:
    - Mean
    - Median
    - Standard deviation
    - Minimum
    - Maximum
    - Interquartile range
    - Skewness
    - Kurtosis
    - 95, 90, 85, 80, 20, 15, 10, 5 percentile
  - Additional speed features:
    - Proportion at very low speed (0 – 5 km/h)
    - Proportion at low speed (5 – 15 km/h)
    - Proportion at low to medium speed (10 – 30 km/h)
    - Proportion at low to medium speed (15 – 30 km/h)
    - Proportion at medium speed (30 – 50 km/h)
    - Proportion at medium to high speed (45 – 80 km/h)
    - Proportion at high speed (80 – 120 km/h)
    - Proportion at very high speed ($\geq$ 120 km/h)
- **Bearing**
  - Features:
    - Mean
    - Median
    - Standard deviation

    * Minimum
    * Maximum
    * Interquartile range
    * Skewness
    * Kurtosis
    * 95, 90, 85, 80, 20, 15, 10, 5 percentile

- **Altitude**
  - Features:
    * Mean
    * Median
    * Standard deviation
    * Minimum
    * Maximum
    * Interquartile range
    * Skewness
    * Kurtosis
    * 95, 90, 85, 80, 20, 15, 10, 5 percentile
    * Proportion below sea
- **Accuracy**
  - Features:
    * Mean
    * Median
    * Standard deviation
    * Minimum
    * Maximum
    * Interquartile range
    * Skewness
    * Kurtosis
    * 95, 90, 85, 80, 20, 15, 10, 5 percentile
- **Time**
  - Features:
    * Trip Length (seconds)
    * Day of the week
    * Weekend indicator
- **Distance**
  - Features:
    * Trip length (km)

- **GPS Frequency**
  - Features:
    - Number of GPS measurements
    - Mean time between subsequent measurements
    - Median time between subsequent measurements
    - Standard deviation time between subsequent measurements
    - Minimum time between subsequent measurements
    - Maximum time between subsequent measurements
    - Number of long GPS gaps (a period of at least 10 min. without GPS observations)

Appendix B OSM features

- **Infrastructure counts and normalized counts**
  - Features:
    - bus station
    - bus stop
    - railway
    - light rail
    - subway
    - tram
    - busway
    - tram stop
    - railway halt
    - railway station
    - bicycle
- **Route proximity for bus, bike, metro, train, and tram routes**
  - Features:
    - Minimum distance of entire track to route
    - Maximum distance of entire track to route
    - Mean distance of entire track to route
    - Standard deviation distance of entire track to route

Appendix C Python scripts

The Git repository (https://github.com/essnet-ssi/geo-transportmode-prediction-ssi0 contains the following Python scripts that contain the code required to implement the transport mode prediction algorithm.

1. transport_mode_main.py
    - The main script for transport mode prediction that will load and run the other scripts.
2. options.py
    - This script contains all options regarding file paths, data preprocessing and model training required in the other scripts.
3. functions_general.py
    - This script contains general functions required for the transport mode prediction process.
4. gps_features.py
    - Contains functions for gps-based feature creation for events and locations data. These features are added to the events dataframe.
5. osm_features.py
    - Contains functions for osm-based feature creation for events and locations data. These features are added to the events dataframe.
6. train_decision_tree.py
    - This script runs a grid search over a hyperparameter set to train the best decision tree model for the given data. The current best result is decision_tree_ssi.pickle.
7. decision_tree_ssi.pickle
    - The best decision tree model for the available development data resulting from the combination of options, feature creation and model training.

Appendix D Rule-based transport mode prediction

The rule-based algorithm presented here was developed by Fourie (2025) and will soon be published, and therefore, we refer to Fourie et al. (expected 2025) for a detailed development description of this rule-based algorithm. Alongside the algorithm, we will present and elaborate the results of this algorithm. The algorithm was developed using the dataset described in Section 3.1. The confusion matrix in Table 14 shows the algorithm's results based on the training data. There are a large number of correct predictions for bike (560), car (825), and walk (573) indicating the algorithm performs well for these categories. Train (80) and metro (29) have relatively lower correct predictions but still show some accuracy. There are some misclassification trends. Bike is often confused with walk (62) and car (28). Car is sometimes misclassified as bike (70) and walk (30). Walk is sometimes misclassified as bike (81) and car (31). Metro and Train are occasionally misclassified as cars. For Buses and trams, the classification accuracy is poor. Bus has no correct predictions (all zeros on the diagonal for that row), meaning it is entirely misclassified. Tram has only 13 correct classifications, frequently misclassified as car (7) or walk (2). In conclusion, the algorithm performs well for bikes, cars, and walks but struggles with buses and trams. Misclassification patterns suggest possible feature overlap between cars, walking, bikes, and between metro and trains.

Table 14: Confusion matrix of training data

Observed\Predicted	**Bike**	**Bus**	**Car**	**Metro**	**Train**	**Tram**	**Walk**
Bike	560	0	28	0	6	0	62
Bus	3	0	6	0	2	0	2
Car	70	1	825	2	8	3	30
Metro	5	0	4	29	1	0	8
Train	3	0	11	2	80	0	5
Tram	0	0	7	0	0	13	2
Walk	81	0	31	2	3	2	573

It was found that some misclassifications for bike, walk, and car were due to wrong labels assigned by the user. This issue was analyzed by Fourie et al. (expected 2025) and found that misclassifications are primarily due to incorrect labeling and data quality issues. Bike trips are misclassified as walking and have an unrealistically low average speed (on average, 4.86 km/h), suggesting labeling errors. Conversely, walks misclassified as bikes have an unusually high average speed (on average, 9.54 km/h) with greater speed variation, indicating possible data inconsistencies. Car misclassifications follow similar trends. Car trips are misclassified as walking and have an average speed of 3.22 km/h, likely due to mislabeling. Car trips misclassified as bikes have an average speed of 13.4 km/h, suggesting low-quality data or user errors.

The test set confusion matrix in Table 15 shows similar misclassification trends as the training set, supporting previous findings. There are persistent misclassifications between bike and walk. Bike is often misclassified as walk (17 instances) and walk as bike (30 instances), consistent with the training set. This aligns with the previous finding that speed-based classification thresholds may be causing incorrect labeling. Car is misclassified as walk (12) and bike (25), similar to the training set pattern. This suggests difficulty distinguishing low-speed car trips from other modes, possibly due to labeling errors or data quality issues. Just like in the training set, bus has zero correct classifications, meaning the model struggles to recognize this mode entirely. There is improved performance for the metro, train, and tram. These categories had limited correct classifications in the training set but showed slightly improved results in the test set. However, some misclassification persists, particularly train being confused with car (8 instances). The algorithm struggles with low-speed distinctions, particularly bike vs. walk and car vs. walk/bike. Bus classification remains a major issue that needs further investigation. The slight improvement in metro, train, and tram suggests some learning transfer but still room for optimization.

Table 15: Confusion matrix of test data

Observed\Predicted	**Bike**	**Bus**	**Car**	**Metro**	**Train**	**Tram**	**Walk**
Bike	243	0	24	0	0	0	17
Bus	2	0	8	0	0	0	0
Car	25	0	360	2	2	2	12
Metro	0	0	1	5	1	0	0
Train	1	0	8	0	32	1	1
Tram	0	0	1	0	0	9	1
Walk	30	1	11	0	1	1	247

Table 16 shows the classification report of the algorithm based on the training set. The algorithm achieves 84% accuracy, indicating good general performance. However, balanced accuracy is much lower (65%), suggesting poor performance on underrepresented classes. There is a strong predictive Performance for the majority classes. Car (F1-score: 0.89), Walk (0.84), and Bike (0.81) are well classified with high precision and recall. These categories have the highest support (sample count), contributing to strong performance. There are severe issues with Bus classifications. Bus has a precision, recall, and F1-score of 0.0, meaning the model fails completely in identifying bus trips. This aligns with the confusion matrix, where bus instances were entirely misclassified. There is moderate performance for Metro, Train, and Tram. Metro (F1: 0.71) and Train (F1: 0.80) show acceptable performance, though metro has a lower recall (0.62), indicating missed detections. Tram has the weakest performance (F1: 0.65) among the non-bus classes, likely due to its small sample size (22). The key takeaways are, that the algorithm performs well for high-frequency classes (Car, Walk, Bike). THe bus classification is completely ineffective. Metro, Train, and Tram need improvement, likely due to lower support and feature overlap. Balanced accuracy (65%) suggests the model struggles with minority classes.

Table 17 shows the classification report of the algorithm based on the training set. The model achieves 85% accuracy, slightly higher than in the train set (84%). Balanced accuracy improves to 70% (from 65%), indicating better handling of class imbalances but still showing weaknesses. There are consistently strong classifications

Table 16: Classification report for training data

Class	Precision	Recall	F1-Score	Support
Bike	0.78	0.85	0.81	656
Bus	0.00	0.00	0.00	13
Car	0.90	0.88	0.89	939
Metro	0.83	0.62	0.71	47
Train	0.80	0.79	0.80	101
Tram	0.72	0.59	0.65	22
Walk	0.84	0.83	0.84	692
Accuracy		0.84		2470
Balanced Accuracy		0.65		2470

for the majority classes: Car (F1-score: 0.88), Walk (0.87), and Bike (0.83) perform well, similar to the training set. Precision and recall are stable across both datasets, suggesting the model generalizes well for these major classes. Bus classification still fails. This confirms that the model cannot recognize and misclassifies bus trips entirely. There are slight improvements for minority classes: Tram (F1: 0.75) improves from 0.65 in the train set, showing better recall (0.82 vs. 0.59). Metro (F1: 0.71) now has balanced precision and recall, unlike in the training set where recall was lower. Train (F1: 0.81) has similar performance but a recall drop (0.74 vs. 0.79), meaning some train trips are still misclassified. The key takeaways and comparison to the train set are that major classes (Bike, Car, Walk) maintain high performance. Bus classification failure persists. Minor classes (Metro, Train, Tram) show slight improvements, especially Tram. Balanced accuracy improves (70% vs. 65%), indicating slightly better recognition of underrepresented classes. The model still struggles with class imbalances and distinguishing low-speed modes.

Table 17: Classification report for test data

Class	Precision	Recall	F1-Score	Support
Bike	0.81	0.86	0.83	284
Bus	0.00	0.00	0.00	10
Car	0.87	0.89	0.88	403
Metro	0.71	0.71	0.71	7
Train	0.89	0.74	0.81	43
Tram	0.69	0.82	0.75	11
Walk	0.89	0.85	0.87	291
Accuracy		0.85		1049
Balanced Accuracy		0.70		1049

The algorithm allowed for multiple classifications for bus and car. This was done since it was challenging to distinguish between these two modes, even with the inclusion of OSM data. It also allowed the classification to be unknown. In the results above, it was found that the bus classification failed. It was found that Bus instances were classified as multiple classifications. This is shown in Tables 18 and 19. When the classification was (Car, Bus) it was also usually a Bus or Car.

Table 18: Multiple classifications in training data

Class Combination	**Bike**	**Bus**	**Car**	**Metro**	**Train**	**Tram**	**Walk**
(car, bus)	9	39	347	1	3	0	17
(unknown)	0	0	2	0	7	0	0

Table 19: Multiple classifications in test data

Class Combination	**Bike**	**Bus**	**Car**	**Metro**	**Train**	**Tram**	**Walk**
(car, bus)	5	33	167	0	2	0	7
(unknown)	0	0	0	0	0	0	1

The algorithm was also tested on the open geo-dataset described in Section 3.2. The results are shown in Table 20. Here, the label ‘Ferry’ was excluded. The model achieves 80% accuracy. Balanced accuracy (83%) is higher than in previous results, indicating improved performance across all classes, even those with fewer samples. Bike has perfect recall but very low precision, leading to a weak F1-score. This suggests the model overclassifies instances as Bike, resulting in many false positives. Bus and metro perform well, though metro has lower recall, meaning some metro trips are missed. Train and tram both show strong performance, with high precision and recall. Walk has high precision but lower recall, meaning some walking trips are misclassified.

Table 20: Classification report for open geo-data

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Bike	0.36	1.00	0.53	9
Bus	1.00	0.80	0.89	5
Metro	1.00	0.60	0.75	5
Train	0.89	0.89	0.89	9
Tram	0.77	0.91	0.83	22
Walk	0.97	0.75	0.85	77
Accuracy		0.80		127
Balanced Accuracy		0.83		127

Table 21 shows the multiple classifications in the open geo-dataset. Here, no multiple classifications were observed because of the absence of car trips in the data. The ‘unknown’ category, where the algorithm failed to classify a trip confidently, only occurred very few times. Thus, 127 out of the 134 tracks were classified.

Table 21: Multiple classifications in open geo-dataset

Predicted/Observed	**Bike**	**Bus**	**Metro**	**Train**	**Tram**	**Walk**
unknown	1	0	0	0	5	1

```
def ALG(df):
    def apply_classification(row):
        modes = []
        # Walking
        if (row['speed_p95'] < 13):
            modes.append('walk')
        else:
            # Bike
            if (13 < row['speed_p95'] < 30):
                modes.append('bike')
            else:
                # Tram
                if(row['min_distance_tram']<0.5 and row['
                    std_distance_tram']<250 or row['mean_distance_tram'
                    ]<100):
                    modes.append('tram')
                else:
                    # Metro
                    if(row['min_distance_metro']<1.5 and row['
                        std_distance_metro']<450 or row['
                        mean_distance_metro']<100):
                        modes.append('metro')
                    else:
                        # Train
                        if (row['min_distance_train']<0.05 or row['
                            std_distance_train']<25 or row['
                            mean_distance_train']<100):
                           modes.append('train')
                        else:
                            # Bus
                            if(row['std_distance_bus']<120 or row['
                                mean_distance_bus']<40 or row['
                                min_distance_bus']<0.015):
                                modes.append('bus')
                            # Car
                            if (30 < row['speed_p95'] < 140):
                                modes.append('car')
        return modes if modes else ['unknown']
    df['modes'] = df.apply(apply_classification, axis=1)
    return df
```

Listing 2: Python code for rule-based transport mode classification