TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Verkehrswissenschaften „Friedrich List“ Institut für Verkehrsplanung und Straßenverkehr

MASTER'S THESIS

Recognition of transport means in GPS data using machine-learning methods

Eingereicht von José María Subías Rapún (Matrikelnummer: 4812596)

Geboren am 16.01.1994 Saragossa, Spanien

Technische Universität Dresden

Fakultät Verkehrswissenschaften „Friedrich List“

Institut für Verkehrsplanung und Straßenverkehr

Professur für Verkehrsökologie

Hochschullehrer/-in: Prof. Dr.-Ing. Udo J. Becker

Betreuer/-in: Dipl.-Geogr. Stefan Huber

Dresden, den 04.07.2019

[illegible]

José María Subías Rapún

Sworn statement

I hereby declare that I am the sole author of this thesis and have written it without the help of third parties. Ideas and quotations from other sources that I used directly or indirectly are marked. This thesis has not yet been presented to an examination office and has not yet been published.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, die vorligende Arbeit selbständig und ohne Hilfe Dritter angefertigt zu haben. Gedanken und Zitate, die ich aus fremden Quellen direkt oder indirekt übernommen habe, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde worgelegen und wurde bisher nicht veröffentlicht.

Dresden, den 04.07.2019

[illegible]

José María Subías Rapún

Bibliography

Subías Rapún, José María

Recognition of transport means in GPS data using machine-learning methods

Technische Universität Dresden, Master’s Thesis, 2019

98 Pages, 37 Sources, 42 Figures, 19 Tables, 1 Appendix

Technische Universität Dresden

Fakultät Verkehrswissenschaften „Friedrich List“

Institut für Verkehrsplanung und Straßenverkehr

Professur für Verkehrsökologie

Abstract

Bicycle transport is today one of the most important measures in urban traffic with a view to moving towards more sustainable mobility. Nowadays, smartphones are equipped with Global Positioning System (GPS), which allows cyclists, through smartphone applications, to record their own routes on a daily basis, which is very useful information for traffic and transport planners.

The problem appears when there is invalid data due to errors in the measurement or in the GPS signal. The solution is transport mode recognition, which consists of classifying the different existing transport modes on the basis of a set of data. The emerging techniques of machine learning allow the development of very powerful models capable of recognizing means of transport with great effectiveness, based on other studies.

Accordingly, this study aims to separate GPS bicycle tracks from the other modes studied (inner-city train (S-Bahn), walk, bike, tram, bus), also classifying the tracks of each means of transport separately. The key contribution of this study is the design and implementation of a machine learning model capable of classifying existing modes of transport in urban traffic in the city of Dresden in Germany.

For this purpose, a cascading classifiers model was designed so that in each phase tracks belonging to a different mode are separated, studying in each phase which of the machine learning algorithms used (Decision Tree, Support Vector Machine and Neural Network) has the best performance. The GPS data was collected with the application for smartphone *Cyface* and from there it was carried out the structuring of data and calculation and selection of features that serve as inputs of the model.

To separate inner-city train (S-Bahn), bike and walk tracks (first three phases) accuracy values above 98 % are obtained for any of the mentioned algorithms. For the fourth phase, where the classification between bus and tram tracks is carried out, the performance of the model is not so outstanding, due to its similar characteristics, but nevertheless reaches an accuracy value of 83 % using a Neural Network Multi-layer Perceptron model. The great performance of the model after the training phase allowed its implementation using unlabeled tracks, achieving very good results with an accuracy of 92.6 % in the prediction of the tracks used, making only mistakes in distinguishing between tram and bus tracks.

Zusammenfassung

Das Fahrrad als Fortbewegungsmittel spielt eine wichtige Rolle in der nachhaltigen Planung des Stadtverkehrs. Mit dem Global Position System (GPS) ausgestatete Smartphones ermöglichen Radfahrern das tägliche aufzeichnen ihrer Routen. Diese Informationen sind sehr wertvoll für Logistiker, die den Stadtverkehr planen.

Ein Problem tritt auf, wenn aufgrund von Fehlern in der Messung oder im GPS-Signal ungültige Daten vorliegen. Die Lösung ist die Verkehrsträgererkennung, die darin besteht, die verschiedenen bestehenden Verkehrsmittel auf der Grundlage eines Datensatzes zu klassifizieren. Die aufkommenden Techniken des maschinellen Lernens ermöglichen die Entwicklung sehr leistungsfähiger Modelle, die in der Lage sind, basierend auf den Ergebnissen existierendener Studien, Verkehrsmittel mit großer Effektivität zu erkennen.

Ziel dieseser Studie ist, die über GPS ermittelten Fahrradwege von Routen anderer Verkehrsmittel (S-Bahn, Fußweg, Fahrrad, Straßenbahn, Bus) zu unterscheiden. Aufbauend darauf soll auch eine Klassifizierung anderer Verkehrsmittel ermöglicht werden. Hauptziel dieser Studie ist die Konzeption und Implementierung eines maschinellen Lernmodells zur Klassifizierung verschiedner Verkehrsmittel im Stadtverkehr Dresdens.

Zu diesem Zweck wurde ein Modell zur kaskadierenden Klassifikation entwickelt, welches in jeder Phase Tracks bestimmten Verkehrsmittel zuordnet und jeweils für jede Phase ermittelt, welcher verwendete maschinelle Lernalgorithmus (*Decision Tree, Support Vector Machine and Neural Network*) die besten Ergebnisse liefert. Die GPS-Daten wurden mit Hilfe der Smartphone-Anwendung *Cyface* gesammelt und nach Strukturierung sowie Berechnung und Auswahl der Variablen als Input für das Modell verwendet.

Zur Trennung von S-Bahn, Rad- und Gehwegen (erste drei Phasen) wird für jeden der genannten Algorithmen eine Genauigkeit über 98 % erreicht. Für die vierte Phase, in der die Klassifizierung zwischen Bus- und Straßenbahngleisen durchgeführt wird, ist die Leistung des Modells aufgrund der ähnlichen Eigenschaften etwas geringer, erreicht aber dennoch durch die Verwendung eines *Neural Network Multi-Layer Perceptron-Modells* eine Genauigkeit von 83 %. Die gute Leistung des Modells nach der Trainingsphase ermöglichte die Implementierung für unbekannte Routen und erzielte mit einer Genauigkeit von 92.6 % sehr gute Ergebnisse, wobei nur Fehler bei der Unterscheidung zwischen Straßenbahn- und Busgleisen gemacht wurden.

Contents

List of Figures

List of Tables

Abbreviations

GPS	Global Positioning System
ITS	Intelligent transportation systems
SVM	Support Vector Machine
RBF	Radial Basis Function
DT	Decision Tree
NN	Neural Network
MLP	Multi-layer Perceptron
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
GIS	Geographical Information Systems

1. Introduction

For the analysis of travel demand, transportation planning and traffic organization, the information on the means of transport used on a trip is a key factor. This information can be very useful to know the behavior of the traveler in order to develop strategies to reduce travel time, traffic congestion and air pollution [1].

Travel surveys are important tools in the planning, design, evaluation and maintenance of the transportation system. The travel surveys have been found in different formats, in which mail-in paper forms, travel journals as well as computer-assisted telephone interviews stand out [2].

The study of transport in modern cities, of increasingly dynamic conditions, resembles complex. The traditional method of surveys exhibits incompatibilities since it collects data only at a single point in time, when cities show constant changes. In addition, the trips described in surveys are not very accurate, as the trip made differs from the trip described for the survey [3].

Faced with this situation, Global Positioning System (GPS), a cost-effective technology, has appeared as an alternative, which allows travel data to be collected, reducing time and workload costs for transport professionals [1]. GPS devices can record individual travel routes automatically, unlike traditional surveys that depend on what the interviewee remembers.

Due to the increasing popularity of the smartphone, which is becoming one of the necessities of daily life, new opportunities have arisen for the practical use of GPS-based survey methods, since in most cases it is equipped with GPS module, allowing to replace exclusive devices for measuring data with GPS [4]. GPS technology allows multiple data to be collected from various route points in real time, which is a major advance in the study of the inherent dynamic and complex behavior of human activity in cities [3].

GPS data provides information on the origin and destination of the route, route used and travel speed, but it is not possible to obtain direct information on the mode of travel used (means of transport), which is a very important for transport planning [5]. It is not feasible for users to manually label modes of transport on their GPS routes. This is primarily due to a lack of motivation to do so, as users see no direct benefits to labeling their tracks. It is also a complex task, as trips usually involve more than one means of transport and it is difficult for the user to remember the exact time of change of mode on the route [6].

Machine learning methods promise to be a valuable alternative to statistical techniques for determining travel modes, advances in the field of machine learning have led to the development of powerful classifiers. Instead of making strict assumptions about the data, machine learning approaches learn how to model complex patterns in a data-driven manner [7].

This study, carried out at the Chair of Transport Ecology at the Technical University of Dresden, aims to collaborate in the improvement of bicycle transport. Trying to contribute to more sustainable mobility (noise and air pollution) is one of the main motivations for this study, which has as its main objective to separate the GPS bicycle tracks from the rest of the studied modes, although an individual classification will be made as a greater challenge.

Applying the problem to bicycle transport planning, the possibility for cyclists to record their own routes using the smartphone is a great advantage in terms of improving work efficiency, which provides a great deal of useful information for traffic planners.

The problem occurs when the user records the track incorrectly, for example when continuing to track the route using another mode of transport. Besides, failures in the GPS signal can also lead to incorrect measures. When analyzing the collected data, there are cases where the track does not belong to the described mode, due to measurement failures.

Due to these potential errors, bike tracks must be checked and separated from other modes of transport. The prototype system and method of this study provides answers to the following two key questions:

a) Is it feasible to design a machine learning method capable of distinguishing GPS bicycle tracks from the other modes of transport analyzed?

The machine learning classification model must be able to classify the different modes of transport proposed, based on the features calculated and then fed into the algorithm. It is a challenge in some complicated cases since the behavior of some means of transport can be similar in different situations. For example, when measuring or calculating speed in a range that is feasible for several modes of transport, it is complex for algorithms to discriminate perfectly the mode of transport used if only speed is used as a feature. In practice, this often occurs for fast walking and slow cycling, for cars and buses in on a congested road and trams and buses. It is, therefore, crucial to know how to calculate which variables are appropriate for each transport mode as well as having data variability, since the bicycle speed for an older person is not the same as for a younger person, or there are routes with more traffic lights than others that can change the behavior of vehicles. To address this problem, one possible solution is to extract more features from a

GPS track (e.g. speed or acceleration percentiles). However, if many features are generated to cover more aspects of GPS trajectories, the challenge of applying an effective process of reduction in the dimension of the variables must be carried out.

On the basis of other studies that will be analyzed, as well as this study itself, it is possible to model a machine learning system that allows the classification of the different modes of transport, with greater or lesser effectiveness, which depends to a great extent on the design of the algorithm as well as on the variables fed.

b) What features are needed to implement the model?

Faced with a problem of classification through a machine learning model, the design phase of the input variables is critical. After the data collection phase, it is crucial to study the different input features that can most faithfully represent each class (in this case each means of transport). In this phase, the engineer's experience in the field plays a key role. Research on the subject from other related studies is a great source of information for the calculation of features. Data collection, calculation and study of potential input variables and simulation of the different algorithms will be conducted to study which features are the most effective to carry out the classification process.

Altogether, the main objectives of this study include:

- Research from other studies related to the topic of classification of modes of transport using machine learning techniques.

- Collect data on the modes of transport on which the classification is to be carried out.

- Develop an algorithm for data structuring and calculate machine learning model inputs.

- Designing the machine learning model.

- Train and validate the developed machine learning model with the data collected, feeding the selected features.

- Analysis and evaluation of the model in order to optimize the fed features and parameters of the algorithm. Discussion of the results.

- Implementation of the model to classify non-labeled GPS tracks

2. Basics

2.1. GPS Data

“The term ‘GPS data’ covers global, satellite-based positioning data”. With the increase in GPS receiver devices, as well as simple recording software, this data can reflect an accurate picture of how people move around on a daily basis [8].

GPS is a generalized positioning tool that collects space-time information from people and objects carrying a GPS-enabled device, such as a smartphone. The great recent penetration of the smartphone in the global market, being used daily, makes it an advantageous device compared to other equipment with GPS measurement. As a result, it is possible to collect a massive amount of data at daily level and in real time on the movement of people and different vehicles in cities.

If it is possible to obtain information (speed, time, location, etc.) from the devices in real time anonymously, they can minimize the costs of traffic planning due to the large amount of information obtained, complementing traditional sensors [9].

The GPS trajectory of an object is built from the connection of the different points measured along it by the device enabled with GPS technology. Latitude, longitude and timestamp information can be found at each GPS point obtained. From these variables other variables can be calculated as speed or acceleration easily. In this study, in addition to the three variables mentioned above, the speed at each GPS point is also obtained by means of the application of the smartphone used.

The data obtained by means of GPS measurements simply provide geometric and temporal information. This is why specific data mining methods are applied in such a way that information about the mode of transport can be obtained. However, due to the fact that a simple GPS trajectory can be composed of several modes of transport, most approaches include two steps: segmentation of the trajectory into a series of single travel-mode and assigning the transport mode to the existing segments [10]. In this study the second step is carried out, as the data analyzed correspond to individual labelled trips.

The study of the patterns of movement of people based on GPS data has carried out the development of applications dedicated to the study of the behavior of human activity in transport, such as learning significant locations, detecting anomalies, location based activity recognition and identifying the mode of transport used, the latter being the objective of this study [1].

Surveys based on data collected using GPS techniques with smartphones also have some disadvantages that should be mentioned, such as the low battery life of smartphones (compared to dedicated GPS devices), unstable signal acquisition in some places or loss of it (when travelling underground or in urban canyons), the cost of transferring data from smartphones to data centers. Due to the possible existing drawbacks, it becomes a key factor to choose an appropriate methodology for data processing [4].

The topic of data protection plays a key role in the collection of GPS data based on smartphones. In Germany, where this study was conducted, there are data protection laws regulated at the state level, where the area of use is subject to the supervision of the individual federal state. The Data Protection Act of Saxony (federal state of Dresden, city of the study) dictates that data providers must provide anonymous data in such a way that it is impossible for customers to obtain individual information from participants [8].

2.2. Machine learning in transportation

"Machine learning is a collection of methods that enable computers to automate data-driven model building and programming through a systematic discovery of statistically significant patterns in the available data". With the development of computer and communication technologies, it has been possible to identify more effectively the patterns in data by means of machine learning algorithms [11].

Machine learning modes can be classified according to the type of learning. In supervised learning, labeled data is used to drive the learning process [11], the algorithm develops a function that leads from input to output based on examples of input-output pairs [12]. In unsupervised learning, unlabeled data is used, the algorithm learns from the patterns and relationships in the input data. Semi-supervised learning uses both labeled and unlabeled data. Finally in learning by reinforcement a series of feedback/reward cycles guide the process [11].

Intelligent transportation systems (ITS) enable efficient transportation improvement, providing travel safety as well as more options for travelers. Data processing by means of machine learning algorithms enables the generation of useful information as well as new functions and services in intelligent transport systems (ITS) [13].

2.3. Support Vector Machine (SVM)

"Support vector machines (SVMs) are risk-based supervised learning methods that classify data patterns by identifying a boundary with maximum margin between data points of each class” [11].

A classification problem entails the separation of the data set into training and testing sets. Every instance in the training set contains a target variable (e.g. transport modes) as well as a set of features (calculated/observed variables). The model (based on training data) must be able to predict the target variables of the test data given only the test data attributes [14].

SVM algorithms aim to find hyperplanes that are capable of separating data groups (classes) so that the distance between the support vectors is maximum. The support vectors are defined by the members of each class closest to the hyperplane. This distance can also be defined as *set error margin*. The objective is to obtain a function that leads from the inputs variable to the output variables so that the predicted output does not differ from the current output more than the set error margin. SVM tries to look for a hyperplane so that this margin is maximized. The greater the margin between classes, the greater the chances that the new input will be correctly classified [11], [15]. A graphic example of this is shown in Figure 2.1.

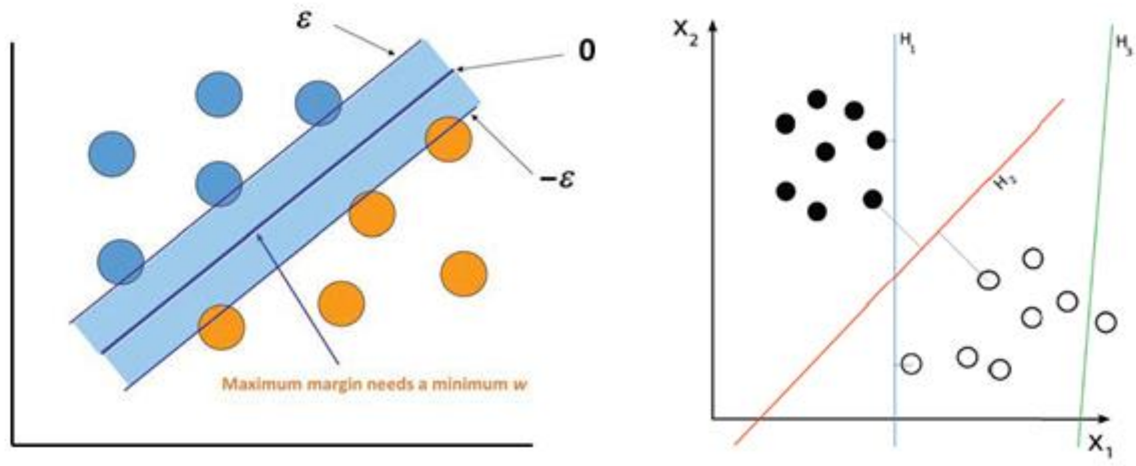


Figure 2.1: Concept of SVM [11] *(left), Linear classification problem. It is observed how the hyperplane H2 (red) is able to separate both classes maximizing the margin, while H1 (blue) separates but does not maximize and H3 (green) is not able to separate the two sets of data* [15] *(right).*

It is a constrained optimization problem. It is an optimization problem because the objective is to maximize the set error margin and the limitation is that the points closest to the hyperplane that define the support vectors cannot be within the margin.

From a labeled dataset (input-output pairs) $S = \{(x_i, y_i)\}_{i=1}^{n}$, where $y_i \in \{-1, +1\}$ represents the class label of a point x_i i.e., $(x_i, y_i) \in R^{d+1}$ where *d* and *n* are the numbers of features and labeled data points respectively, the SVM solves the optimization problem represented in Equations 2.1, 2.2 and 2.3 [10], [11].

SVM classifies the data classes through a hyperplane contained in the decision function (Equation 2.1) [16]:

$$f(x) = sign(\langle w, \phi(x)\rangle + b) \tag{2.1}$$

The solution *w* for SVM is defined in the Equation 2.2 by the support vectors, which are on the margin and carry all the relevant information for the classification problem [16].

$$w = \sum_i \propto_i \Phi_i(x_i) \tag{2.2}$$

In order to optimize the calculation of the hyperplane, with a maximum margin between the two classes, the following approximations appear, which allow to extend the problem of binary classification to multi-classification. Equation 2.3 shows the formulation of SVM to solve the classification problem, where two terms are observed. The objective of this function is to minimize the first term, which is equivalent to maximizing the margin between classes. Equations 2.4 and 2.5 represent the associated constraints [16].

$$min_{w_i \xi_i b}\left(\frac{1}{2}w^T w + C\sum_{i=1}^{n} \xi_i\right) \tag{2.3}$$

$$y_i(w^t \phi(x_i) + b) \geq 1 - \xi_i \tag{2.4}$$

$$\xi_i \geq 0\,; i = 1, \ldots, n \tag{2.5}$$

where y_i takes the values -1 or 1 depending on the class to which the x_i point belongs, parameter *w* is the normal vector to the hyperplane and parameter *b* represents the offset of the hyperplane from the origin along the normal vector *w*. The magnitude of penalty for incorrect classification is measured by parameters *C* and ξ_i. The $\phi(x_i)$ function in Equation 2.4 projects the training vector x_i to a higher dimensional space. Equation 2.5 performs the limitation to points with positive errors [10], [11], [15], [17].

When the hyperplane separation turns complicated, it becomes a solution to map the data points (input-output) pairs to a higher dimensional space to find the hyperplane. At this point, SVM becomes more effective when combined with Kernels. The $\phi(x_i)$ function in Equation 2.4 is used to define the function of the Kernel. The Kernel function is used to project the input vector into generally a higher dimensional space where this non-linear separation problem is transformed into a linear one in this new feature space, where learning process takes place. The Kernel functions return the inner product between points in a suitable space, where similarity is defined. One of the advantages of using SVM is the low computational cost despite working in large spaces with Kernel functions [11], [15], [16]. The following equation is called the Kernel function:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) \tag{2.6}$$

The following equations show some basic Kernel functions that can be combined with SVM [14]:

- Linear: $K(x_i, x_j) = (x_i)^T (x_j)$
- Polynomial: $K(x_i, x_j) = (\gamma (x_i)^T (x_j) + r)^d;\ \gamma > 0$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma (x_i)^T (x_j) + r)$

A Gaussian Radial Basis Function (RBF) Kernel, depicted in Equation 2.7, is used in this study. This is a non-linear separation problem, so a Gaussian Kernel is more appropriate than a linear Kernel. RBF adapts to problems where the relationship between classes and attributes is both linear and non-linear [10], [15].

$$k(x_i, x_j) = \exp\left(-\gamma \left\|x_i - x_j\right\|^2\right) \tag{2.7}$$

In the training process, there are two parameters for the RBF Kernel that must be optimized, *C* and *gamma*. The inverse of the radius of influence of the support vectors selected by the model can be understood as the *gamma* parameter. High *gamma* values have a low influence while low values go further [18].

The parameter *C* compensates on the one hand the correct classification of the training examples and on the other hand the maximization of the decision function’s margin, it is understood as a regularization parameter. For a high value of *C*, there will be a greater accuracy of classification and a smaller margin. For a low value of *C*, a higher margin and a simpler decision function will be allowed, at the cost of losing classification effectiveness in training process [18].

2.4. Decision Tree Learner (DT)

Decision trees are non-parametric supervised learning methods used for classification and regression. The purpose is to develop a model that predicts the value of a target variable by learning simple decision rules from the data features [18]. In parametric models, the structure of the model can be specified a priori, while in the case of non-parametric models, the structure of the model is determined from the data. In decision trees, the structure is not defined in the first instance, but grows, branches and leaves are added in the learning process that depends on the complexity of the problem [19]. In comparison to other classifying algorithms, decision trees are easy to interpret and understand. They are robust to biased distributions, but small changes can drastically alter results. In addition, another disadvantage of decision trees is that they can be easily overfitted [20].

The resolution process ends when all possible decisions and outcomes are considered, resulting in a tree-like structure, where the logical sequence of decisions leads to the initial decision. In a classification problem, which is what this study is about, the process follows the structure of the tree in a hierarchical way, so that the next answers to the questions depend in turn on those that are currently formulated [11].

A decision tree is made up of internal decision nodes and end leaves. Each *m* decision node implements a $f_m(x)$ function with discrete results labeling the branches. In each node the test is performed and the branch is chosen according to the result, starting from a given input. The process is repeated until the leaf nodes are reached, whose value represents the output result [19]. A simple example can be seen in Figure 2.2, which consists of developing a decision tree from the initial dataset separated into classes.

To create the decision tree model, some criteria are important, such as the definition of attributes at each level or node, the number of splits or branching factor and the maximum depth of the tree. If the model finally classifies the data patterns correctly, the separation between branches and classes is considered pure. To measure the impurity between classes or branches there are methods such as entropy-based impurity, Gini impurity or misclassification impurity [11].

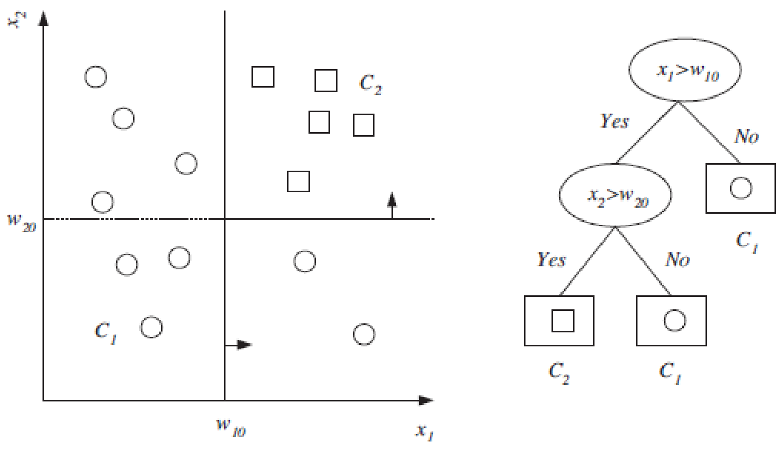


Figure 2.2: Example of the dataset and decision tree. The oval nodes are the decision nodes and the rectangular ones represent the terminal nodes. It is observed how the decisions are made for values in the two axes and the classification is produced [19]**.**

The model looks through a variable to get a value for a variable that divides data into two or more groups. The best split is the one that minimizes the error in the resulting subsets. Minimizing error is also known as minimizing the degree of impurity. Therefore, in order to find the best split, the degree of impurity of the child nodes must be measured. To measure the impurity between classes or branches there are methods such as entropy-based impurity (Equation 2.8), Gini impurity (used to measure how often the element of a set can be incorrectly labelled) (Equation 2.9) or misclassification impurity (Equation 2.10) [11], [20]. The following equations, obtained from S. H. Fang *et al.* [20], show the named methods:

$$Entropy = H(x) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2.8}$$

$$Gini\ impurity = 1 - \sum_{i} p_i^2 \tag{2.9}$$

$$Classification\ error = 1 - \max(p_i) \tag{2.10}$$

where p_i is the probability function of the i-th sample.

2.5. Neural Network (NN)

Neural networks or artificial neural networks are designed to mimic the nervous system, in terms of architecture and function. Since their introduction, they have gained popularity as powerful tools in the field of machine learning and data analysis [11].

Multilayer Perceptron (MLP) neural network is a supervised learning algorithm that learns a $f(\cdot) = R^m \rightarrow R^o$ function from the training of a dataset, where *m* represents the number of input dimensions and *o* the number of output dimensions. From a set of features and a target variable, the algorithm is capable of learning a non-linear method for classification or regression. Between the inputs and outputs layer there may be one or more non-linear layers, known as hidden layers [18].

Neural networks are made up of units called nodes, which are connected to each other. The union between two units propagates the activation of one to the other unit. In addition, each union has an associated numerical weight, which measures the strength and sign of the connection [12]. For each unit there is a dummy input ($a_0 = 1$) as well as a bias (B) for the neuron. These parameters can be seen in Figure 2.3.

Each unit first makes a weighted sum of its inputs (Equation 2.11) and then applies the activation function (Equation 2.12) to obtain the output. The activation or transfer function is typically non-linear, such as a radial basis or sigmoid [11], [12].

$$inputs_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{2.11}$$

$$a_j = g(inputs_j) = g(\sum_{i=0}^{n} w_{i,j} a_i) \tag{2.12}$$

Figure 2.4 shows a neural network structure. The first layer is the input layer, whose neurons represent the input features. In the second layer, the hidden layer, the linear sum of the inputs is made with the effect of the weights of the connections and the activation function is applied. The last layer, the output layer, receives the values of the last (in this unique case) hidden layer and transforms them into the output values [18].

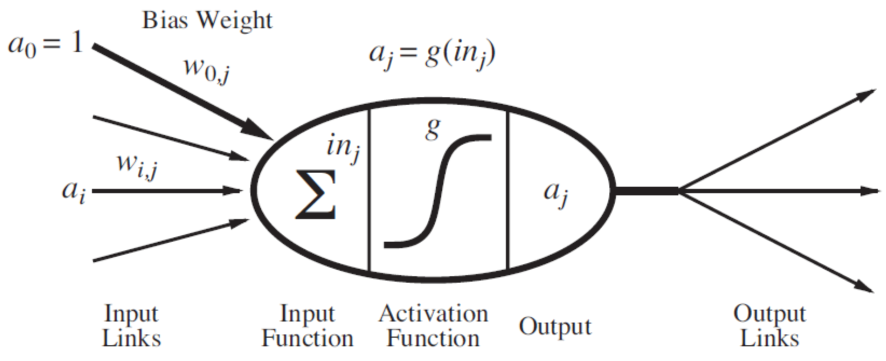


Figure 2.3: A simple mathematical representation of a neuron is observed, where a_i represents the output activation of unit i and $w_{i,j}$ is the weight of the connection from unit i to the one represented in the figure [12].

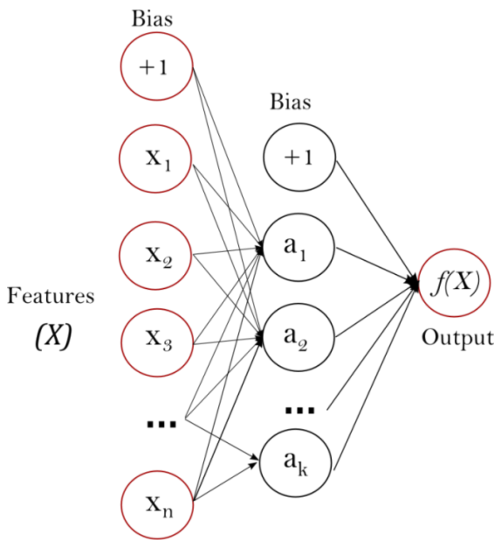


Figure 2.4: One hidden multilayer perceptron neural network [18].

The neuron can be adapted to input-output pairs so that it learns existing patterns from training cycles in the hidden layer [5]. The weights, which connect the input layer with the hidden ones and the hidden ones with the output ones, represent the backbone of the neural network, stand for the trained neural network and are used to evaluate new data whose output value is unknown. The learning process consists in determining the correct values for these connections. In the neural network training phase, the calculated output value is compared with the correct known value being used for training. If the value is not correct, the network performs a readjustment of the weights of the input, hidden and output layers, known as the backpropagation method. This method propagates the error from the output layer to the input layer, improving effectiveness by changing weights and biases. For the Backpropagation model, inputs must be normalized between 0 and 1 if the activation function is a sigmoid and between -1 and 1 for the hyperbolic tangent, for each of the dataset input vectors [2], [11], [21].

The training process is repeated until a certain user-defined effectiveness is reached. This limit can also be defined by a number of *epochs*, which represent the iterations to be performed. For the neural network model, different adjustments can be made that affect the training process, such as the number of hidden layers, the learning ratio and the number of epochs. Based on the attributes of the data and the parameters chosen in the network design, it is able to automatically identify the characteristics of each mode of transport and make a correct classification of new trips [2].

Figure 2.5 shows an example from the study of P.A. Gonzalez *et al.* [2], which represents a neural network model for classifying modes of transport, which is very similar to the case of this study.

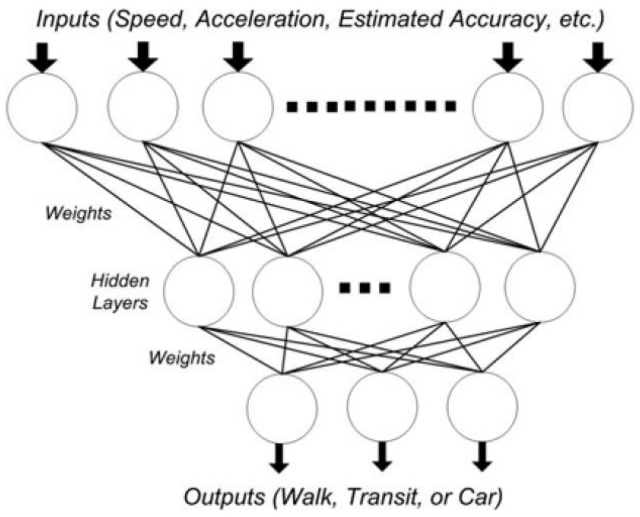

Inputs (Speed, Acceleration, Estimated Accuracy, etc.)
Weights
Hidden Layers
Weights
Outputs (Walk, Transit, or Car)

Figure 2.5: Neural network scheme designed for the classification of modes of transport. It is observed in the layer of inputs the fed features and in the layer of outputs the classes of transport

modes that are required to classify [2].

2.6. Normalization

An important phase in data preprocessing is normalization or standardization. It is a process that consists of data conditioning within certain limits to reduce redundancy, eliminate numerical problems and improve interpretation of the results. The scaling of the features, through the normalization of their values, allows a better performance of the model and training speed. Some known standardization methods are logarithm transformation, square root transformation or Gaussian and Poisson distribution [11], [17].

When applying some machine learning models (such as SVM), the input vectors must be normalized. Model prediction may vary depending on the standardization method applied, so it is important to choose an appropriate method to get the most out of the model's performance [21].

The standardization process of a data set is a common requirement for many machine learning models. If data is not standardized, the model may not behave properly. This is why the individual features fed to the system should appear as a normal data distribution, such as Gaussian, with a mean value of 0 and a unit of variance. Elements used in the objective function, such as the radial basis function of Support Vector Machines of a learning algorithm assume that all attributes are centered on 0 and have a variance in that order. The problem arises when the order of magnitude of the variance of a particular attribute is considerably greater than that of others. In this case it would have a greater effect on the learning process, since it would dominate the objective function and would not allow the machine learning model to learn from the rest of the attributes as expected [18].

The normalization of an input or output component *x* is carried out through of the following equation (Equation 2.13), where μ and σ are the mean and standard deviation of the variable *x* respectively.

$$x_{normalized} = \frac{x - \mu}{\sigma} \tag{2.13}$$

Normalization can also be performed by rescaling to new values as desired. Equation 2.14 shows the formula necessary for rescaling, where the current value of component *x* with its maximum and minimum is observed, as well as the new maximum and minimum values set to define the desired interval for rescaling [11].

$$x_{rescaled} = \frac{(x - x_{min})}{(x_{max} - x_{min})} \cdot \left(x_{new,max} - x_{new,min}\right) + x_{new,min} \tag{2.14}$$

2.7. Principal Component Analysis (PCA)

The main objective of the principal component analysis (PCA) is to transform a set of original variables into a new set of variables called principal components. PCA analyzes the correlation between variables. Two quantitative variables are correlated when the values of one of them vary systematically with respect to the homonymous values of the other. The objective is to try to create variables that are a linear combination of the originals, so that the original variable can be explained by the new variable created. If the PCA is reliable, when the original variables change in one direction, the new ones will change in the same direction.

The final result consists of a summary of the original variables without loss of information with respect to the original dataset. The final components do not depend on each other, each one contains information of the possible different correlations between the totality of the variables.

As PCA is an algorithm to reduce the dimension of the original total features, it allows the reduction from a higher dimension level to two or three dimensions so that it is possible to visualize and better understand the data. It is necessary to standardize the data before applying PCA by scaling the features to obtain an optimal performance, as it happens for many machine learning algorithms.

Moving from one space with more dimensions to another with less dimensions leads to loss of information. Reducing dimensions may not be the best solution if it involves a high loss of variance. Therefore, it is important to analyze the number of dimensions to which the original variables are to be reduced in order to avoid a great loss of information, making the procedure ineffective.

The objectives pursued by the PCA are to extract the most important information that can explain a set of data, compress the size of the dataset while keeping the relevant information, simplify the description of the dataset and analyze the structure of observations and variables. As previously described, the new variables are obtained as a linear combination of the original ones. The first principal component requires having the highest percentage variance, which means that this component is the one that explains most of the dataset variance. The second component is calculated on the condition that it is orthogonal to the first component and that it has the highest possible percentage that explains the variance of the data, which will be lower than that of the first principal component. All other principal components are calculated using the same procedure. The values of these new variables for observations are called factor scores. Geometrically they can be understood as the projection of the observations in the principal components [22].

PCA is a widespread technique used for applications such as dimensionality reduction, feature extraction, and data visualization. There are two common definitions for the PCA, from which the same algorithm is obtained. PCA can be defined as the orthogonal projection of data in a space of smaller linear dimension, known as principal subspace, so that the variance of the projected data is maximized. Equivalently, it is defined as the linear projection that minimizes the mean square distance between the data points and its projections (also known as average projection cost) [23]. The process of orthogonal projection can be seen in Figure 2.6.

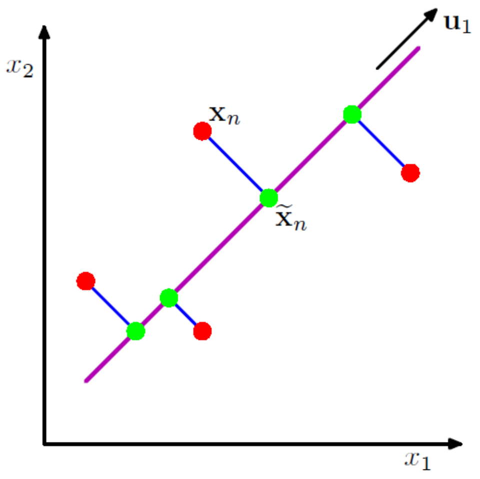


Figure 2.6: PCA tries to find a space of lower dimensionality, known as the principal subspace, represented by the magenta line. PCA tries to maximize the variance of the projected points in the subspace (green dots). Green dots represent the orthogonal projection of the original data (red dots). In other words, PCA tries to minimize the sum of the squares of the projected errors (blue line) [23].

As described above, there are correlations between the calculated features, which generate redundancy in the available dataset information. Having redundant data means that the same information is spread across multiple features. This information can be exactly the same or have variations. In the first case, there is total collinearity between the variables. In the second case, could be collinearity between two variables of the set or multicollinearity between more than two variables of the set [24]. In summary, in a system where there may be redundant information that can be explained by other variables, it is important to remove this redundancy.

Related to data redundancy, it is important to mention the following terms [24]:

- Variance: redundancy is unique to a particular feature. If it is correlated or associated with the output, it can add direct contribution to the output prediction. The variance explains how large the changes in a feature's data are, how spread out they are.

- Covariance: redundancy is common to several features, as they are correlated. In this case, if the common information is relevant to the output, the algorithm must decide which feature has more weight when choosing between the features. The covariance explains how large the correlation between two or more variables is.

- Random noise component: information due to measurement problems or random failures that do not contribute to the good performance of the model. When there is noise, it is not possible to obtain a good representation of the data.

The above three terms go together. To reduce the impact of noise, the minimum set of variables that maximize the performance of the machine learning model must be selected. Another method is to merge redundant information together using a weighted average. This consists of creating a new feature whose principal component is the shared variance of multiple features [24].

When creating the new feature from the existing ones, the weight of each variable is calculated from the technique known as Singular Value Decomposition (SVD). SVD has as applications to compress the data and find hidden patterns in the data. For data compression purposes, PCA uses SVD to achieve the goal [24].

2.8. Model evaluation

It is important to evaluate the performance of the machine learning model. Starting from the original data set, one part is used for the training phase and there should be another used to test that the learning has been done correctly. For this purpose, before feeding the inputs into the machine learning model, a division is made between test and training set. Once this step has been carried out, the training set is used to train the machine learning algorithm while the test set is used to measure its effectiveness. Finally, all data will have been used for training and testing but never at the same time [11].

The k-fold cross-validation is one of the widely accepted techniques used in machine learning systems to estimate the performance of the model [11]. The k-fold cross-validation performs random divisions of the data set (with known correct outputs) into *k* blocks. For each iteration, a block is used as a verification dataset and the remaining *k-1* blocks are used as a training dataset. It is defined for each iteration what percentage of the dataset is used for verification and what percentage for training. This process is repeated *k* times, with a different block for the verification set in each of these iterations. The mean of the *k* process effectiveness results obtained from the verification datasets evaluates the performance of the model. This measure assumes an estimate of the model's behavior on unknown future data. The way to measure the performance of the different models used is through the effectiveness ratio in the classification problem. When comparing the different machine learning models implemented, the one with the highest effectiveness ratio in the classification success is the one that has performed best [2], [15]. Figure 2.7 represents a schematic of how the method works.

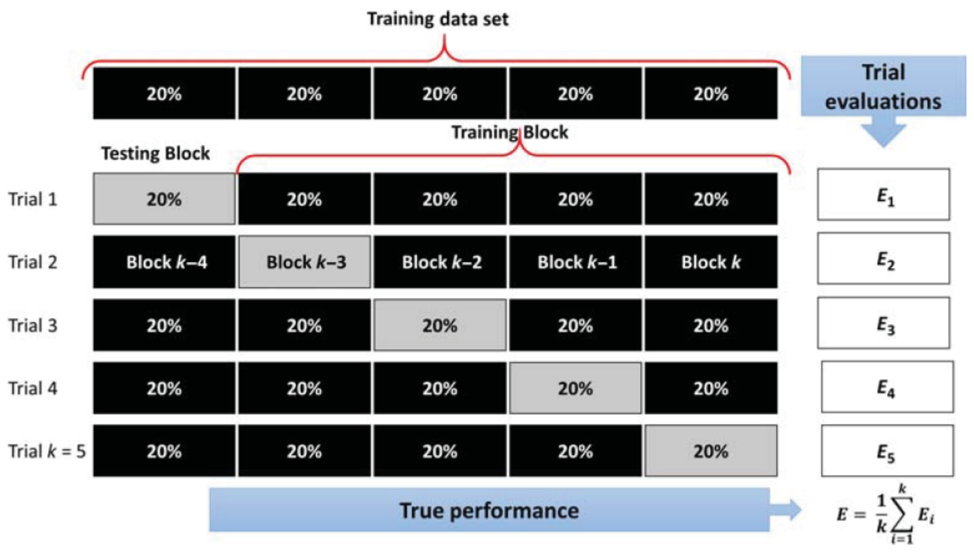


Figure 2.7: Example of the k-fold cross-validation method for a sample of k = 5. It is observed how the original dataset is divided into blocks representing 20 % of the data. In each iteration, a different block is used as test data (20 %) and the remaining blocks are used as training data (80 %) [11].

Overfitting occurs when the algorithm is over-adapted to the training set and therefore does not perform correctly in the testing phase. The goal in the training phase is to minimize the performance error, while in the testing phase is to try to make correct predictions on unseen data. The problem of overfitting occurs when the algorithm memorizes in the training phase instead of learning. Methods such as cross-validation act as a powerful tool to tackle the problem of overfitting [25].

The problem of overfitting can be solved by adjusting the model with a sample for training and another for testing. It is important not to choose a single partition to evaluate effectiveness, but a large number of them through methods such as cross validation [15]. Figure 2.8 and Figure 2.9 show the example of a classification problem to observe the overfitting case, where a Support Vector Machine (SVM) is used as classifier.

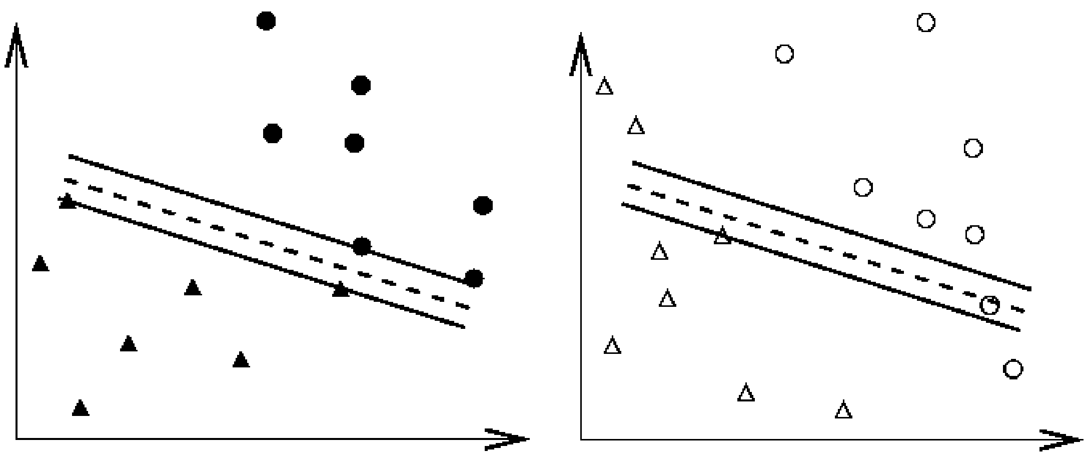


Figure 2.8: It is observed that the accuracy of the test is not good (right). In the test phase the classifier does not manage to make a correct separation of the classes, because the classifier overfits the training data (left). The filled figures represent the data in the training phase while the empty ones represent the data in the test phase

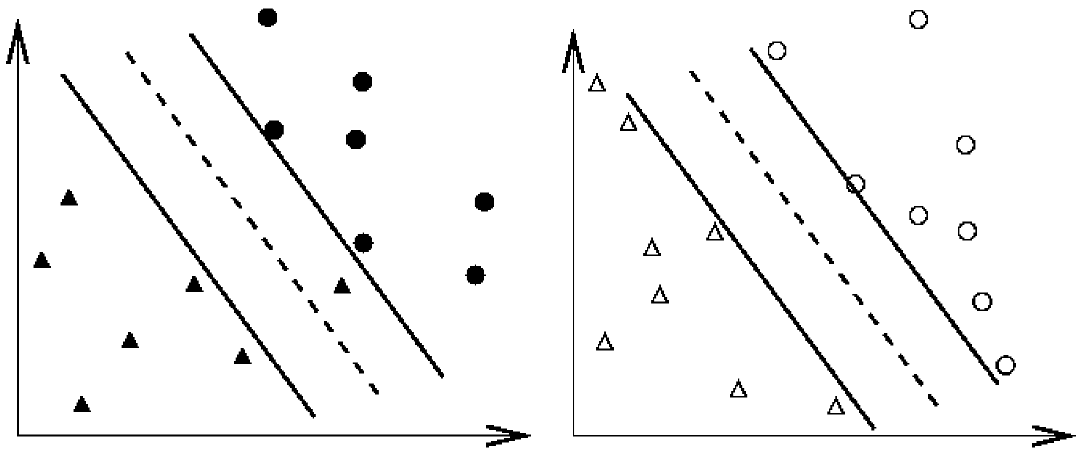


Figure 2.9: The classifier is not over-adjusted to the training data (left) and allows to obtain a better classification of the test data and therefore a better performance (right). The filled figures represent the data in the training phase while the empty ones represent the data in the test phase.

2.9. Evaluation measures

A common method of evaluating machine learning experiments as pattern recognition or binary classification is to use the Recall, Precision, Accuracy, and F-factor metrics.

To explain these metrics it is important to first define the terms *true and false positive and true and false negative*. It is common to introduce these evaluation measures in the context of a binary classification problem, where classes are positive and negative and predictions are summarized in a four-cell contingency table, also known as the confusion matrix [26].

From a classifier and an instance that is to be classified, there are four possible results. If the instance belongs to the positive class and is classified as positive, the result is true positive, if it is classified as negative, the result is false negative. If the instance belongs to the negative class and is classified as negative, the result is *true negative*, if it is as classified positive, the result is *false positive* [27].

Figure 2.10 shows the confusion matrix for the generic binary classification problem, where *p* (positive) and *n* (negative) represent the current classes, and *Y* and *N* represent the predictions of the classes produced by the model. The major diagonal represents the correct decisions made, and the results outside the diagonal the errors (confusions). *P* (total positives) is the sum of true positives and false negatives and *N* (total negatives) is the sum of *false positives* and *true negatives*.

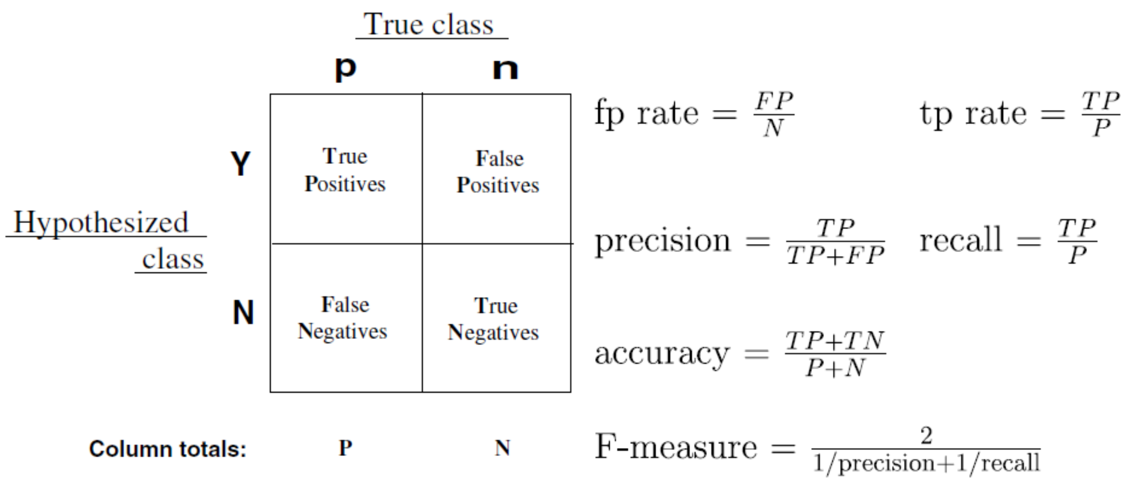


Figure 2.10: Confusion matrix for the generic binary classification problem. From the true classes (p and n) and the predictions (Y and N) the confusion matrix is obtained with the evaluation measures (false positive rate, true positive rate, precision, recall, accuracy, F-measure) [27].

Precision metric intuitively represents the ability of the classifier not to classify as positive a sample belonging to the negative class. Recall represents the classifier's ability to find all positive samples. F-measure can be interpreted as the weighted harmonic mean of the precision and recall metrics [18].

3. State of the art

In order to introduce the topic of classifying modes of transport using machine learning methods, many studies have been used in the research phase. The aim of using other studies was to understand the application of different methods for classifying different modes of transport, with GPS survey data. In order to do so, it is necessary to understand the fundamentals of the machine learning algorithms most commonly used for this type of classification, for which the influence of the different studies is a key factor.

In addition to knowing the most commonly used machine learning algorithms, the studies have been important to know the calculated features of the means of transport, which are the inputs of the model. It also serves as an introduction to the GPS data format and how to work with it. Finally, the results obtained in this study will be compared with those used in the research phase.

Many of the studies address the issue of segmentation (separating segments when there is a change of mode of transport on a route) and classification. Only the topic of classification has been addressed for this study, but these studies have been equally useful.

L. Zhang *et al.* [10] perform a classification in two stages. In the first stage, walk, bike and motorized vehicle modes are determined from the segments identified by speed, acceleration, heading parameters and travel time. In the second stage, a classification of motorized modes of transport (cars, buses, trams, and trains) is carried out using Support Vector Machines (SVMs) method. The study presents an accuracy higher than 94% for all types of classification.

The study carried out by F. Zong *et al.* [28] proposes the identification of modes of transport with Support Vector Classification, using a parameter optimization algorithm known as Genetic Algorithm (GA). The Genetic Algorithm is inspired by evolutionary biology processes like inheritance, selection, crossover and mutation. The classified means of transport are walking, bicycle, subway, bus and car. The success rate in the classification of segments exceeds 80% for all means of transport.

H. Omrani *et al.* [15] presents four machine learning methods (artificial neural net-MLP, artificial neural net-RBF, multinomial logistic regression and support vector machines) for predicting travel modes of individuals in city of Luxembourg. A national survey provides de data set, which contains information on the daily mobility of Luxembourg workers and residents. Classification uses individual characteristics, transport mode specifications, and data related to workplaces and residences. The classification was made for private cars, public transport and soft mode (bikes, walk), and the results show that the artificial networks perform better compared to other alternatives.

Convolutional neural network (CNN) architectures are able to automatically drive high level features from raw input. S. Dabiri and K. Heaslip [1] use CNN schemes to predict the mode of transport of GPS tracks labeled as walk, bike, bus, car and train. In the design of the input layer are fundamental characteristics of movement such as speed, acceleration, jerk and bearing rate. In this study, a variety of CNN configurations have been evaluated to achieve the best CNN architecture. The best results are obtained for a CNN configuration that achieves an accuracy of 84.8%.

Traditional neural networks are generally trained by back-propagation algorithms. G. Xiao *et al.* [29] defend that with this type of algorithms the NN can be trapped in local optimum. That is why Particle Swarm Optimization (PSO) algorithms are used to train NNs. The designed PSO-NNs configurations are used to classify four different means of transport (walk, bike, bus and car) from GPS tracks obtained from smartphone travel surveys. Observing the results obtained for the test set, the best configuration (with PSO-NNs) achieves an accuracy of 94.44 %, improving the performance made with the configuration made with the back propagation algorithm (89.37 %).

Y.J. Byon and S. Liang [9] use neural networks trained with back-propagation algorithms to classify GPS walk, bike, bus and car tracks. Specific patterns to each mode of transport such as speed, number of satellites in view, and electromagnetic levels are detected, from smartphone sensors such as accelerometer and magnetometer and GPS data. The study performs two types of classification, multiple modes of transportation (1) and auto vs non-auto modes (2).

F. Yang *et al.* [5] carry out a study in complex urban environments, developing a method suitable for places with high population density. Develops first a smartphone application for passive GPS data collection and then an algorithm for trip segmentation division is carried out. A neural network module is developed for mode detection on the basis of cell phone GPS location and acceleration data, for a classification between walk, bike, bus and car. The accuracy for detecting all modes was greater than 85 %.

The study carried out by P.A. Gonzalez *et al.* [2] is focused on the possibility of using neural networks to automatically detect the mode of transport from GPS data. For this purpose, an application for GPS-equipped smartphones that can passively determine the mode of transport used with neural networks is developed. The study also analyzes the classification problem using only critical points of the route (in addition to the problem with the entire route), in order to save costs. Depending on the type of problem, the features used in the input layer are different. Some of the features used are average and maximum speeds, standard deviation of distances between stop locations, average dwell time, average and maximum acceleration and total distance. The highest accuracy was 91.23 %, to classify walk, bus and car trips, using only critical points of the route. It was obtained for a learning rate of 0.1 and a training time (number of epochs) of 300.

Y. Zheng *et al.* [6] uses four different models including Decision Tree, Bayesian Net, Support Vector Machine (SVM) and Conditional Random Field (CRF) to detect the mode of transport used in raw GPS data. The approach consists in three different parts: a change point-based segmentation method, an inference model and a post-processing algorithm based on conditional probability, to classify car, bus, walk and bike. The change point-based method achieved a higher degree of accuracy in predicting transportation modes and detecting transitions between them, highlighting the performance of Decision Tree above all other models

The study carried out by C. Carpineti *et al.* [30] focuses on how to obtain the means of transport from the user's experience, based on the smartphone's sensors. Two different techniques are used to carry out transport mode recognition (GPS-based and sensor-based) of walking, still, vehicle (car, train, bus) and bike. The classification was made for four different algorithms (DT, Random Forest (RF), SVM, NN) obtaining the best results for the RF model, reaching an accuracy of 93 %.

S. Reddy *et al.* [31] create a classification system that uses smartphone with GPS receiver and accelerometer. The means of transport classified by the system are stationary, walking, running, biking or in motorized transport. The general classification system consists of a decision tree complemented with a first-order discrete Hidden Markov Model which achieves an accuracy of 93.6 % in the test phase of a dataset obtained from GPS tracks collected by 16 individuals.

L. Stenneth *et al.* [32] propose a method to obtain the mode of transport used by the user from the GPS sensors of the smartphone and from the knowledge of the transport network (real time bus locations, spatial rail and spatial bus information). The method can reach up to a maximum of 93.5 % accuracy for classifying various means of transport such as car, bus, aboveground train, walking, bike, stationary. The accuracy obtained by this method exceeds that obtained by methods using only GPS or using GPS and Geographical Information Systems (GIS). The models that have been experimented with are Bayesian Net, Decision Tree, Random Forest, Naïve Bayesian and Multilayer Perceptron.

4. Methods

4.1. Literature review

The information used to carry out this study has been obtained mainly through the Internet, searching for studies related to the topic. In the beginning, some documents were provided by the Chair of Transport Ecology of the Technical University of Dresden, which were used for the introduction on the topic.

The search for information was conducted through the search engine *Google Scholar* typing key words or titles related to the title of the study, such as ‘transport mode recognition’, ‘machine learning for transport mode classification’, ‘machine learning transport mode GPS’, ‘transportation mode classification smartphone’, combinations of the latter and some more related to the topic.

This search led to different scientific portals such as *ResearchGate*, *Elsevier* or *ScienceDirect* among others. After conducting the search through *Google Scholar*, many of the studies that could be accessed, had free access by *SLUB TU Dresden*, which is the university library and thanks to this has been able to obtain very useful information for this study.

Another technique used was to search for documents based on bibliographies of studies considered interesting. This type of search was useful to find studies or related topics in a specific way. The same technique was used with *Wikipedia*. Through general searches in *Google*, references found in *Wikipedia* articles served to redirect to other studies.

The experimental part has been carried out using the *Jupyter Notebook* platform, through which different packages and libraries such as *NumPy*, *Pandas* or *Scikit-learn* have been used. The platform and its libraries contain a lot of online documentation that has also been very useful and has also served to learn how to develop the algorithms necessary for the application of the experimental part.

Finally, the use of books (in digital format) should also be mentioned, especially to complete the basics part. After the use of documentation, the references have been edited through the program *Mendeley*, which allows to organize the bibliography and to create the desired format for a more efficient work.

4.2. Jupyter Notebook

Jupyter Notebook is an open-source web application that allows the creation and sharing of documents containing live code, equations, visualizations and narrative text. Some uses are data transformation and cleaning, numerical simulation, statistical modeling, data visualization, machine learning and much more. More than 40 programming languages can be used in the notebook. *Python* has been the language used for this study. In the *Jupyter Notebook* environment, some packages with multiple options have been imported in order to work with data and machine learning [33].

- Pandas: *Pandas* is a *Python* package that provides data structures to work in an easy and intuitive way. They are fast, flexible and expressive data structures that allow to work in a practical or professional way with the analysis of real world data through *Python* [34].

- Numpy: "NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more" [35].

- Scikit-learn: *Scikit-learn* is a package that allows to work with machine learning in python. It contains simple and efficient tools for working with data mining and data analysis, being accessible to everyone and reusable in different contexts. It is based on *NumPy*, *SciPy* and *matplotlib*, being open source and commercial use [18].

4.3. Data collection

4.3.1. Study Area

This study focuses on the urban region of Dresden, the capital of the state of Saxony, Germany. The city has an area of 328.31 km^2 and a population of 557'098 inhabitants (2018). GPS tracks have been collected using GPS-equipped smartphones through an application called *Cyface*. The data collection period took place between December 2018 and June 2019. The data was collected both individually by the app and provided by the Institute Chair of Transport Ecology of the Technical University of Dresden. The collected data can be found in the attached disk in the paper version of this study.

The modes of transport chosen for classification were inner-city train (S-Bahn), bus, tram, bicycle and walking. Figure 4.1 shows a general classification of existing means of transport. Highlighted in yellow are the modes of transport for which the classification is carried out, within the general scheme of means of transport.

It is considered relevant to mention an important fact for the study. Dresden is a city where traffic is not too intense, compared to larger cities, where there are many stops due to traffic jams. This causes buses and trams to behave similarly, since they do not make stops caused by traffic, simply by traffic lights or the stops established for passengers. If road traffic was intense, the behavior of both would be different, but it has been observed that in this city in similar due to the arguments above mentioned. The results obtained demonstrate the above, which will be analyzed later.

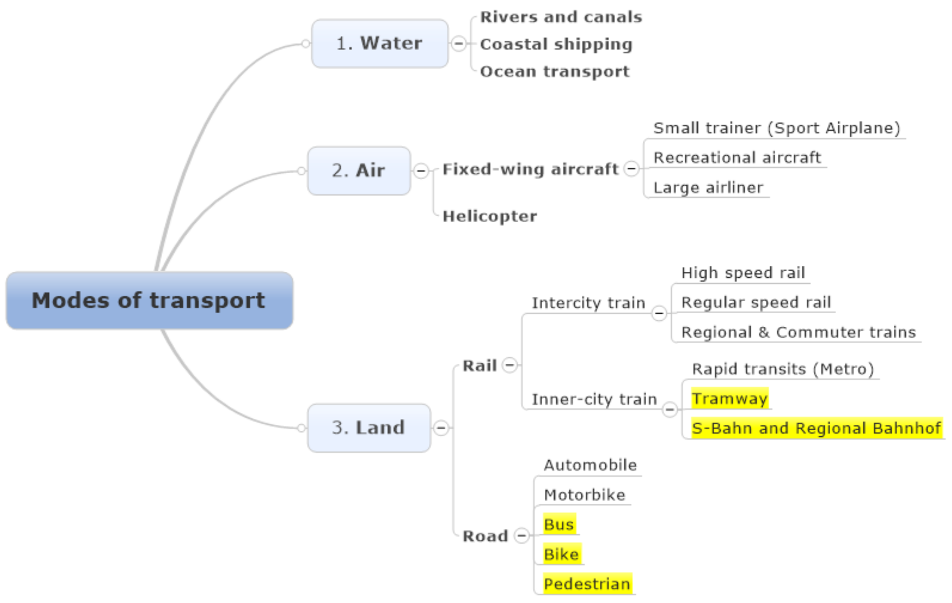


Figure 4.1: Existing means of transport. In yellow, modes of transport selected for analysis in this study

The routes to collect the data were chosen randomly in different time slots (with different traffic conditions) to obtain variability in the data, so that they can faithfully represent the movement of people in the day to day. The routes and time slots of the data provided, which were collected by other people, are unknown, so they also provide variability to the data to have a reliable final representation of the movement of people in the city.

The tracks were manually labeled, an essential step towards the implementation of a supervised machine learning algorithm. After each trip, a note was taken of the mode of transport used to carry out the labeling procedure correctly. According to the diagram in Figure 4.1, Table 4.1 shows the number of tracks collected for each means of transport to carry out the study.

Mode of transport	Means of transport	Number of tracks collected
Road	Pedestrian	106
	Bike	139
	Bus	47
	Car	-
	Motorbike	-
Rail	Tram	76
	Metro	-
	Inner-city Train (S-Bahn)	39
	Intercity Train (IC/EC)	-
	High Speed Rail (ICE)	-
Air	-	-
Water	-	-
Total		407

Table 4.1: Number of tracks collected for study for each mode of transport

4.3.2. Cyface App

The *Cyface App* is a tool for capturing traffic data on roads, bicycle paths or sidewalks. The data can be recorded independently with cars, bicycles or other means of transport During the ride, motion and vibration data are recorded by the sensors built into the smartphone [36].

The *Cyface App* uses the sensors already built into the smartphone. With the help of the accelerometer, gyroscope and GPS sensor, a wide range of data can be collected. During the ride by bicycle or car, vibrations are recorded autonomously (sampling rate of 50-200 Hz.), stored and transmitted to previously defined servers when a WLAN connection is established [36].

The *Cyface App* is very easy to use. The recording of motion and vibration data is conveniently activated or deactivated via the start/stop button. The synchronization of the recorded raw data takes place automatically at the next WLAN connection. For this study, the synchronization function was disabled [36].

Figure 4.2 shows the graphical user interface of the application, with the central button that allows to start and pause data acquisition. It is also possible to see the three buttons that allow the user to define the mode of transport that is being used. This last functionality is not developed, so it does not matter which mode is selected. Data collection is a passive process, the user can lock the mobile during data collection. A notification balloon can be seen on the smartphone indicating that data capture is in progress.

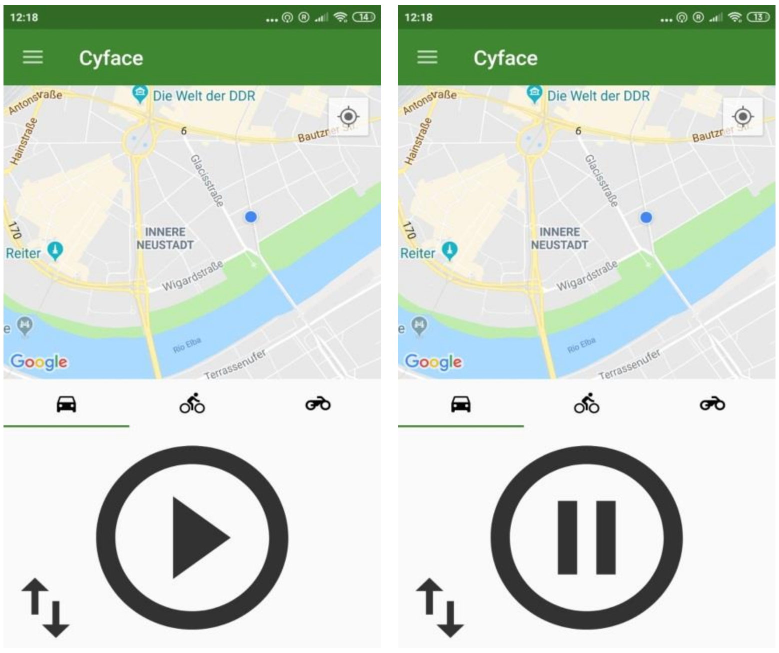


Figure 4.2: Graphical user interface of Cyface App. Main view and travel-mode selection (not depeloved)

The left image in Figure 4.3 shows the different options and settings that can be seen in the graphical interface. It can be seen that the synchronization functionality is disabled. On the right, the measurements section can be observed. Each track is identified by a measurement number. As can be seen in the drop-down list, the tracks are all exported at the same time. In case of any erroneous measurement, it can be selected and deleted before being exported.

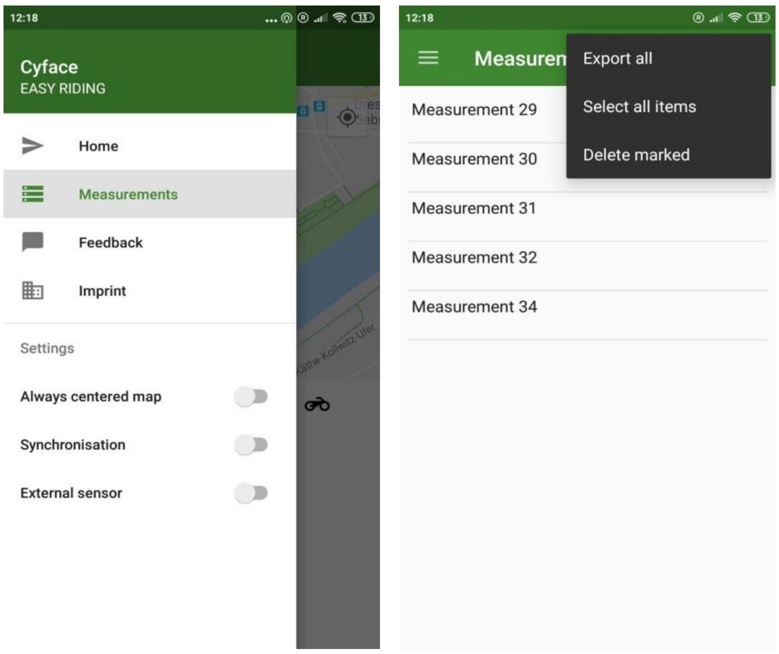


Figure 4.3: Options and settings of Cyface App (left). Measurements section (right)

If the *Cyface App* is used during a ride, it records the distance travelled. By means of GPS signal and time stamp, the journey can be traced exactly. In addition to the route travelled, other important information can also be extracted. As the number of users and the amount of data increases, statements can be made about the volume of traffic and the flow of traffic, and hazardous areas can also be identified.

Latitude, longitude and speed information is available for each GPS point measured. This information is taken every 1 second, corresponding to a frequency of 1 Hz. Each point has an individual id and the measurement number corresponding to the track number to which that point corresponds.

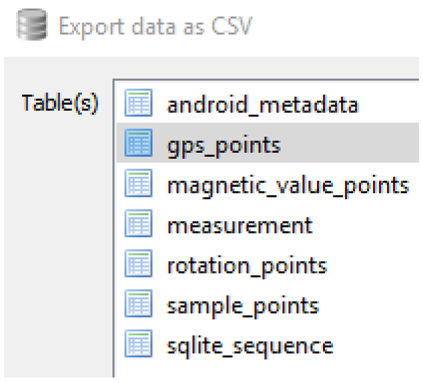
4.4. Data Pre-processing

The final objective of data pre-processing is to calculate and prepare the input variables that will feed the machine learning algorithm. Data collection comes from different sources, so it is important and necessary to design a way to combine all data into a single, cohesive and structured data set. Designing the pre-processing phase effectively makes it possible to quickly and easily add new data to the final set. In order to train a machine learning model, it is a great advantage to design an easy integration of new data, which allows for better training and a more effective model.

4.4.1. Exporting data from the smartphone

The first step is to export the data from the *Cyface* application. The exported tracks, each with its assigned measurement number (Figure 4.3) are contained in a single file. This file is opened in the smartphone by means of the application *SQLite Viewer* that allows to visualize files with extension **.sql*.

Once it is possible to visualize the file, it is exported to the extension **.cvs* through the program *DB Browser for SQLite* already in the computer. From all possible data collected by the *Cyface App*, for this study only GPS data is exported, as can be seen in Figure 4.4.

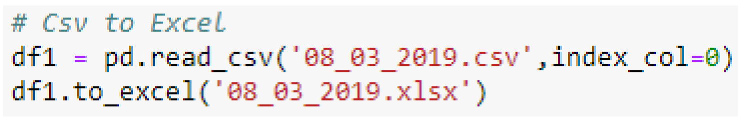


Export data as CSV

Table(s)
- android_metadata
- gps_points
- magnetic_value_points
- measurement
- rotation_points
- sample_points
- sqlite_sequence

*Figure 4.4: Information obtained after exporting tracks from the Cyface application. Only GPS data will be exported to *.csv for this study.*

After exporting the file to extension **.cvs*, it is transformed to **.xlsx*, to be able to visualize the data in the computer in a correct way and carry out the labeling of the tracks. The transformation step to **.xlsx* extension is done easily in *Jupyter Notebook* through the lines of code in Figure 4.5. The document is first to read and then converted to extension **.xlsx* by the function *to_excel()*.



```
# Csv to Excel
df1 = pd.read_csv('08_03_2019.csv',index_col=0)
df1.to_excel('08_03_2019.xlsx')
```

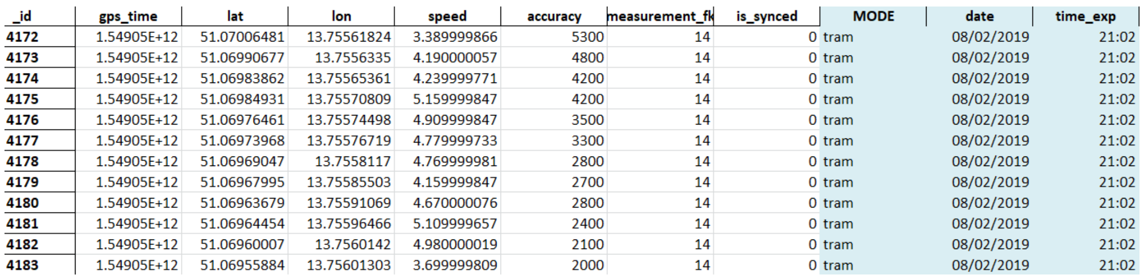
*Figure 4.5: Code to export the file in *.xlsx extension. index_col = 0 allows to place the data from the first box of the document.*

After export, all file extensions are saved named under the date when they were exported. Once the final **.xlsx* file has been exported, the tracks are labeled, a key process in this study. As previously described, each trip is assigned a measurement number. At the time of collecting data, each track was assigned the mode of transport used in that trip. In the data provided by other users also came the information of the mode of transport used. In the process of collecting the information of the means of transport used was noted at the time of beginning the acquisition of data from the particular track to avoid measurement failures.

4.4.2. Data labeling

As previously explained, each track consists of GPS points recorded every second along the trip. Each point is assigned the travel measurement to which it belongs. In the **.xlsx* file, the mode of transport used is written for each GPS point.

Figure 4.6 shows an example of data labeling, where a set of points belonging to a track is taken. Specifically, this is measurement 14, as can be seen in the column *measurement_fk*. For each GPS point information is obtained on timestamp, latitude, longitude, speed and accuracy. The columns in blue show the manual data labeling. The column *MODE* contains the mode of transport used for the track in question, *date* indicates the date on which the document was exported and *time_exp* the time of export.



_id	gps_time	lat	lon	speed	accuracy	measurement_fk	is_synced	MODE	date	time_exp
4172	1.54905E+12	51.07006481	13.75561824	3.389999866	5300	14	0	tram	08/02/2019	21:02
4173	1.54905E+12	51.06990677	13.7556335	4.190000057	4800	14	0	tram	08/02/2019	21:02
4174	1.54905E+12	51.06983862	13.75565361	4.239999771	4200	14	0	tram	08/02/2019	21:02
4175	1.54905E+12	51.06984931	13.75570809	5.159999847	4200	14	0	tram	08/02/2019	21:02
4176	1.54905E+12	51.06976461	13.75574498	4.909999847	3500	14	0	tram	08/02/2019	21:02
4177	1.54905E+12	51.06973968	13.75576719	4.779999733	3300	14	0	tram	08/02/2019	21:02
4178	1.54905E+12	51.06969047	13.7558117	4.769999981	2800	14	0	tram	08/02/2019	21:02
4179	1.54905E+12	51.06967995	13.75585503	4.159999847	2700	14	0	tram	08/02/2019	21:02
4180	1.54905E+12	51.06963679	13.75591069	4.670000076	2800	14	0	tram	08/02/2019	21:02
4181	1.54905E+12	51.06964454	13.75596466	5.109999657	2400	14	0	tram	08/02/2019	21:02
4182	1.54905E+12	51.06960007	13.7560142	4.980000019	2100	14	0	tram	08/02/2019	21:02
4183	1.54905E+12	51.06955884	13.75601303	3.699999809	2000	14	0	tram	08/02/2019	21:02

Figure 4.6: Example of GPS points belonging to a route. Columns in blue show the added information for the points in the labelling process

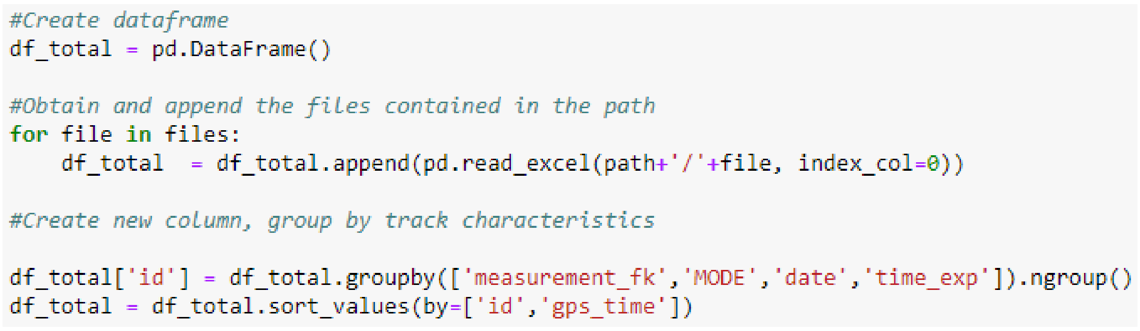
Each track collected by the *Cyface* application is assigned a number. When working with provided data collected by other smartphones this number can match, which is a problem in organizing and structuring the data as the measurement number is the way to identify the trip. This is why the *date* and *time_exp* columns are added, in order to provide more individual track characteristics. The following section will describe how to work with data in the *Jupyter Notebook* environment.

4.4.3. Organizing and structuring data

The first operation is to create a *Pandas DataFrame* through the *Pandas* library. In the first line of code of Figure 4.7 it can be observed how it is created. A *Pandas DataFrame* is the primary *Pandas* data structures. It is defined by the *Pandas* documentation as “Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns)”.

Once the *Pandas DataFrame* has been created, the path where all the labeled data files with **.xlsx* extension are located is chosen and they are appended to the *Pandas DataFrame* by means of the second line of code (Figure 4.7), as seen in the *for* loop. In the unified *Pandas DataFrame*, all the points belonging to each trip are grouped by *measurement_fk*, *MODE*, *date* and *time_exp*, so that each track will be unique even if the measurement number is repeated (*measurement_fk*). The points of each track must have the same four characteristics at the same time, so it is a very efficient method to unify data collected by *Cyface App* from different devices.

The fourth line of code (Figure 4.7) performs the grouping by characteristics. A new column is created in the *Pandas DataFrame*, *id*, which assigns each track its identification number after grouping. Finally, in the last line of the code in Figure 4.7, the values are sorted by *timestamp* and *id*, in such a way as to guarantee the correct order of the GPS points recorded on each track.



```
#Create dataframe
df_total = pd.DataFrame()

#Obtain and append the files contained in the path
for file in files:
    df_total  = df_total.append(pd.read_excel(path+'/'+file, index_col=0))

#Create new column, group by track characteristics

df_total['id'] = df_total.groupby(['measurement_fk','MODE','date','time_exp']).ngroup()
df_total = df_total.sort_values(by=['id','gps_time'])
```

Figure 4.7: Lines of code that create the Pandas DataFrame, append the data to it, group the tracks by individual characteristics, and sort the GPS points on each of them.

4.4.4. Calculation of features

Once all the data have been unified in the *Pandas DataFrame* in an organized way, the different possible features are calculated. In the first instance, the calculation of potential variables is carried out, based on other studies and on the experience of the engineers of the Chair of Transport Ecology of the Technical University of Dresden. In subsequent steps, the analysis and selection of the most appropriate variables for a better performance of the model will be conducted.

Once the features have been calculated, it is interesting to observe the graphical representation of the track value distributions for each means of transport. This is useful to have a first estimation of how the different classes are better classified (observing the graph), from a certain feature.

In order to structure the data it is more efficient to follow the procedure in this way: 1) Unify data in *Pandas DataFrame*, 2) Calculate potential features, 3) Analysis and selection of features, 4) Feed the features to the model. This section describes which features and how the features have been calculated from the unified *Pandas DataFrame*.

4.4.4.1. Distance between consecutive points and total distance of the track

From the longitude and latitude of each point, it is possible to calculate the distance between two consecutive points within a track. The sum of all distances will therefore be the total distance of the track in question. This distance is calculated from the haversine function, explained by T. Feng and H. J. P. Timmermans [37] as follows:

Haversine's formula calculates the shortest distance between two points on the Earth's surface. Figure 4.8 and the following equations show how the calculation is made.

$$haversine(c) = haversine(a-b) + \sin(a) + \sin(a)\cdot\sin(b)\cdot haversine \quad (4.15)$$

$$d = R\cdot haversine^{-1}(h) = 2R\cdot\arcsin(\sqrt{h}) \quad (4.16)$$

$$haversine(\theta) = sin^2\left(\frac{\theta}{2}\right) = (1-\cos(\theta))/2 \quad (4.17)$$

$$h = haversine(\varphi_2 - \varphi_1) + \cos(\varphi_2)\, haversine(\Delta\lambda) \quad (4.18)$$

where haversine is the haversine function; d is the distance between the two consecutive points; R is the radius of the earth (fixed at 6371 km); φ_1 and φ_2 latitudes of point 1 and 2 respectively and $\Delta\lambda$ is the longitude separation. Combining the Equation 4.16 with the Equation 4.17, the following equation is obtained:

$$h = sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cos(\varphi_2)\, sin^2(\Delta\lambda/2) \quad (4.19)$$

where $\Delta\varphi$ is the altitude separation.

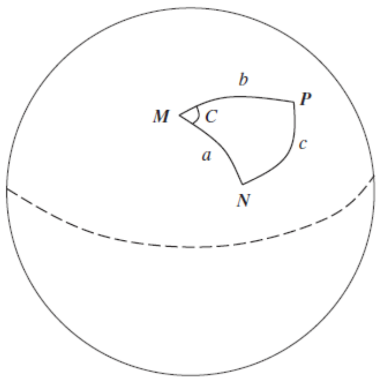
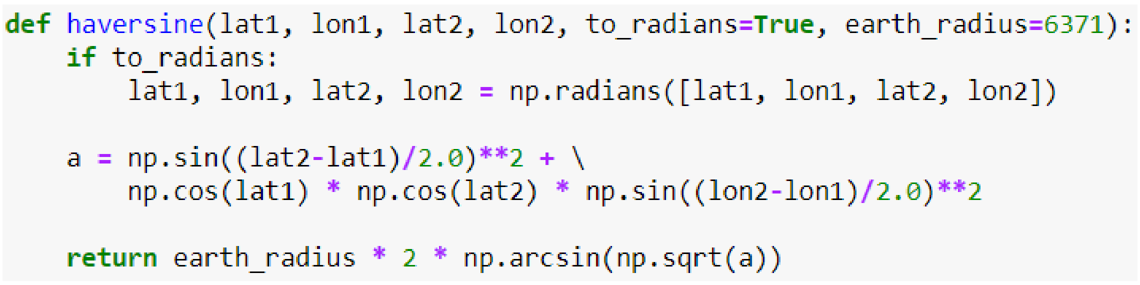


Figure 4.8: The law of the Haversines [37].

For the calculation in *Jupyter Notebook*, a function is created that will be applied between consecutive points. Figure 4.9 shows the performed code. As a result, the function returns the distance *d* between points, as stated in Equation 4.16. As can be seen in the code, calculations with angles are performed in radians, the value of the earth's radius is fixed at 6371 km, *lat1*, *lon1* belong to the first point and *lat2*, *lon2* to the consecutive point.



```
def haversine(lat1, lon1, lat2, lon2, to_radians=True, earth_radius=6371):
    if to_radians:
        lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    a = np.sin((lat2-lat1)/2.0)**2 + \
        np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return earth_radius * 2 * np.arcsin(np.sqrt(a))
```

Figure 4.9: Code of the haversine function applied in Jupyter Notebook

Having the distances between consecutive points of a track, it is easy to calculate the total distance travelled, with the total sum of all distances. Calculating the distances between consecutive points is very useful to later obtain other features, such as acceleration. In addition, the total distance can be considered as feature, the total distance of each track varies independently of the mode of transport used. In theory, S-Bahn, bus and tram travel greater distances, but there may also be tracks collected for small distances using these means.

4.4.4.2. Ratio of distances

The previous point explains how to calculate the distance travelled. The following explains how to calculate the ratio of distances, which is the relationship between the total distance travelled and the minimum distance considering the first and the last point of the trajectory, that is, the straight line that joins these two points. This minimum distance is calculated by applying the haversine formula, as seen in the previous point. Instead of between two consecutive points, now it is done for the first and last point of the trajectory.

As can be seen in Figure 4.10, the ratio is lower for tram and S-Bahn transport (closer to 1) which means that their trajectories have a more rectilinear character, particularly for the S-Bahn case.

For the rest of the modes of transport, it can be observed that the distributions reach higher values, especially for the cases of bicycle and walk, whose trajectories are less straight and with more variations in turns.

As can be seen in figure 4.10, this feature is useful for differentiating classes, especially for bus and tram, the most complex classification.

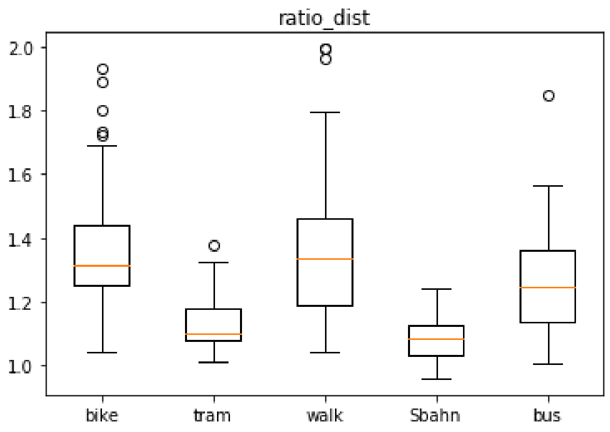


Figure 4.10: Boxplot of ratio of distances.

4.4.4.3. Bearing factor

The rate of change in the heading direction of the different modes of transport varies. Trams and buses follow their direction from the established tracks and streets, while walking or cycling this direction can vary much more. As can be seen in Figure 4.11, bearing measures the angle by means of the line that joins two consecutive points and a reference, such as the magnetic direction of north or south [1].

The bearing ratio differs from the bearing between two consecutive points. Starting from three consecutive points, the bearing between points 1 and 2 and the bearing between points 2 and 3 can be calculated. The bearing rate is calculated as the difference in absolute value of the two calculated bearing values. In Figure 4.11 the bearing between points can be seen graphically.

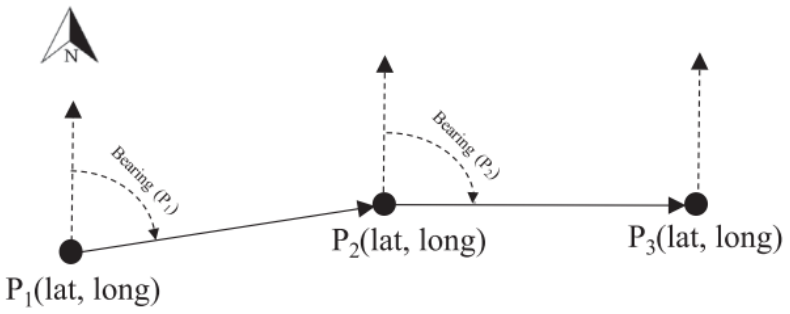


Figure 4.11: Bearing between consecutive GPS points. The ratio bearing would be calculated as $BR = |Bearing_{P2} - Bearing_{P1}|$ *[1].*

Figure 4.12 shows the function used in *Jupyter Notebook* to calculate bearing between two consecutive points. *lat1*, *lon1* belong to the first point and *lat2*, *lon2* to the consecutive point. The function returns the bearing angle in degrees. If a negative value is obtained, 360 degrees are added to work with positive values.

```
def get_bearing(lat1,lon1,lat2,lon2):
    dLon = lon2 - lon1;
    y = math.sin(dLon) * math.cos(lat2);
    x = math.cos(lat1)*math.sin(lat2) - math.sin(lat1)*math.cos(lat2)*math.cos(dLon);
    brng = np.rad2deg(math.atan2(y, x));
    if brng < 0: brng+= 360
    return brng
```

Figure 4.12: Code of the function of the algorithm that calculates the bearing. The result is the (positive) value of the bearing between two points in degrees.

Figure 4.13 shows the bearing and bearing rate graphs. The boxplot represents the distribution of average values for each trip. In general, both graphs show lower values for S-Bahn and tram cases, since, as mentioned above, the trajectory is straighter. In the case of the bearing, it is observed in the boxplot that the values of the distributions are in wide ranges of values, so it does not seem very effective to classify. In the case of the bearing rate, the value ranges are similar, especially comparing the cases of bike, tram and bus. To differentiate S-Bahn and walk from the rest of the classes it is presented as a good feature.

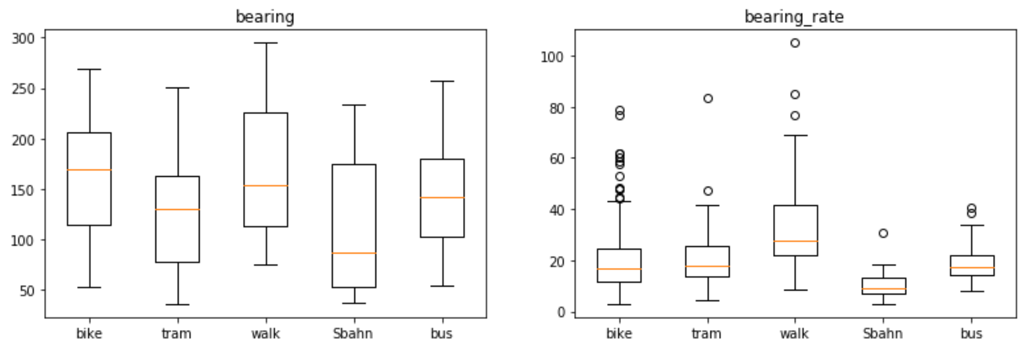


Figure 4.13: Boxplot graphics for bearing and bearing rate.

4.4.4.4. Total time

The timestamp in the data acquisition is one second. Every second GPS information is obtained from a point on the trajectory. The time elapsed between two consecutive points of the trajectory is one second, so simply adding the times between the consecutive points gives the total time of the trajectory. The time between two points as well as the total time are useful magnitudes for calculating other features such as the acceleration.

4.4.4.5. Acceleration

From the distance between two consecutive points and the speed measured at each point by GPS, acceleration is obtained. Two types of acceleration have been considered, with and without absolute values. The acceleration without absolute values has been calculated to take into account the effect of deceleration, since in the case of S-Bahn, bus and tram it is important to consider it due to the stops they make, so the braking effect is considerable.

4.4.4.6. Ratio of stops

In general, public transport makes more stops than a person riding a bicycle or walking on a particular route. The bus and tram make stops every distance, so it is interesting to calculate the number of times the means of transport makes a stop on a track. For this study, a ratio has been calculated which estimates the number of times the means of transport stops on a track. This has been calculated from Equation 4.20.

$$ratio_{stops} = \frac{(number\ of\ GPS\ points\ with\ zero\ speed)_{TRACK}}{(number\ of\ total\ GPS\ points)_{TRACK}} \tag{4.20}$$

Figure 4.14 shows the graphical representation of the values obtained for the ratio of the different modes of transport. As expected, the ratio is higher for S-Bahn, bus and tram as they make more stops. As can be seen in the boxplots, it could be a good differentiation criterion to classify the different modes, especially bike and walk from the rest.

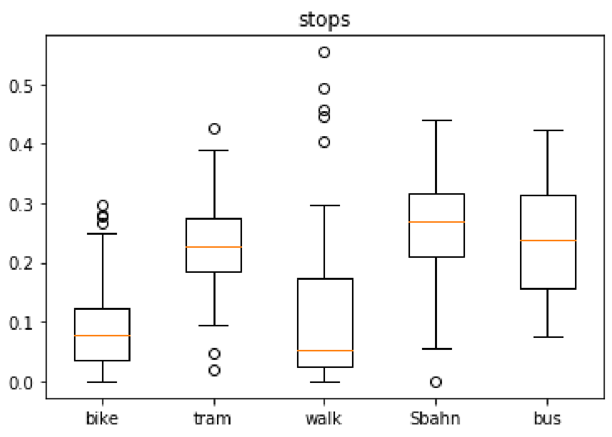


Figure 4.14: Boxplots ratio of stops.

4.4.4.7. Maximum, mean, percentiles and standard deviation of velocity and acceleration values

Maximum values: it is interesting to calculate them. In the case of speed, the S-Bahn, bus and tram reach maximum speeds higher than the rest (particularly S-Bahn). On a bicycle, the maximum speed will always be greater than the speed of walking. The problem arises when there are GPS signal failures, as there could be high measured values that do not correspond to the actual measurement. As can be seen in figure 4.15, in the case of acceleration, both in absolute and non-absolute values, the distribution of data is similar in boxplots. Distribution values of bike, tram and bus present similar ranges. At first glance it seems that the maximum speed has a greater differentiating character of the modes. The bus and tram values are very similar but otherwise it is presented as a very effective feature.

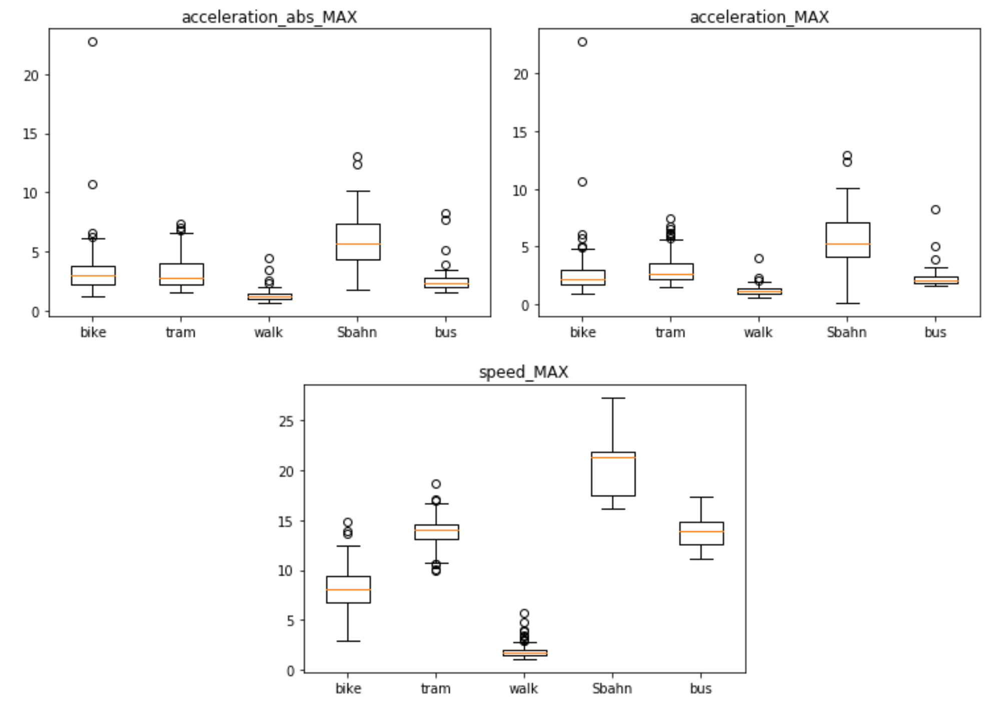


Figure 4.15: Boxplots of acceleration, absolute acceleration and maximum speed for the tracks of the different means of transport.

Mean values: Calculating the average values of the tracks seems to be a good option to distinguish the modes of transport, in the case of absolute acceleration and speed, where it is observed that for each class the values are in different ranges (except for bus and tram that, once again, the values are similar). In the case of acceleration with non-absolute values, the mean values of the tracks are around zero, due to the compensation of positive and negative values. Everything described can be seen in Figure 4.16.

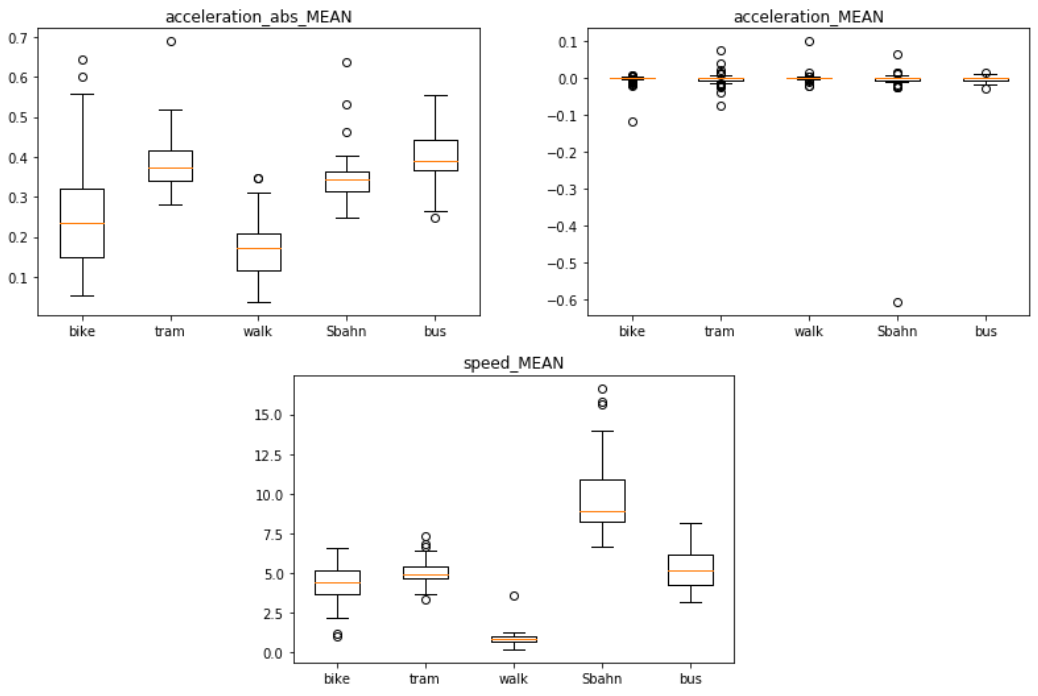


Figure 4.16: Boxplots of acceleration, absolute acceleration and mean speed for the tracks of the different means of transport.

Standard deviation values: the standard deviation of the values of the three parameters has been calculated for each track and compared (Figure 4.17). The standard deviation could act as a differentiating attribute of the classes. Particularly noteworthy is the case of the standard deviation of velocity, with a greater difference in values between the different classes. For all three cases, the values for tram and bus are in very similar ranges.

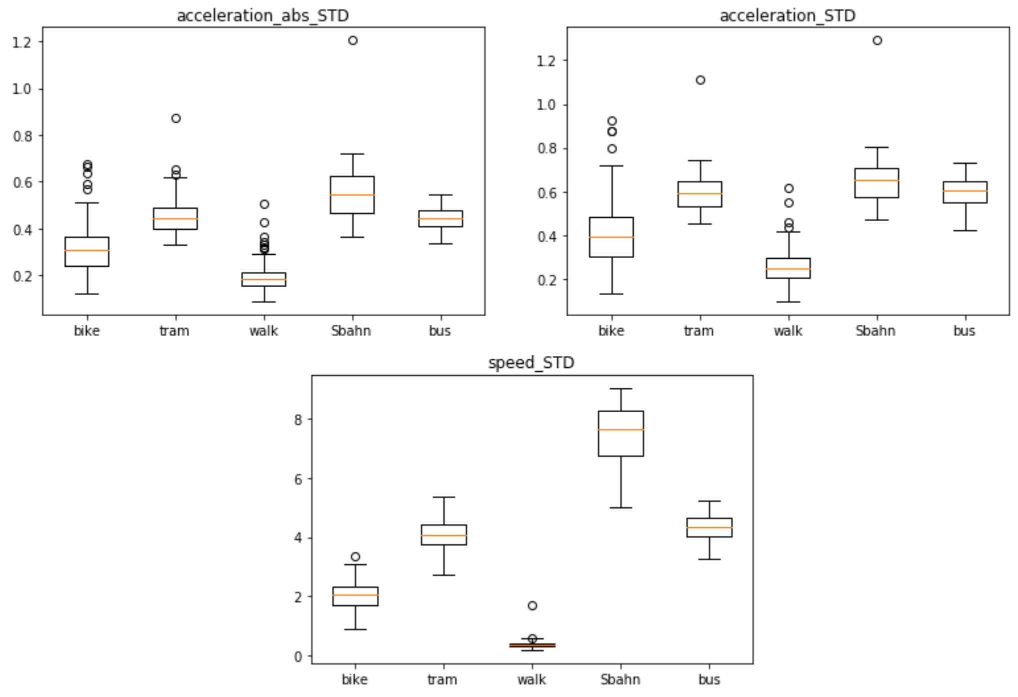


Figure 4.17: Boxplots of acceleration, absolute acceleration and standard deviation speed for the tracks of the different means of transport.

Percentiles: according to other studies such as F.Zong *et al.* [28] or G. Xiao *et al*. [29] as well as the experience of the engineers of the Chair of Transport Ecology of the Technical University of Dresden, it has been considered appropriate to calculate the percentiles of the acceleration and speed variables for the tracks. Percentiles have therefore been calculated from 10 % to 90 % (10 by 10) and plotted using boxplots. In this way, it is possible to analyze which percentiles are the most appropriate to carry out the classification. The graphs calculated for the 20th, 40th, 60th and 80th percentiles, for absolute and non-absolute acceleration variables and speed are shown as an example.

In Figure 4.18, for the representation of the percentiles of absolute acceleration, it is observed how the 60th and 80th percentiles seem more appropriate to carry out the division of classes. The 20th and 40th percentiles move in similar ranges for all the different classes. Representing the percentiles for the acceleration variable without absolute values (Figure 4.19) there are some differences. The 20th percentile case stands out, whose distribution of values (negative due to deceleration) distinguishes the classes better than for the previous case. The 80th percentile case seems more appropriate than the 40th and 60th percentiles, which are useless for the classification. Finally, the case of speed is similar to the case of acceleration with absolute values. As shown in Figure 4.20, the 60th and 80th percentiles are again more suitable for classification.

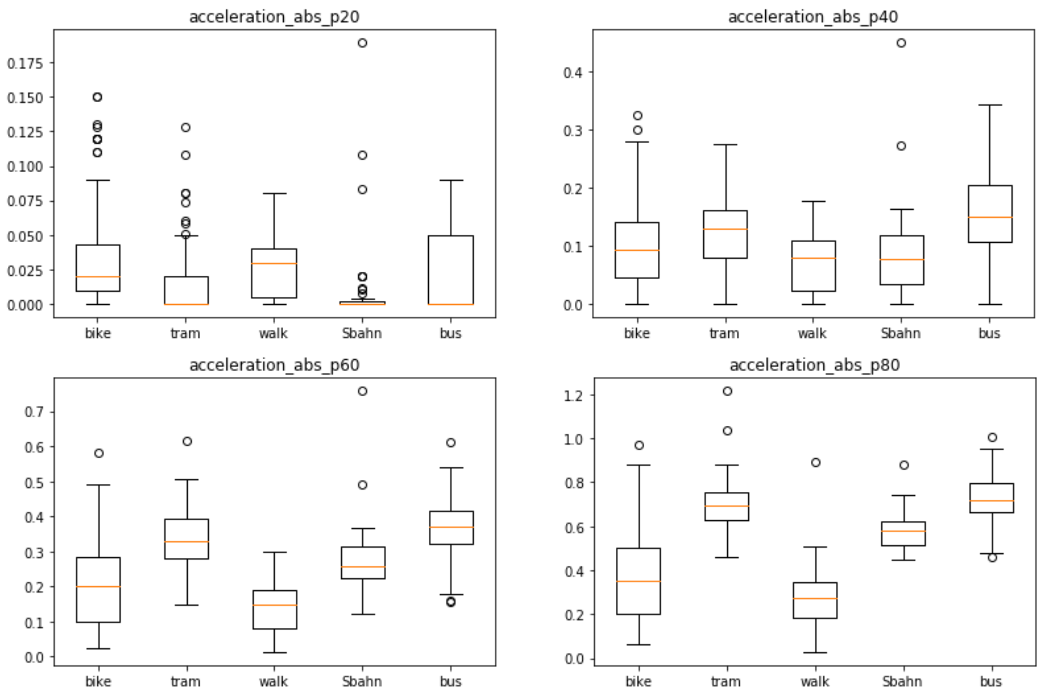


Figure 4.18: Representation of the boxplot for the 20th, 40th, 60th and 80th percentiles of absolute acceleration.

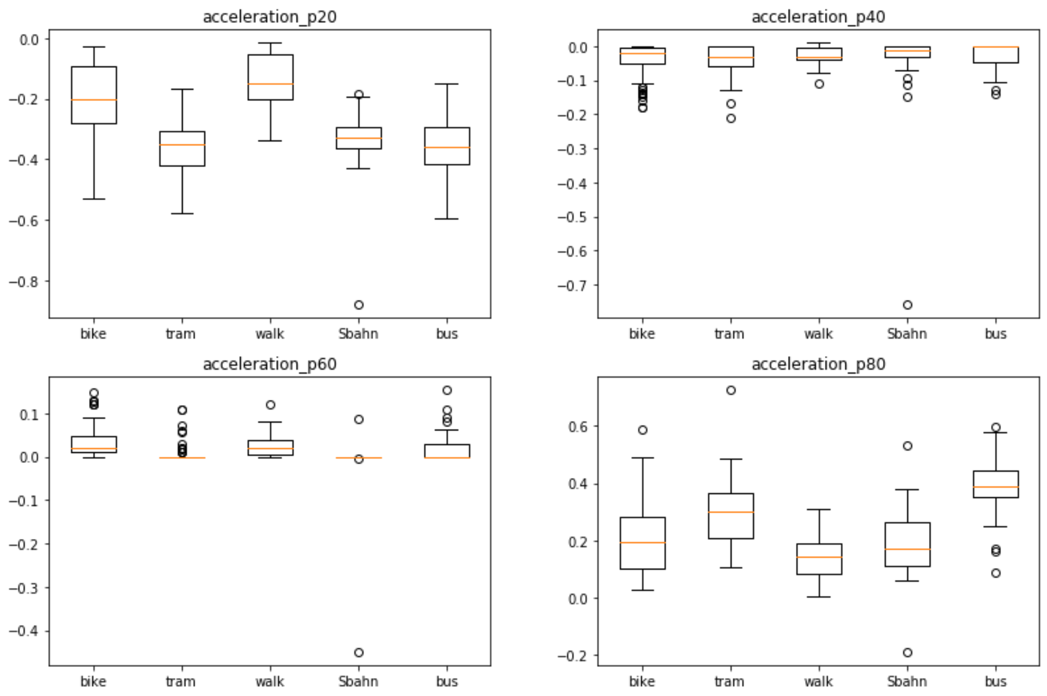


Figure 4.19: Representation of the boxplot for the 20th, 40th, 60th and 80th percentiles acceleration.

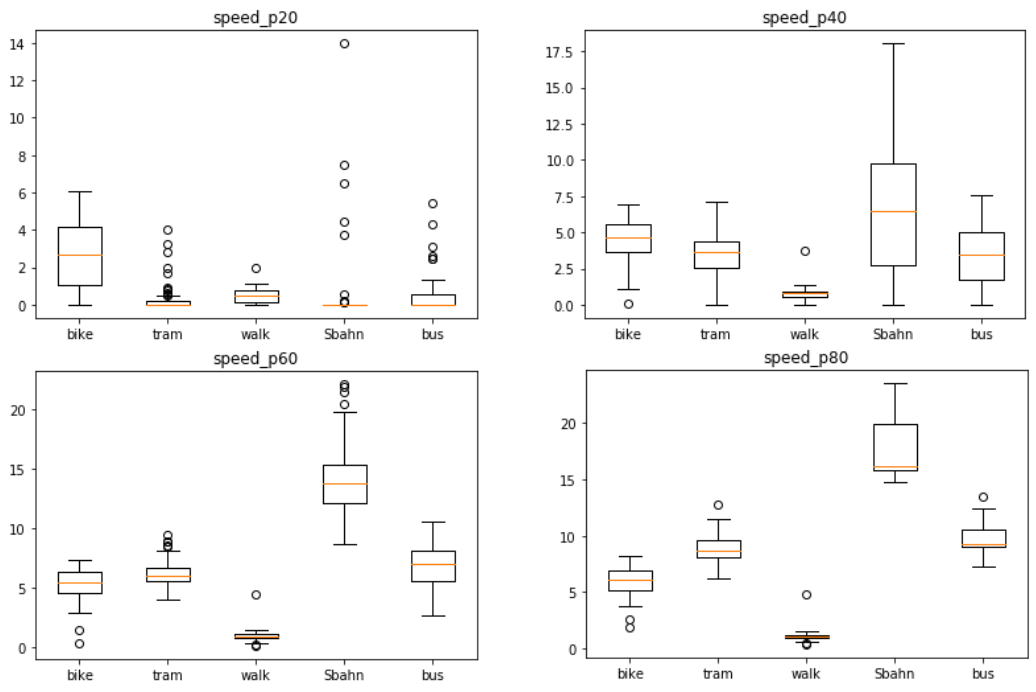


Figure 4.20: Representation of the boxplot for the 20th, 40th, 60th and 80th percentiles of speed.

Finally, it is interesting to visualize the representation of the speed percentiles in a line plot (Figure 4.21). As can be seen, the value of percentiles higher than 75 % is very characteristic of each mode of transport, particularly for the S-Bahn (red line), walk (green line) and bike (blue line) cases. In addition, for the percentile values greater than 50 %, it is very easy to distinguish between walk (green line) and S-Bahn (red line).

It is difficult to distinguish the values of bus (purple line) and tram (orange line) because, as can be seen in the figure, they follow a very similar trend. As previously mentioned, the behavior of the bus and tram in the city of Dresden is very similar, which explains why in the percentile values there is not much difference, as happens with a large number of features for these modes, as described above.

The area of percentiles below 25 % is interesting for distinguishing the bicycle (blue line) from other modes, while S-Bahn, tram, bus and walking (green line) values are very similar.

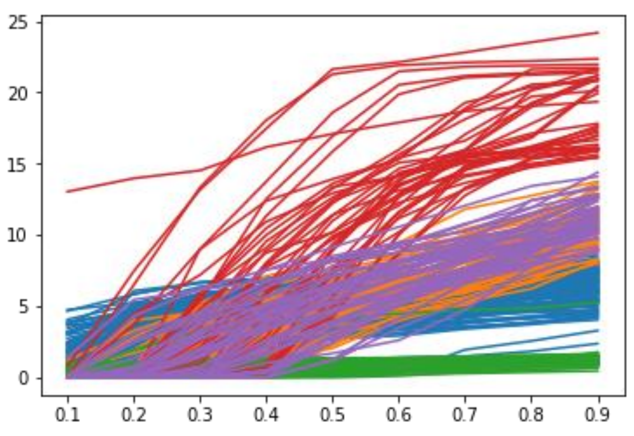


Figure 4.21: Line plot of the speed percentiles for the tracks measured: S-Bahn (red line), tram (orange line), bus (purple line), bike (blue line), walk (green line).

4.5. Data Processing

4.5.1. Cascading classifiers

In order to carry out classification, a cascading process was chosen. This process is performed through different phases with different classifiers. As can be seen in Figure 4.22, four classifiers have been designed for a classification of five modes of transport. The diagram shows how the phases take place and the means of transport that are classified in each of them. There are several reasons why it was decided to use this classification method:

1) Allows to order and structure the problem of the study from least to most difficult when classifying modes of transport.

2) It is interesting to study which is the most effective machine learning algorithm according to the means of transport to be classified, and this design allows a more detailed analysis when selecting the algorithm to carry out the classification.

3) Classifying S-Bahn, walk, and bike (individually) with respect to other means of transport is not a great difficulty, due to their different characteristics. This method allows focussing (in the last classifier) the problem in the distinction between bus and tram, after classifying the rest of modes of transport. In this way, it is possible to study in detail the best conditions and parameters to carry out this last classification, which entails the greatest difficulty of the problem.

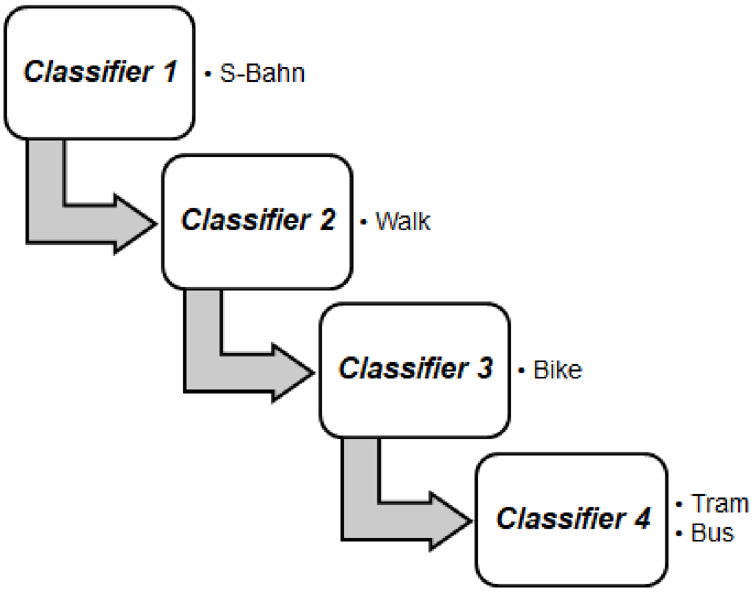


Figure 4.22: Diagram showing the process performed for a cascading classification. The classifiers 1, 2, 3 and 4 are observed in which the classification of S-Bahn, walk, bike and tram and bus respectively is carried out.

It is important to highlight the transition between phases, where there are two possible situations, explained with the following examples:

1) After performing the classification of S-Bahn against the rest of modes of transport (phase 1), all tracks that are not of S-Bahn move to phase 2. For all other transitions, the process is carried out in the same way.

2) After performing the classification of S-Bahn against the rest of modes of transport (phase 1), all tracks that the model has predicted as not S-Bahn move to phase 2. For all other transitions, the process is carried out in the same way.

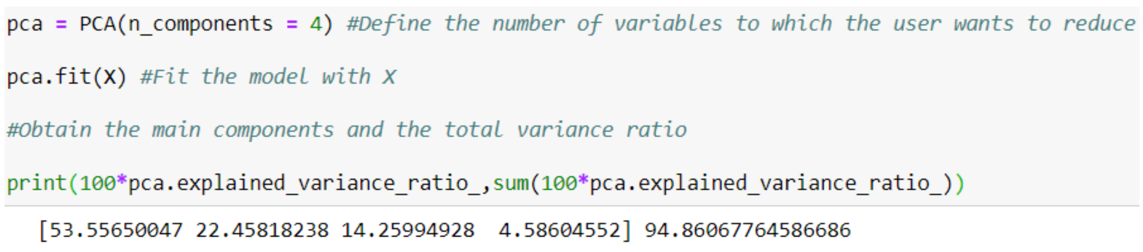
For this study, the second option has been chosen, since it is more realistic, as the classifier can erroneously predict. The first option contemplates a perfect classification in each phase and does not accumulate the error between phases, being, therefore, less realistic.

4.5.2. Principal Component Analysis (PCA)

After standardizing the data set, it is considered in this study to apply the dimensional reduction of the features calculated by means of the principal component analysis. For this purpose, a linear reduction of the dimension of the data is carried out using Singular Value Decomposition (SVD), in order to project them in a lower dimensional space [18]. As mentioned above, the case will be studied applying PCA and not applying PCA. For the first case, the code in figure 4.23 has been developed in *Jupyter Notebook*. The required functions have been imported from the *Scikit-learn* library.

The code in Figure 4.23 shows an example of a reduction using the PCA method. In the first line, the user defines the number of variables to which the dimension is to be reduced (4, in this case). Through the function in the second line, the method is applied to the variables stored in *X*. Finally, it is asked to display the value of the main components and their total sum, which represents the ratio of total variance explained by the variables, in other words, the amount of information that is retained with respect to the original variables after applying the dimension reduction.

For this example, a total variance ratio of 95 % is observed, which means a very effective reduction process since there is very little loss of information in the dimension reduction. In the following sections the final reduction applied will be evaluated and compared with the cases where the PCA process is not applied.



```
pca = PCA(n_components = 4) #Define the number of variables to which the user wants to reduce

pca.fit(X) #Fit the model with X

#Obtain the main components and the total variance ratio

print(100*pca.explained_variance_ratio_,sum(100*pca.explained_variance_ratio_))
```

```
   [53.55650047 22.45818238 14.25994928  4.58604552] 94.86067764586686
```

Figure 4.23: Code where the PCA method is applied. The number of variables to be reduced is defined, the model is applied to the variable X, which contains the feature values and the total variance ratio is obtained.

A balance must be found between the number of variables to be reduced and the percentage of total variance obtained. The process is effective when for the lowest number of variables there is a percentage of variance for which there is little loss of information. The most important factor is that the new variables contain a high percentage of the information from the originals. That is why a reduced number of variables should not be chosen if this entails a percentage of variance in which there is a significant loss of information. Once the number of variables to be reduced is calculated, it can be applied to the cascading classifiers.

4.5.3. Stratified Shuffle Split

For model validation and training, the data set is split into train and test set. In the *Jupyter Notebook* algorithm, the splitting has been carried out using the *Stratified Shuffle Split cross-validator* function, implemented from the *Scikit-learn* library.

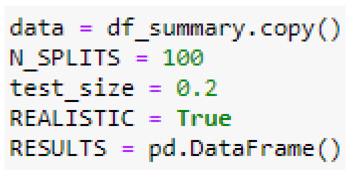
This cross-validation object is a mixture of *StratfiedKFold* and *ShuffleSplit*, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class. It should be mentioned that (like the *ShuffleSplit* strategy) stratified random divisions do not guarantee that all folds are different, although this is very likely in the case of large data sets [18]. In each iteration in which the division is carried out, the percentage of the existing classes of the total data set is maintained, there will be the same percentage of tracks for each mode of transport.

Figure 4.24 shows the code of the algorithm used in *Jupyter Notebook*. In the second line is defined the number of times to perform the process of division, for this case, have been defined 100. In the third line, the percentage of the data set that is used for the test set is defined. The remaining percentage will be used for the train set. In this case, 20 % has been selected for the test set. In this way, the number of iterations to be performed and the size of the test and train sets are defined.

4.5.4. Classifier Algorithm

Once the tracks have been grouped, the necessary variables are calculated to obtain the features of each track. The final result of the process is a *Pandas DataFrame* called *df_summary* that contains in each line each track with the calculated features. This *Pandas DataFrame* is converted to *array* type, storing the data in a variable called *measures*, since this format is necessary to apply the normalization and machine learning models.

This section explains the steps followed in each phase of the classification. First, the parameters observed in Figure 4.24 are defined. The realistic method is applied using the command *True* and *df_summary* is copied into the variable *data* (Figure 4.24).



```
data = df_summary.copy()
N_SPLITS = 100
test_size = 0.2
REALISTIC = True
RESULTS = pd.DataFrame()
```

Figure 4.24: Code lines showing the parameters set before the cascading classifiers.

As can be seen in the Figure 4.25, each executed line has a previous explanation. The code in the image is explained from top to bottom. First of all, it is important to assign a particular name to the case being processed, since there may be several different cases depending on the parameters used. In the case of the Figure 4.25, a Decision Tree Learner has been chosen. Then select the mode of transport on which the classification is to be made (in this case *walk*). In the variable *Y* the existing classes are calculated, in this case *walk* (that will take the value 1) and the rest of present modes (that will take the value 0). The values of the calculated features are stored in variable *X*, as an array (required to apply the following methods). Only the numeric values of the variable *measures* are copied, grouped by tracks as in *data* where the variable *df_summary* was copied.

As mentioned in section 2.6, standardization of the data set is necessary to apply machine learning models. In the code of figure 4.25 two types of scaler appear, the *StandardScaler* (in Figure 4.25, *scaler_1*) and the *MinMaxScaler* (in Figure 4.25, *scaler_2*). These two scaler types are described in section 2.6. The *StandardScaler* applies a distribution with a mean of zero and a unit of variance, while the *MinMaxScaler* scales the values between -1 and 1. In this study the *StandardScaler* has been used in each of the following phases. After standardizing the data, the scaler is saved by means of the function *pickle.dump()*.

As can be seen in the code in the Figure 4.25, after normalizing the data, the PCA can be applied for the number of variables calculated beforehand, as explained in the previous section. After defining the number of splits (100) and the test size (20%), the *Stratified Shuffle Split* function is applied through the last line of the code in the Figure 4.25.

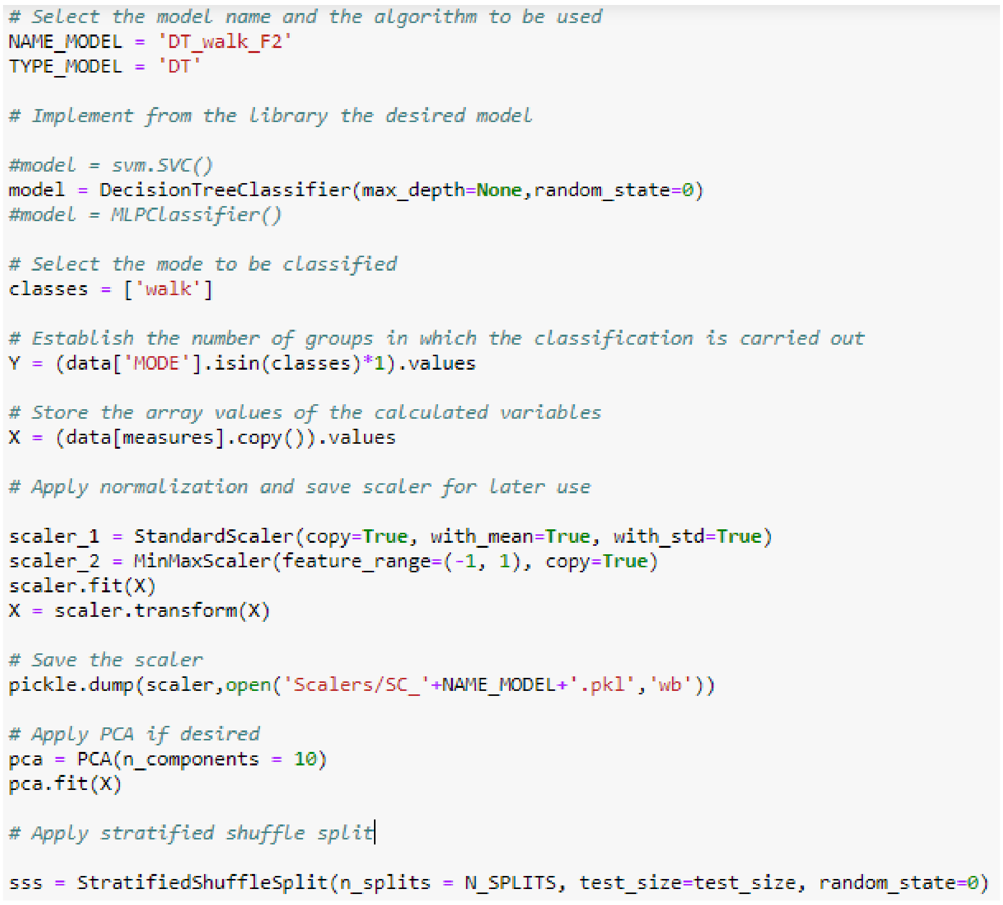
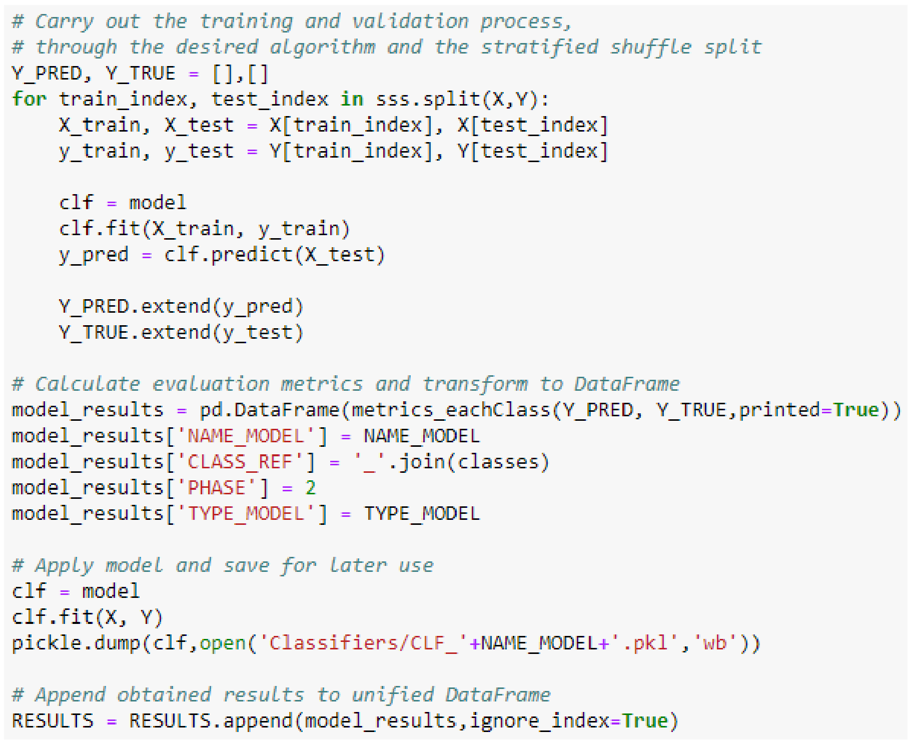

```
# Select the model name and the algorithm to be used
NAME_MODEL = 'DT_walk_F2'
TYPE_MODEL = 'DT'

# Implement from the library the desired model

#model = svm.SVC()
model = DecisionTreeClassifier(max_depth=None,random_state=0)
#model = MLPClassifier()

# Select the mode to be classified
classes = ['walk']

# Establish the number of groups in which the classification is carried out
Y = (data['MODE'].isin(classes)*1).values

# Store the array values of the calculated variables
X = (data[measures].copy()).values

# Apply normalization and save scaler for later use

scaler_1 = StandardScaler(copy=True, with_mean=True, with_std=True)
scaler_2 = MinMaxScaler(feature_range=(-1, 1), copy=True)
scaler.fit(X)
X = scaler.transform(X)

# Save the scaler
pickle.dump(scaler,open('Scalers/SC_'+NAME_MODEL+'.pkl','wb'))

# Apply PCA if desired
pca = PCA(n_components = 10)
pca.fit(X)

# Apply stratified shuffle split

sss = StratifiedShuffleSplit(n_splits = N_SPLITS, test_size=test_size, random_state=0)
```

Figure 4.25: Lines of the code used for the classifiers (Part 1). In this case a DT is applied for phase

2 (classifier 2), where the walk mode is classified against the rest.

Figure 4.26 shows how the process continues. After calling the *Stratified Shuffle Split* function, the training and validation process is applied for the selected algorithm, which in this case is a DT. The evaluation metrics are calculated and the data corresponding to the case is transformed to *Pandas DataFrame* format to finally attach them to the final results table, once the model is applied to the *X* and *Y* variables. The final model is saved by the function *pickle.dump()* as it was done with the scaler.

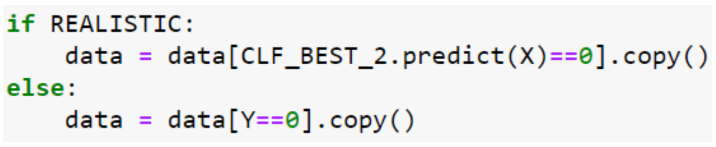


```
# Carry out the training and validation process,
# through the desired algorithm and the stratified shuffle split
Y_PRED, Y_TRUE = [],[]
for train_index, test_index in sss.split(X,Y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = Y[train_index], Y[test_index]

    clf = model
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    Y_PRED.extend(y_pred)
    Y_TRUE.extend(y_test)

# Calculate evaluation metrics and transform to DataFrame
model_results = pd.DataFrame(metrics_eachClass(Y_PRED, Y_TRUE,printed=True))
model_results['NAME_MODEL'] = NAME_MODEL
model_results['CLASS_REF'] = '_'.join(classes)
model_results['PHASE'] = 2
model_results['TYPE_MODEL'] = TYPE_MODEL

# Apply model and save for later use
clf = model
clf.fit(X, Y)
pickle.dump(clf,open('Classifiers/CLF_'+NAME_MODEL+'.pkl','wb'))

# Append obtained results to unified DataFrame
RESULTS = RESULTS.append(model_results,ignore_index=True)
```

Figure 4.26: Lines of the code used for the classifiers (Part 2). In this case a DT is applied for phase 2, where the walk mode is classified against the rest.

Figure 4.27 shows the code used for the transition between classifiers. After classification, the next classifier is passed through the transition phase. In this case, the values that in this phase have not been predicted as *walk*, will pass to the next phase. In this way the work is done taking into account the possibility of failure of the classifier, being the model more realistic.

As can be seen in the code in Figure 4.27, if the model selected is of the *REALISTIC* type, the *data* variable stores the data that the model has predicted as not *walk*, accumulating the possible prediction error. If the *REALISTIC* model has not been selected, all the tracks that do not belong to the *walk* class, go on to the next classifier.



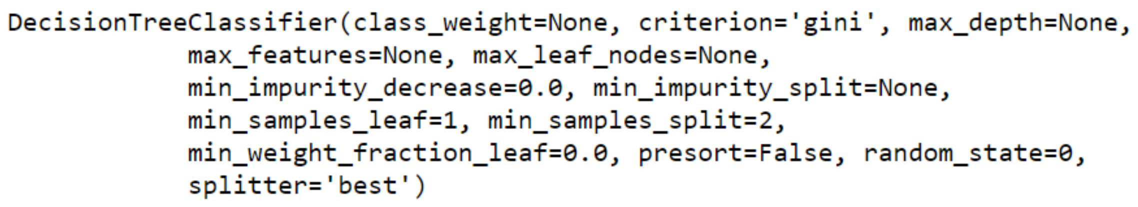
```
if REALISTIC:
    data = data[CLF_BEST_2.predict(X)==0].copy()
else:
    data = data[Y==0].copy()
```

Figure 4.27: Code for the transition between phases, in this case, between classifier 2 (walk) and classifier 3 (bike), after performing the classification of walk.

4.5.5. Machine Learning models

For each classifier, three machine learning algorithms have been used, all of them implemented from the *Scikit-learn* library. The classification algorithms, as previously described, are Decision Tree Learner, Support Vector Machine and Neural Networks (Multi-layer Perceptron). The objective is to carry out the classification in each of the four phases and to compare which is the machine learning algorithm with which the best results are obtained. The parameters set by default by the library are used. In some cases modifications have been made to try to optimize the model.

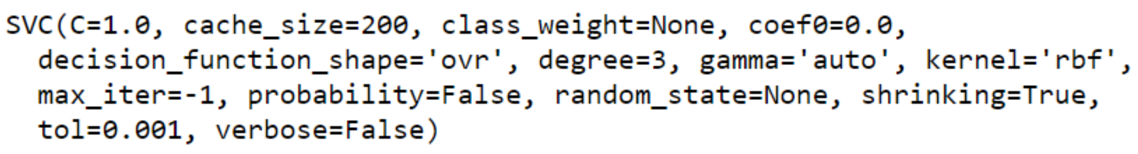
The following figure (Figure 4.28) shows the algorithm used in this study with the parameters selected for the case of Decision Tree Learner (default *Scikit-learn* library). A more detailed description of these parameters can be found in the Annex A.



```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

Figure 4.28: Decision Tree Learner algorithm implemented in the model with the parameters used.

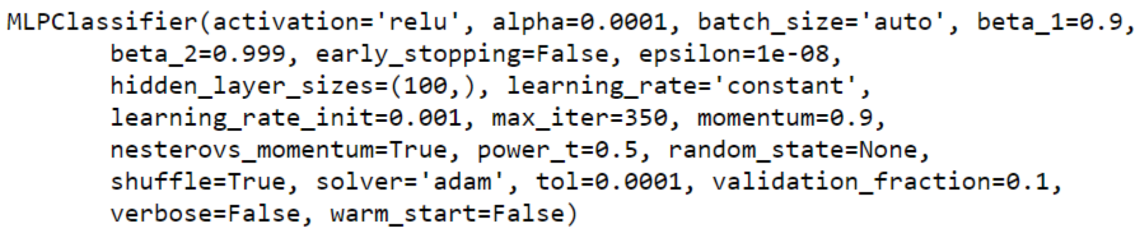
The following figure (Figure 4.29) shows the Support Vector Machine model used in this study. The parameters are also those that come by default from the *Scikit-learn* library, as in the case of Decision Tree Learner. A more detailed description of these parameters can be found in the Annex B.



```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Figure 4.29: Support Vector Machine algorithm implemented in the model with the parameters used.

Finally, Figure 4.30 shows the neural network algorithm used. It is a multi-layer perceptron classifier and has been implemented from the *Scikit-learn* library. In this case, it was decided to vary the parameter of the maximum number of iterations, which was set at 350, after the tests carried out. The rest of the parameters are kept by default. A more detailed description of these parameters can be found in Annex C.



```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=350, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Figure 4.30: Neural Network Multi-layer Perceptron algorithm implemented in the model with the parameters used.

4.6. Model implementation

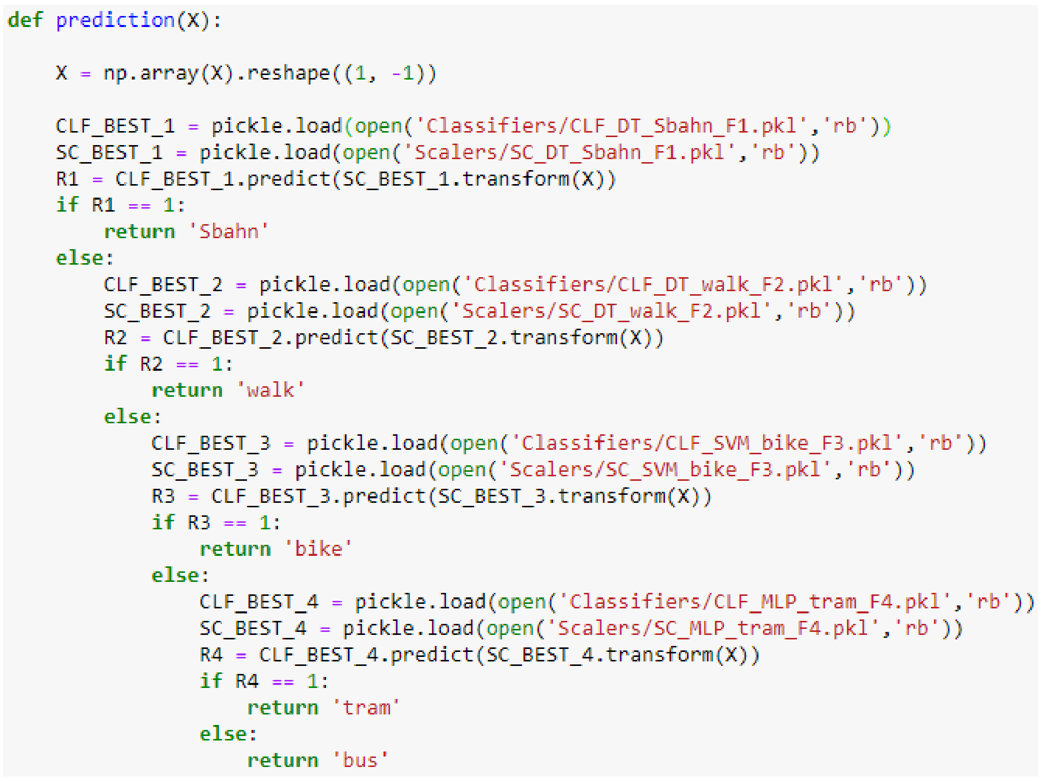
The objective of implementing the model is to predict the mode of transport used for a particular track that is not labeled, i.e. the mode of transport used in the track is not known. The unlabeled tracks will be processed by the final algorithm and the result will be the prediction for each of the tracks.

For the application of the algorithm to make sense, unlabeled tracks must belong to the group of means of transport on which the study was carried out: S-Bahn, walk, bike, tram, bus.

In order to make the prediction, the function shown in Figure 4.31 has been developed. The prediction function has four blocks, belonging to the four phases described. Between the blocks the conditionals if-else can be observed, so that after a certain block the track receives the prediction if it fulfills the condition and if it does not, it passes to the next block that contains the next classifier.

In this way, the first two lines of each block load the classifier and scaler saved in the corresponding directories using the *pickle.load()* function. It is important to note that in order to scale new unlabeled data, the trained scaler must be used. The new data are scaled from the mean and standard deviation of the training phase.

As seen in the third line of each block, the scaler is applied to the new data from the function *transform()* and the value of the track is predicted with the function *predict()*. The track will take value 1 if it belongs to the class to be distinguished in the block and if not, it will take value 0 and go on to the next block. In the last block, the classification is made between tram (1) and bus (0). The function returns the name corresponding to the means of transport predicted in each case by means of the *return* command.

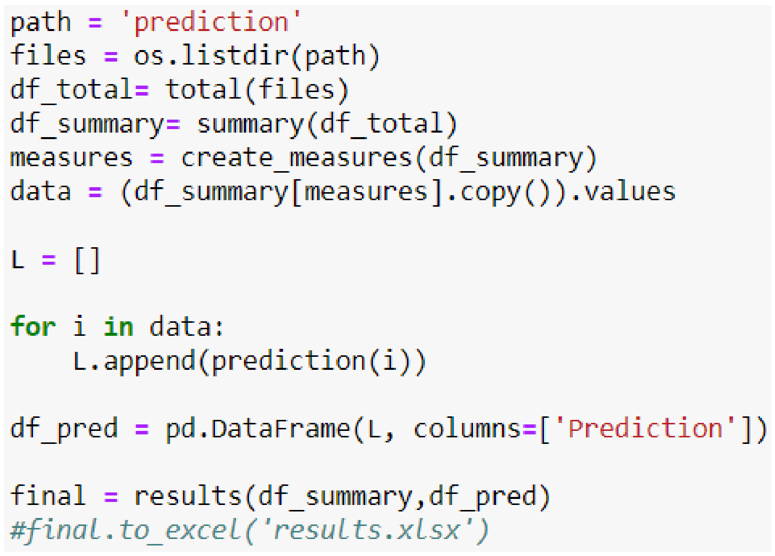


```
def prediction(X):

    X = np.array(X).reshape((1, -1))

    CLF_BEST_1 = pickle.load(open('Classifiers/CLF_DT_Sbahn_F1.pkl','rb'))
    SC_BEST_1 = pickle.load(open('Scalers/SC_DT_Sbahn_F1.pkl','rb'))
    R1 = CLF_BEST_1.predict(SC_BEST_1.transform(X))
    if R1 == 1:
        return 'Sbahn'
    else:
        CLF_BEST_2 = pickle.load(open('Classifiers/CLF_DT_walk_F2.pkl','rb'))
        SC_BEST_2 = pickle.load(open('Scalers/SC_DT_walk_F2.pkl','rb'))
        R2 = CLF_BEST_2.predict(SC_BEST_2.transform(X))
        if R2 == 1:
            return 'walk'
        else:
            CLF_BEST_3 = pickle.load(open('Classifiers/CLF_SVM_bike_F3.pkl','rb'))
            SC_BEST_3 = pickle.load(open('Scalers/SC_SVM_bike_F3.pkl','rb'))
            R3 = CLF_BEST_3.predict(SC_BEST_3.transform(X))
            if R3 == 1:
                return 'bike'
            else:
                CLF_BEST_4 = pickle.load(open('Classifiers/CLF_MLP_tram_F4.pkl','rb'))
                SC_BEST_4 = pickle.load(open('Scalers/SC_MLP_tram_F4.pkl','rb'))
                R4 = CLF_BEST_4.predict(SC_BEST_4.transform(X))
                if R4 == 1:
                    return 'tram'
                else:
                    return 'bus'
```

Figure 4.31: Code of the function developed to make the prediction of unlabeled tracks.

Finally, the final code to be executed by the user is shown in Figure 4.32. The directory where the user stores the documents with the tracks in **.xlsx* format is defined. The data must be labeled with the date and time of export, to make each track unique, in case the *measurement_fk* number matches, as the data could have been collected with different devices.

Then from the *total*, *summary* and *create_measures* functions, the data is structured and the features are calculated for each track. An *L* list is then created where all predictions will be stored, and in the *for* loop the prediction function is applied. Finally the *Pandas DataFrame* is created that contains the prediction for each track, which can be saved as file **.xlsx*.



```python
path = 'prediction'
files = os.listdir(path)
df_total= total(files)
df_summary= summary(df_total)
measures = create_measures(df_summary)
data = (df_summary[measures].copy()).values

L = []

for i in data:
    L.append(prediction(i))

df_pred = pd.DataFrame(L, columns=['Prediction'])

final = results(df_summary,df_pred)
#final.to_excel('results.xlsx')
```

Figure 4.32: Final code for the application of the model. The functions of data structuring, feature calculation and track prediction are found.

5. Results

5.1. Cascading classifiers

This chapter presents the results obtained for the model designed. As described in the previous chapter, the classification has been carried out in four phases through a cascading classifiers process.

In each phase, the classification has been performed with the three machine learning algorithms mentioned above, and their effectiveness has been analyzed employing the accuracy, F-measure, precision and recall metrics. In each phase, binary classification is carried out, between the means of transport (value 1) to be recognized and the rest of the existing classes (value 0).

The analysis of results focuses on class 1, which represents the mode of transport to be classified in each of the phases. However, it is important that the following two points are understood:

- Based on the results obtained for class 1: Precision metric intuitively represents the ability of the classifier not to classify as class 1 a sample belonging to class 0. The recall represents the classifier's ability to find all class 1 samples [18].

- Based on the results obtained for class 0: Precision metric intuitively represents the ability of the classifier not to classify as class 0 a sample belonging to class 1. The recall represents the classifier's ability to find all class 0 samples [18].

As previously explained, there is the training process and the test process. After training each of the models, the test phase is carried out, for which the evaluation measures are obtained for each classifier, which are then analyzed.

When evaluating the performance of the model, the metrics obtained for each machine learning algorithm in each of the phases will be compared separately. The algorithm with the best results will be the one finally used for the implementation of the model. As criteria to analyze the metrics, it has been decided to study mainly the accuracy and the F-measure, since the latter represents the harmonic mean of the precision and recall.

Numerous tests have been conducted with different feature combinations. The aim was to carry out a study selecting a low number of variables that could represent the behavior of each means of transport. However, the best performance of the cascade classifiers has been obtained for the set of 43 variables in Table 5.1, calculated for every track.

Variable	Component
Acceleration	- mean, maximum, standard deviation
	- percentile: 10, 20, 30, 40, 50, 60, 70, 80, 90
Absolute Acceleration	- mean, maximum, standard deviation
	- percentile: 10, 20, 30, 40, 50, 60, 70, 80, 90
Speed	- mean, maximum, standard deviation
	- percentile: 10, 20, 30, 40, 50, 60, 70, 80, 90
Distance ratio	- ratio
Bearing	- mean
Bearing rate	- mean
Stops	- ratio
Total distance	- total
Total distance 2	- total
Total time	- total

Table 5.1: Set of variables calculated for each track, selected to train the model.

The performance of the classifiers is also measured with the application of Principal Component Analysis. There has been a reduction of 43 variables in 10 principal components, with a variance of 95% for each of the classifiers. Conducting the study with or without PCA does not mean any difference in simulation time.

The performance of the classifiers is then analyzed individually, finally selecting the most appropriate machine learning algorithm for the classification in each of the 4 phases. Each case will be analyzed with and without the application of PCA.

In the results tables the columns with the precision metrics (Accuracy, F-measure, Precision, and Recall), the class (1 or 0), the classified mode (*class_ref*) and the type of algorithm used (DT, SVM and MLP, representing the last Neural Network Multi-layer Perceptron algorithm) are observed.

5.1.1. Classifier 1

5.1.1.1. Results without PCA

Table 5.2 presents the results obtained for the first classifier, where it is intended to distinguish S-Bahn tracks (class 1) from the rest (class 0). In order to evaluate the performance of the model, it is necessary to look at the results obtained for class 1, which is the class to be distinguished.

For any of the models used, it can be seen that the metrics are very high, which explains the great performance of the classifier. The best result is obtained in the case of the

Decision Tree, with an accuracy of 99.7 % and an F-measure (average between precision and recall) of 98.5 %. The results obtained for the SVM and MLP model are also very high and guarantee a good classification for the S-Bahn tracks, but the DT model has been chosen to train the model in this phase, as it has the highest values of the evaluation measures.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.997073171	0	0.998380785	0.999864865	0.996901105	Sbahn	DT
0.997073171	1	0.984790875	0.97125	0.998714653	Sbahn	DT
0.99304878	0	0.996163425	1	0.992356175	Sbahn	SVM
0.99304878	1	0.963058976	0.92875	1	Sbahn	SVM
0.993536585	0	0.996416255	0.995675676	0.997157937	Sbahn	MLP
0.993536585	1	0.967101179	0.97375	0.96054254	Sbahn	MLP

Table 5.2: Evaluation measures for the first classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of S-Bahn (1) of the rest of classes (0).

5.1.1.2. Results with PCA

Table 5.3 shows the results obtained for the classification of S-Bahn with the application of PCA. Comparing the class 1 classification metrics for the three types of algorithms with the analysis without PCA, identical results are observed for the DT and SVM cases. In the case of MLP, taking into account the evaluation measures of accuracy and F-measure, a slight worsening can be observed in the case of PCA, insignificant for the performance of the classifier.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.997073171	0	0.998380785	0.999864865	0.996901105	Sbahn	DT
0.997073171	1	0.984790875	0.97125	0.998714653	Sbahn	DT
0.99304878	0	0.996163425	1	0.992356175	Sbahn	SVM
0.99304878	1	0.963058976	0.92875	1	Sbahn	SVM
0.993414634	0	0.996348391	0.995540541	0.997157553	Sbahn	MLP
0.993414634	1	0.966501241	0.97375	0.959359606	Sbahn	MLP

Table 5.3: Evaluation measures for the first classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of S-Bahn (1) of the rest of classes (0) with the application of PCA.

5.1.2. Classifier 2

5.1.2.1. Results without PCA

In the second phase, the aim is to classify the walk tracks (1) from the rest (0). As previously explained, after phase 1, the tracks that the model has classified as not S-Bahn move to phase 2.

Based on the evaluation metrics for class 1 (walk), it can be seen in Table 5.4 that this classifier performs better in the case of the Decision Tree since its accuracy and F-measure (which summarizes the precision and recall metrics) have the highest values when compared to SVM and MLP. The differences are very small and any of them could be used for the classification, but the training of classifier 2 with Decision Tree will be chosen due to its slightly better performance. The tracks that the model has predicted as no walk (0) pass to the next classifier

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.991216216	0	0.99384761	0.990566038	0.997150997	walk	DT
0.991216216	1	0.984651712	0.992857143	0.976580796	walk	DT
0.989459459	0	0.992644285	0.993018868	0.992269985	walk	SVM
0.989459459	1	0.981410867	0.98047619	0.982347328	walk	SVM
0.980540541	0	0.986350711	0.981698113	0.991047619	walk	MLP
0.980540541	1	0.966117647	0.977619048	0.954883721	walk	MLP

Table 5.4: Evaluation measures for the second classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of walk (1) of the rest of classes (0).

5.1.2.2. Results with PCA

Comparing the performance of the classifier for class 1, the evaluation measures (Table 5.5) are identical for the DT and SVM cases in the case without PCA. In the case of MLP, a slight improvement is observed in the metric F-measure, barely appreciable.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.991216216	0	0.99384761	0.990566038	0.997150997	walk	DT
0.991216216	1	0.984651712	0.992857143	0.976580796	walk	DT
0.989459459	0	0.992644285	0.993018868	0.992269985	walk	SVM
0.989459459	1	0.981410867	0.98047619	0.982347328	walk	SVM
0.980540541	0	0.986342944	0.981132075	0.991609458	walk	MLP
0.980540541	1	0.966165414	0.979047619	0.953617811	walk	MLP

Table 5.5: Evaluation measures for the second classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of walk (1) of the rest of classes (0) with the application of PCA.

5.1.3. Classifier 3

5.1.3.1. Results without PCA

In classifier 3, the bike tracks (1) will be distinguished from the remaining classes (0). Table 5.6 shows the evaluation measures obtained. As in the previous cases, a great performance of the classifier is observed for the three types of machine learning models used. In this case, the performance of the SVM stands out slightly, with an accuracy of 99.4 % and an F-measure of 99.4 %. After classifying the bike tracks, the tracks that the model has predicted as bus and tram will go on to the next phase.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.974150943	0	0.972528574	0.97	0.975070366	bike	DT
0.974150943	1	0.975592375	0.977857143	0.973338073	bike	DT
0.994150943	0	0.993781344	0.9908	0.996780684	bike	SVM
0.994150943	1	0.994479074	0.997142857	0.991829485	bike	SVM
0.988490566	0	0.987836491	0.9908	0.984890656	bike	MLP
0.988490566	1	0.989077887	0.986428571	0.991741472	bike	MLP

Table 5.6: Evaluation measures for the third classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of bike (1) of the rest of classes (0).

5.1.3.2. Results with PCA

Table 5.7 shows the evaluation metrics obtained for the classification of bicycle tracks with the application of PCA. As for the two previous cases of analysis with PCA, the evaluation measures are identical for the cases of DT and SVM in this classifier (comparing with the case without the application of PCA). Metrics for MLP are slightly higher (accuracy and F-measure) for the case without PCA application.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.974150943	0	0.972528574	0.97	0.975070366	bike	DT
0.974150943	1	0.975592375	0.977857143	0.973338073	bike	DT
0.994150943	0	0.993781344	0.9908	0.996780684	bike	SVM
0.994150943	1	0.994479074	0.997142857	0.991829485	bike	SVM
0.98754717	0	0.98684735	0.9904	0.983320095	bike	MLP
0.98754717	1	0.988176281	0.985	0.991373113	bike	MLP

Table 5.7: Evaluation measures for the third classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of bike (1) of the rest of classes (0) with the application of PCA.

5.1.4. Classifier 4

5.1.4.1. Results without PCA

Finally, in phase 4, the classification between tram and bus tracks is carried out, which is the most complex of the study, due to the similar behavior of both means of transport in city traffic. After different tests, Table 5.8 shows the best results obtained.

Compared to the previous phases, it is observed that the values of evaluation measures are lower since the classification is more difficult than in the previous phases. For this phase, the performance of MLP stands out over DT and SVM, with better metrics.

MLP has an accuracy of 83 % compared to 79 % and 68 % for SVM and DT respectively. A comparison of the F-measure shows a better performance for MLP (86 %) compared to 84 and 74 % for SVM and DT respectively.

For this case it is important to also observe the metrics obtained for class 0, which are the results obtained for the classification of bus tracks. The results for accuracy and F-measure are 83 and 79.2 % respectively, in the MLP case. Based on the evaluation metrics of DT and SVM for bus classification (0), MLP obtains better results, as in the case of tram. When comparing with the metrics obtained for class 1, it is observed that the model has a better performance to classify tram tracks vs. bus than bus tracks vs. tram tracks.

At this stage, the metrics obtained are not as similar as in the other classifiers. That is why for the fourth classifier, the MLP algorithm is chosen to train the model to guarantee better performance.

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.6832	0	0.600806452	0.596	0.605691057	tram	DT
0.6832	1	0.737400531	0.741333333	0.733509235	tram	DT
0.79	0	0.698795181	0.609	0.819650067	tram	SVM
0.79	1	0.83880872	0.910666667	0.777461582	tram	SVM
0.8312	0	0.792118227	0.804	0.780582524	tram	MLP
0.8312	1	0.857912458	0.849333333	0.866666667	tram	MLP

Table 5.8: Evaluation measures for the fourth classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of tram (1) and bus (0).

5.1.4.2. Results with PCA

The metrics for classifier 4 with the application of PCA are shown in Table 5.9. As is the case for the other classifiers, when comparing the metrics with and without PCA analysis, it is observed in this phase that the evaluation measures for the tram vs. bus classification are identical for the DT and SVM types. Comparing the MLP case, the metrics (based on accuracy and F-measure) are slightly higher when PCA is applied, for class 1 (for class 0, the F-measure is slightly lower).

Accuracy	Class	F-measure	Precision	Recall	Class_ref	Type
0.6832	0	0.600806452	0.596	0.605691057	tram	DT
0.6832	1	0.737400531	0.741333333	0.733509235	tram	DT
0.79	0	0.698795181	0.609	0.819650067	tram	SVM
0.79	1	0.83880872	0.910666667	0.777461582	tram	SVM
0.8316	0	0.791687284	0.8	0.783545544	tram	MLP
0.8316	1	0.858677409	0.852666667	0.864773496	tram	MLP

Table 5.9: Evaluation measures for the fourth classifier (accuracy, F-measure, precision, recall) for the three types of models (DT, SVM, MLP), for the classification of tram(1) and bus (0) with the application of PCA.

5.2. Model implementation results

After the training and test phase, for each of the four phases with the three machine learning algorithms (DT, SVM, MLP), the results obtained are analyzed (evaluation measures) and the cascading classifiers model is trained. In each phase, the machine learning algorithm that has performed best in the tests is chosen. In this way, Table 5.10 summarizes the phases of the cascading classifiers.

It has been decided not to use PCA for the study. As seen in the previous section, the results were always better (slightly) for the case without the application of PCA. In addition, when applying PCA, there is a small percentage of information loss (5%), although it does not practically affect the outcome of the model. If the simulation time were considerably longer when working with all features instead of working with the principal components, the application of PCA could be considered, as it works with 10 principal components instead of 43 features.

Classifier	Algorithm	Mode of transport to classify
1	Decision Tree Learner (DT)	S-Bahn
2	Decision Tree Learner (DT)	Walk

3	Support Vector Machine (SVM)	Bike
4	Neural Network Multi-layer Perceptron (NN-MLP)	Tram, Bus

Table 5.10: Summary of the machine learning algorithms used for the cascade classification process.

With the cascading classifiers model trained and the algorithm designed to make the prediction of unlabeled GPS tracks, the prediction was made for two batches of tracks collected to observe how the classification algorithm responds to non-labeled trips, knowing the mode of transport used to know if the prediction was made correctly.

Tables 5.11 and 5.12 show the results obtained for batch 1 and batch 2 respectively. For batch 1 there are 26 tracks and for batch 2 a total of 42 tracks, with variation of the five modes considered in this study. The columns contain the *measurement_fk* number assigned by the *Cyface* application and the time and date when the data were exported. The last two columns show the prediction assigned by the algorithm for each track and the mode of transport actually used. In the last two columns, the errors produced are highlighted.

measurement_fk	time_exp	date	Prediction	Mode used
1	10:41:00	20/05/2019	tram	tram
2	10:41:00	20/05/2019	tram	tram
3	10:41:00	20/05/2019	tram	tram
4	10:41:00	20/05/2019	tram	tram
5	10:41:00	20/05/2019	walk	walk
6	10:41:00	20/05/2019	bus	tram
8	10:41:00	20/05/2019	tram	tram
9	10:41:00	20/05/2019	walk	walk
10	10:41:00	20/05/2019	tram	tram
11	10:41:00	20/05/2019	tram	tram
12	10:41:00	20/05/2019	tram	tram
13	10:41:00	20/05/2019	Sbahn	S-Bahn
14	10:41:00	20/05/2019	Sbahn	S-Bahn
15	10:41:00	20/05/2019	Sbahn	S-Bahn
16	10:41:00	20/05/2019	Sbahn	S-Bahn
17	10:41:00	20/05/2019	Sbahn	S-Bahn
18	10:41:00	20/05/2019	Sbahn	S-Bahn
19	10:41:00	20/05/2019	Sbahn	S-Bahn
20	10:41:00	20/05/2019	Sbahn	S-Bahn
21	10:41:00	20/05/2019	Sbahn	S-Bahn
22	10:41:00	20/05/2019	Sbahn	S-Bahn

23	10:41:00	20/05/2019	Sbahn	S-Bahn
24	10:41:00	20/05/2019	Sbahn	S-Bahn
25	10:41:00	20/05/2019	walk	walk
26	10:41:00	20/05/2019	Sbahn	S-Bahn
28	10:41:00	20/05/2019	tram	tram

Table 5.11: Prediction results obtained for batch 1. Errors are highlighted with a box.

measurement_fk	time_exp	date	Prediction	Mode used
169	18:13:00	28/06/2019	tram	tram
170	18:13:00	28/06/2019	bus	tram
171	18:13:00	28/06/2019	walk	walk
172	18:13:00	28/06/2019	walk	walk
173	18:13:00	28/06/2019	walk	walk
174	18:13:00	28/06/2019	bike	bike
175	18:13:00	28/06/2019	bike	bike
176	18:13:00	28/06/2019	bike	bike
177	18:13:00	28/06/2019	bike	bike
178	18:13:00	28/06/2019	tram	tram
180	18:13:00	28/06/2019	tram	tram
181	18:13:00	28/06/2019	walk	walk
182	18:13:00	28/06/2019	walk	walk
183	18:13:00	28/06/2019	walk	walk
184	18:13:00	28/06/2019	tram	tram
185	18:13:00	28/06/2019	walk	walk
186	18:13:00	28/06/2019	bike	bike
187	18:13:00	28/06/2019	bike	bike
188	18:13:00	28/06/2019	bike	bike
189	18:13:00	28/06/2019	bike	bike
190	18:13:00	28/06/2019	bike	bike
191	18:13:00	28/06/2019	bike	bike
192	18:13:00	28/06/2019	tram	tram
193	18:13:00	28/06/2019	tram	tram
194	18:13:00	28/06/2019	walk	walk
195	18:13:00	28/06/2019	bike	bike
196	18:13:00	28/06/2019	bike	bike
197	18:13:00	28/06/2019	bike	bike
198	18:13:00	28/06/2019	bike	bike
199	18:13:00	28/06/2019	bike	bike
200	18:13:00	28/06/2019	bus	tram
201	18:13:00	28/06/2019	bus	bus
202	18:13:00	28/06/2019	bus	bus

203	18:13:00	28/06/2019	bus	bus
204	18:13:00	28/06/2019	bus	bus
205	18:13:00	28/06/2019	bus	bus
206	18:13:00	28/06/2019	tram	bus
207	18:13:00	28/06/2019	bus	bus
208	18:13:00	28/06/2019	bus	bus
209	18:13:00	28/06/2019	bus	bus
210	18:13:00	28/06/2019	bus	bus
211	18:13:00	28/06/2019	tram	bus

Table 5.12: Prediction results obtained for batch 2. Errors are highlighted with a box.

6. Discussion

6.1. Performance of this study

After presenting in the previous chapter the evaluation metrics obtained for this study, the results are analyzed and interpreted. The results obtained for the first three classifiers reflect a great performance of the model and ensure a very good classification of the tracks of S-Bahn, walk and bike.

Several conclusions can, therefore, be drawn from these results:

- In the city of Dresden, the behavior of these three modes of transport differs from the rest of the modes studied, making it very easy for the model to classify each of them individually.

- The behavior of each mode within the traffic can be understood through the calculated features (section 4.4.4). The results indicate a very good criterion when calculating and selecting the list of features that have been inputs in the machine learning models. Based on the results obtained, the calculated features faithfully represent these three means of transport.

- The machine learning models implemented have been very well chosen since the performance is outstanding. The chosen parameters (mostly set by default from the *Scikit-learn* library) make the algorithms work almost to perfection, there is not much improvement in the results.

- For each classifier, attention has been paid to the results obtained for class 1, which represents the mode to be classified in each phase. The results obtained for class 0 represent the performance of the model to classify the rest of the classes. Taking into account both types of classification, it is observed that for the first three phases the classification is excellent, with metrics (accuracy, precision, recall) around 96-99 %.

As explained above, the analysis of results has been based on the metrics obtained for class 1 in each classifier, since the objective is to obtain a good performance to separate the mode of transport in question (class 1) from the rest of the modes (class 0). On the basis of the results obtained, it would also be possible to measure the performance of each classifier by looking at the results obtained for class 0, which represent the performance of the model to classify the rest of the modes (class 0) of transport as opposed to the mode in question (class 1).

In any case, for the first three classifiers and all the algorithms used, the accuracy, precision, and recall metrics are very high, which can be summed up in a very good performance of the model to differentiate the modes of transport of S-Bahn, walk and bike from the rest of the modes.

The performance of the model in the fourth classifier, whose function is to classify the remaining tracks between bus and tram is not as outstanding compared to previous phases. In addition, there is an important difference depending on the machine learning algorithm used, highlighting the neural networks above the rest, with metrics around 85%. The worse performance of this classifier compared to the previous three is due to some reasons:

- In the city of Dresden, tram and bus behavior in traffic is very similar. There are no major traffic jams, so traffic is fluid. Both modes of transport make stops at traffic lights and at the stops established for passengers getting in and out of the vehicle, so the number of times they stop is similar. In addition, they share many stretches of road and traffic lights. The speeds reached in straight sections are very similar, as has been seen in the calculation of features. These reasons justify the similar behavior of both modes in traffic.

- It may seem that the bus makes more turns on its routes compared to the tram, which was thought to be a differential criterion. As can be seen in section 4.4.4, the turn measurement was calculated from the bearing and bearing rate features, and it turned out not to be so differential, as can be seen in Figure 4.13 and in the performance of classifier 4.

- A wide set of features has been calculated, and the model has been tested with different combinations to obtain the best performance of the model. The similar behavior of tram and bus has made difficult to calculate features able to distinguish the tracks of both means of transport. In addition, for the model of designed cascading classifiers, a list of features is calculated, that will be inputs for every classifier so that for this model the combinations of features are not calculated for the individual classification of each mode. That is why the final list contains a large number of features in which tram and bus have very similar values, hence the performance of classifier 4 is not as outstanding as the previous ones. However, very high metrics (accuracy, precision, recall) have been reached (around 85 % for NN-MLP) which guarantees a good performance in the implementation of the model.

Comparing in phase 4 the precision and recall metrics for tram classification (class 1) with those for bus classification (class 0), it can be seen that the classifier 4 performs better for tram classification. For any of the algorithms, F-measure (precision mean and recall) is higher for tram classification. This can be understood as follows:

- Comparing the precision and recall metrics and taking into account the global set of remaining tracks (tram and bus) that pass to phase 4, it is concluded that the algorithm (DT, SVM, MLP) has greater ability to classify the tram vs. bus tracks than the bus vs. tram tracks.

- This may be due to the fact that the inputs fed to the algorithm are more adapted to the real characteristics of the tram than the bus and therefore better results are obtained. Another reason could be that the implemented algorithms are better adapted to the tram classification. This study focuses on the application of the algorithms and does not attempt a deep optimization of them, since these reasons are simply assumptions.

The application of Principal Component Analysis (PCA) is not a big change in either the results or the simulation time, in fact it is invaluable. The 43 features calculated are reduced to 10 principal components, with a variance of 95% (where there is a minimum loss of information). This is why for the implementation of the model the case with application of PCA has not been considered.

6.2. Performance of other studies

In this section, a comparison of the results obtained with those of the studies mentioned in the State of Art section will be made. For the implementation of the model developed in this study, the cascading classifiers model will be used, and in each phase (classifier) the algorithm that has had the best performance in the training and test phase will be used. For comparison, the metrics obtained in the test phase will be analyzed. When comparing with other studies, it is necessary to take into account key factors that make each case different:

- Each study was conducted in a different city. This means a different situation since traffic varies according to the size of the city, population, traffic lights, routes, infrastructures, etc.

- Vehicles may have different characteristics depending on the city in which the study is carried out (speed, acceleration, etc.), which in turn depends on the route established and the type of traffic existing in each case.

- Four phases have been carried out in this study. In each phase, three different algorithms are used to classify a type of transport mode. The best metrics obtained have been taken as a reference to compare with other studies and to select the algorithm that will be used in the implementation of the model. Each study represents a particular case, many different algorithms have been used to make a different classification of means of transport and also using mean values of metrics to study the results. It is, therefore, not easy to find comparable results from other studies. A comparison will be made for individual classification of each means of transport.

Table 6.1 shows the results obtained (accuracy, precision, recall) in the test phase of the different studies for Support Vector Machine.

Study	Algorithm	MODE	Accuracy	Precision	Recall
[10]	SVM	Bus	91%	-	-
[10]	SVM	Tram	78%	-	-
[10]	SVM	Train	100%	-	-
[28]	SVM	Walk	100%	-	-
[28]	SVM	Bike	100%	-	-
[28]	SVM	Bus	92.70%	-	-
[31]	SVM	Walk	-	88.80%	86.90%
[31]	SVM	Bike	-	81.60%	87.10%
Thesis	SVM	S-Bahn	99.3%	93%	100%
Thesis	SVM	Tram	79%	91%	77.7%
Thesis	SVM	Bus	79%	60.9%	82%
Thesis	SVM	Bike	99.4%	99.7%	99.2%
Thesis	SVM	Walk	99%	98%	98.2%

Table 6.1: Evaluation metrics obtained in different studies for different modes of transport using the SVM algorithm.

Observing the results of L. Zhang *et al.* [10] the accuracy of the means of transport bus, tram and train is observed as an evaluation measure. High values are observed, especially for train and bus. Comparing with the accuracy measurements obtained in this study (99.3% S-Bahn and 79% for bus and tram) it is concluded that the performance of the model for classifying train (S-Bahn) and tram is very similar, but the study of L. Zhang *et al.* [10] has a better performance for classifying bus mode. It is important to note again the similar behavior of bus and tram in the city of Dresden, which makes it difficult to distinguish between bus and tram due to their similar characteristics.

Comparing to the results of F.Zong *et al.* [28], a better performance for the bus classification is again observed. The accuracy results obtained in this study for walk and bike (99 % and 99.4 % respectively) reflect a very good performance of the model using SVM, as well as in the study of F.Zong *et al.* [28].

Finally, the evaluation metrics (precision and recall) for the walk and bike modes obtained from the study of S. Reddy *et al.* [31] are compared. The results obtained in this study show a better performance. For walk, a precision and recall of around 99% are achieved, surpassing those of S. Reddy *et al.* [31], below 90%. For bike, the precision and recall metrics are still around 99%, higher than those obtained by S. Reddy *et al.* [31], which are again below 90%.

Table 6.2 shows the results obtained (accuracy, precision, recall) in the test phase of the different studies for Neural Networks.

Study	Algorithm	MODE	Accuracy	Precision	Recall
[31]	NN	Walk	-	87.30%	88%
[31]	NN	Bike	-	84.80%	84.20%
[29]	NN	Walk	-	98.51%	96.59%
[29]	NN	Bike	-	87.50%	94.59%
[29]	NN	Bus	-	88.37%	92.68%
[1]	NN	Walk	-	81.60%	95.70%
[1]	NN	Bike	-	90.30%	82.60%
[1]	NN	Bus	-	80.70%	81.10%
[1]	NN	Train	-	92.30%	85.30%
[2]	NN	Walk	100%	-	-
[2]	NN	Bus	81.58%	-	-
[5]	NN	Walk	96.10%	-	-
[5]	NN	Bike	93.30%	-	-
[5]	NN	Bus	88.80%	-	-
Thesis	NN	S-Bahn	99.3%	97.4%	96%
Thesis	NN	Tram	83.12%	85%	86.7%
Thesis	NN	Bus	83.12%	80.4%	78 %
Thesis	NN	Bike	98.8%	98.6%	99.1%
Thesis	NN	Walk	98%	98%	95.5%

Table 6.2: Evaluation metrics obtained in different studies for different modes of transport using the NNs algorithm.

Looking at the metrics obtained for the different studies in Table 6.2, the precision and recall metrics from the study of G. Xiao *et al.* [29] stand out. The precision and recall metrics obtained respectively for this study are 98 % and 95.5 % for walk, 98.6 % and 99.1 % for bike and 80.4 % and 78 % for bus.

Comparing the results obtained in this study with the highest values (those obtained by G. Xiao *et al.* [29]) a very similar performance is observed for the walk case, a better performance for the bike case and a worse performance for the bus classification. Once again, the existing problem is observed for the classification of bus and tram of this study.

It is also worth mentioning the train classification results of S. Dabiri and K. Heaslip [1], with precision and recall values lower than those obtained in this study (92.3 % and 85.3 % vs. 97.4 % and 96 %). Finally, studies are compared with accuracy as the only evaluation measure. The values obtained in this study for walk (98 %) and bike (98.8 %) are very high but once again the accuracy obtained for bus (83.12 %) is below other studies such as the accuracy from the study of F. Yang *et al.* [5], although for this case the difference is not that high.

Finally, table 6.3 presents the metrics obtained for Decision Tree in other studies.

Study	Algorithm	MODE	Accuracy	Precision	Recall
[31]	DT-DHMM	Walk	-	92.40%	90.80%
[31]	DT-DHMM	Bike	-	87.90%	90.60%
[31]	DT	Walk	-	87.60%	88.40%
[31]	DT	Bike	-	84.50%	85.30%
[32]	DT	Walk	-	94.70%	98.90%
[32]	DT	Bike	-	85.50%	94.60%
[32]	DT	Bus	-	88.90%	84.20%
[32]	DT	Train	-	96.80%	96.80%
Thesis	DT	S-Bahn	99.7%	97.1%	99.9%
Thesis	DT	Bus	68.3%	59.6%	60.6%
Thesis	DT	Bike	97.4%	97.8%	97.3%
Thesis	DT	Walk	99.1%	99.3%	97.65%

Table 6.3: Evaluation metrics obtained in different studies for different modes of transport using the DT algorithm.

In the study of S. Reddy *et al.* [31] two different models of DT are presented, one of them followed by a first-order discrete Hidden Markov Model, which has a better performance (for the classification of bike and walk). The metrics obtained in the present study for precision and recall with DT algorithm are respectively 99.3 and 97.65 % (walk), 97.8 and 97.3 % (bike), 97.1 and 99.9 % (S-Bahn) and 59.6 and 60.6 % (bus).

The performance to classify the walk, bike and S-Bahn modes is very high and surpasses those of other studies for this case. It is observed that the metrics for the walk and train classification in the results obtained by L. Stenneth *et al.* [32] also reflect a good performance. The case of bus classification is clearly better in other studies (like the last one mentioned). Especially in the case of DT, the metrics obtained in the present study are low, around 60%, which does not ensure a good performance.

6.3. Model implementation

The results obtained in Tables 5.11 and 5.12 show the predictions made by the model for unlabeled data. They are a total of 68 tracks, 26 in batch 1 and 42 in batch 2. Tables 6.4 and 6.5 summarize the number of tracks for each mode in the different batches. As seen in Tables 5.11 and 5.12, in batch 1 there is only one prediction error, a tram track predicted as bus. In batch 2 there are four errors, two tram tracks predicted as bus and two bus tracks predicted as tram.

As can be seen, the prediction errors produced are only for the case of tram and bus, for both batches. Based on the general results (Table 6.6), three errors are observed for the prediction of tram tracks (predicted as bus) and two errors for the prediction of bus tracks (predicted as tram). If the accuracy column is observed, the results obtained are as expected. For the prediction of bike tracks, walk and S-Bahn, the prediction is perfect, as shown by the 100% accuracy. The accuracy for the prediction of tram tracks is 83 % and for the prediction of bus tracks is 82 %, results very similar to those obtained in the validation phase for classifier 4 with MLP (Table 5.8). Observing the general results (Table 6.6), for a total of 68 tracks there were only five prediction failures, which means an accuracy of 92.64%, a great performance of the model.

As expected, prediction errors occur for tram and bus tracks. It is then demonstrated the capacity of the model to distinguish without problem unlabeled tracks belonging to the modes of S-Bahn, walk and bike. In the case of tram and bus, the prediction capacity (around 80 %) has a high level of success, but there may be failures. In order to correct this, it would be necessary to try to obtain features that more faithfully represent each mode of transport or to use other techniques, such as Geographical Information Systems (GIS) or other data captured by smartphone sensors, such as accelerometer or magnetometer.

Mode	Mode used	Accuracy of prediction	Errors (Predicted)
tram	10	90%	1 (bus)
S-Bahn	13	100%	-
walk	3	100%	-
total	26	96.15% (General)	1

Table 6.4: Results obtained for batch 1. It shows the accuracy obtained by modes, total accuracy and errors committed in the prediction.

Mode	Mode used	Accuracy of prediction	Mistakes (Predicted)
tram	9	75%	2 (bus)
bike	15	100%	-
bus	11	81.81%	2 (tram)
walk	8	100%	-
total	43	90.7% (General)	4

Table 6.5: Results obtained for batch 2. It shows the accuracy obtained by modes, total accuracy and errors committed in the prediction.

Mode	Mode used	Accuracy of prediction	Mistakes (Predicted)
tram	18	83.33333333	3 (bus)
bike	15	100	-
bus	11	81.81818182	2 (tram)
walk	11	100	-
S-Bahn	13	100	-
total	68	92.64 % (General)	5

Table 6.6: Results obtained for the total set of tracks. It shows the accuracy obtained by modes, total accuracy and errors committed in the prediction.

7. Summary and Outlook

7.1. Summary of the work

This study focuses on developing a machine learning model capable of automatically classifying unlabeled GPS tracks into different modes of transport (S-Bahn, tram, bus, bike and walk), although the main purpose is to separate the bike tracks from the rest. The purpose of the Chair of Transport Ecology is to promote sustainability in transport, focusing this study on trying to improve mobility by bicycle. However, it is more challenging to try to classify a greater number of modes. Some of the existing machine learning techniques are first introduced and then applied to the classification problem.

At the beginning of the work, the concept of GPS as a tool to collect passenger information is briefly introduced, together with the existence of new machine learning techniques for the specific case of classification of modes of transport, useful for the design and planning of traffic in cities.

The Decision Tree Learner, Support Vector Machine and Neural Network techniques are introduced to have a basic knowledge of the machine learning models to be applied. Other techniques, which are used for data processing, such as Normalization, Principal Component Analysis, Model Training and Validation as well as Result Evaluation Metrics are also explained.

Once the theoretical bases of data processing and machine learning have been defined and understood, the study is carried out, setting on the one hand the objective and on the other the scope of the study. The objective is to classify unlabeled GPS tracks collected through a smartphone application into the corresponding modes of transport from the above mentioned machine learning techniques. The scope will range from data collection, labelling, data preprocessing and data processing to implementation of the model for non-labeled tracks.

The methods section details the tools used to work with the data and how the process of data collection, labeling, structuring, feature calculation, standardization, training and validation and implementation of the final model has been developed. It is important to mention the development of a cascading classifiers model, where tracks are classified in phases according to mode type. The process is explained in detail in the Data Processing section.

The results obtained were compared with those of other studies, having a very comparable performance. The comparison was made on the basis of the mode of transport and the evaluation metrics used. The most important conclusions that can be extracted from this study are the following:

- The possibility of using the smartphone with GPS, compared to traditional methods (surveys) to collect tracks of travelers is a very useful tool to improve efficiency in designs and planning of traffic.

- The possibility for travelers to track their routes (applying privacy measures) is a great advantage, but due to lack of motivation or distrust, it is difficult to obtain a large amount of data this way.

- Structuring data efficiently is a key factor when working with large amounts of data. The libraries implemented in the *Jupyter Notebook* platform contain a large number of functions that allow a great versatility for working with data, as well as documentation with great help to solve the problems that appear when processing data.

- After calculating the potential features, it is essential to make a correct selection of those that most faithfully represent each means of transport. To represent graphically (in boxplots for this study) is a first estimation to observe which of them could better carry out the classification. The second phase consists of feeding the list of features selected in the machine learning algorithm and studying the results. Choosing the right features (inputs of the model) ensures a better performance.

- The cascading classifiers model allows an organized classification with which the problem can be focused on the most complicated classification, between those modes whose characteristics are more similar (tram and bus) and therefore more difficult to distinguish. This model ensures great performance when predicting modes of transport with different characteristics to the rest (S-Bahn, walk, bike), through the design of several phases.

- The results obtained show that for the three algorithms used (DT, SVM, NN-MLP) the model presents an outstanding performance when classifying the tracks belonging to the S-Bahn, walk and bike modes. The evaluation metrics obtained (accuracy, precision, recall) are around 99%. This shows the great success in features and machine learning algorithms selection. The separation of bicycle tracks, the first objective set, is, therefore, entirely feasible.

- The problem of classification between tram and bus should be highlighted. These modes have similar characteristics in the traffic of the city of Dresden, so their features have similar values in many cases, which is a problem for their distinction. However, the results obtained ensure a very good performance of the model, with evaluation metrics of around

85%. In the case of classifying vehicles with similar characteristics, special attention is required to the selection of features as well as to the optimization of algorithms, in order to try to improve the performance of the model.

- The good performance obtained has allowed the implementation of the model for use with unlabeled tracks, with great results, which has been a success in meeting the proposed objectives. There is a small percentage of cases in which the prediction is incorrect between bus and tram, as expected. In order to classify the tracks belonging to the bike mode, the initial objective of the study, the prediction of new tracks is totally correct.

7.2. Outlook

After obtaining a good performance of the model developed in this study, it would be interesting to consider its implementation in other projects, in order to classify GPS tracks or, at least, separate tracks from S-Bahn, bike or walk from the rest due to its great effectiveness. For projects related to the design of bicycle traffic, it could be used in projects of the Chair of Transport Ecology of the Technical University of Dresden.

This study aims to recognize the mode of transport used on GPS tracks consisting of routes where only one mode of transport has been used. As seen in other studies, it would be interesting to complement the classification of the mode with the segmentation of the route. In future studies, it would be interesting to design a process of route segmentation, with several modes of transport used, to divide routes into segments and then recognize the mode of transport used.

The amount of data collected is sufficient to carry out this study. However, it would be convenient to collect more tracks. This could be done through mobility campaigns in the city (individuals tracking their routes and labeling trips) or by placing GPS devices on public transport in order to track the different routes. This would allow for greater data variability and a more accurate representation of mobility in the city, leading to better training of the model.

Due to the available resources, the classification has been made for inner-city traffic between the five modes of transport mentioned above. In future studies, it would be interesting to include more modes (car, motorbike, etc.) as well as to include intercity traffic in the mode recognition.

It would be convenient to carry out a deeper study of the features in order to achieve a more faithful representation of the modes, especially in the case of modes of transport with similar characteristics such as tram and bus (and car). More precise features could be obtained by using other smartphone sensors such as the accelerometer or magnetometer. For this study, only GPS information has been considered, but the Cyface application also allows to obtain this type of information measured by the sensors. Another option would be to include GIS information, in order to know, for example, stop locations or the location of train or tram tracks, which could mean very useful information, for a calculation of more realistic features and therefore a more real model, which would allow obtaining better results for those modes with similar behavior.

As previously described, the study is based on the application of machine learning methods for the recognition of means of transport but does not consist of an in-depth analysis of them. This work aims to open lines of research for a deeper study of machine learning methods, to better understand their functioning and optimize parameters. In this study, the parameters established by default from imported libraries have been used, but a more in-depth analysis would be convenient to try to improve the efficiency of the model.

The study has been conducted for traffic in the city of Dresden. The calculated features represent the characteristics of public transport in this city. It would be interesting to study traffic in other cities in order, if it has similarities, to be able to apply the means of transport recognition model to other cities with similar traffic characteristics. It is considered possible to use this method to separate walk and bike tracks in other cities since their characteristics are more general for any city.

8. References

[1] S. Dabiri and K. Heaslip, “Inferring transportation modes from GPS trajectories using a convolutional neural network,” *Transp. Res. Part C Emerg. Technol.*, vol. 86, no. December 2017, pp. 360–371, 2018.

[2] P. A. Gonzalez *et al.*, “Automating mode detection for travel behaviour analysis by using global positioning systems-enabled mobile phones and neural networks,” *IET Intell. Transp. Syst.*, vol. 4, no. 1, p. 37, 2010.

[3] S. Kim *et al.*, “Urban sensing: Using smartphones for transportation mode classification,” *Comput. Environ. Urban Syst.*, vol. 53, pp. 76–86, 2014.

[4] L. Wu, B. Yang, and P. Jing, “Travel mode detection based on GPS raw data collected by smartphones: A systematic review of the existing methodologies,” *Inf.*, vol. 7, no. 4, 2016.

[5] F. Yang, Z. Yao, and P. J. Jin, “GPS and Acceleration Data in Multimode Trip Data Recognition Based on Wavelet Transform Modulus Maximum Algorithm,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2526, pp. 90–98, 2016.

[6] Y. Zheng, L. Liu, L. Wang, and X. Xie, “Learning Transportation Mode from Raw GPS Data for Geographic Applications on the Web,” pp. 247–256, 2008.

[7] J. Hagenauer and M. Helbich, “A comparative study of machine learning classifiers for modeling travel mode choice,” *Expert Syst. Appl.*, vol. 78, pp. 273–282, 2017.

[8] A. Francke and S. Lißner, “Big Data in Bicycle Traffic. A user-oriented guide to the use of smartphone-generated bycicle traffic data,” *Tech. Univ. Dresden*, 2017.

[9] Y. J. Byon and S. Liang, “Real-time transportation mode detection using smartphones and artificial neural networks: Performance comparisons between smartphones and conventional global positioning system sensors,” *J. Intell. Transp. Syst. Technol. Planning, Oper.*, vol. 18, no. 3, pp. 264–272, 2014.

[10] L. Zhang, S. Dalyot, D. Eggert, and M. Sester, “Multi-Stage Approach To Travel-Mode Segmentation and Classification of Gps Traces,” *ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XXXVIII-4/, no. October, pp. 87–93, 2012.

[11] R. Polikar, D. Dera, I. Safro, P. Bhavsar, and N. Bouaynaya, “Machine Learning in Transportation Data Analytics,” *Data Anal. Intell. Transp. Syst.*, no. December, pp. 283–307, 2017.

[12] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd ed.)*, 3rd ed. Prentice Hall, 2010.

[13] J. Zhang, F. Y. Wang, K. Wang, W. H. Lin, X. Xu, and C. Chen, “Data-driven intelligent transportation systems: A survey,” *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, 2011.

[14] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, “A Practical Guide to Support Vector Classification,” *BJU Int.*, vol. 101, no. 1, pp. 1396–1400, 2008.

[15] H. Omrani, “Predicting travel mode of individuals by machine learning,” *Transp. Res. Procedia*, vol. 10, no. July, pp. 840–849, 2015.

[16] A. Bolbol, T. Cheng, I. Tsapakis, and J. Haworth, “Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification,” *Comput. Environ. Urban Syst.*, vol. 36, no. 6, pp. 526–537, 2012.

[17] A. Jahangiri and H. Rakha, “Developing a Support Vector Machine (SVM) Classifier for Transportation Mode Identification by Using ...,” no. January, p. 14p, 2014.

[18] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[19] E. Alpaydin, “Introduction to Machine Learning,” *MIT Press*, 2014.

[20] S. H. Fang *et al.*, “Transportation modes classification using sensors on smartphones,” *Sensors (Switzerland)*, vol. 16, no. 8, pp. 1–15, 2016.

[21] D. Kim, S. Lee, and S. Cho, “Input vector normalization methods in support vector machines for automatic incident detection,” *Transp. Plan. Technol.*, vol. 30, no. 6, pp. 593–608, 2007.

[22] D. J. Bartholomew, “Principal components analysis,” *Int. Encycl. Educ.*, pp. 374–377, 2010.

[23] C. M. Bishop, *Pattern Recognition and Machine Learning*. 2006.

[24] J. P. Muller and L. Massaron, *Machine Learning for dummies*. 2016.

[25] D. Chicco, “Ten quick tips for machine learning in computational biology,” *BioData Min.*, vol. 10, no. 1, pp. 1–17, 2017.

[26] D. M. W. Powers, “Evaluation : From Precision , Recall and F-Factor to ROC , Informedness , Markedness & Correlation,” no. December, 2007.

[27] T. Fawcett, “An introduction to ROC analysis,” vol. 27, pp. 861–874, 2006.

[28] F. Zong, Y. Bai, X. Wang, Y. Yuan, and Y. He, “Identifying travel mode with GPS data using support vector machines and genetic algorithm,” *Inf.*, vol. 6, no. 2, pp. 212–227, 2015.

[29] G. Xiao, Z. Juan, and J. Gao, “Travel Mode Detection Based on Neural Networks and Particle Swarm Optimization,” pp. 522–535, 2015.

[30] C. Carpineti, V. Lomonaco, L. Bedogni, M. Di Felice, and L. Bononi, “Custom Dual Transportation Mode Detection by Smartphone Devices Exploiting Sensor Diversity,” pp. 1–15, 2018.

[31] S. Reddy, M. I. N. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, “Using Mobile Phones to Determine Transportation Modes,” vol. 6, no. 2, 2010.

[32] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, “Transportation Mode Detection using Mobile Phones and GIS Information,” no. July 2014, 2011.

[33] DataCamp, “Jupyter Notebook,” *DataCamp*, 2016.

[34] W. Mckinney and Pydata Development Team, “Pandas : powerful python data analysis toolkit release 0.13.1,” *Python Packag.*, p. 1211, 2014.

[35] NumPyCommunity, “NumPy User Guide,” pp. 1–135, 2016.

[36] A. Schnabel, K. Muthmann, and D. Ackner, “Cyface | Analyse von Verkehrs- & Straßenzustandsdaten.” .

[37] T. Feng and H. J. P. Timmermans, “Transportation mode recognition using GPS and accelerometer data,” *Transp. Res. Part C Emerg. Technol.*, vol. 37, pp. 118–130, 2013.

Annex

Annex A: Decision Tree Parameters

Parameter	***Selected***	***Description***
Class_weight	**None** (All clases have weight one)	Weight associated with classes
Criterion	**'gini'** (Impurity)	Measures the quality of a split
Max_depth	**None** (Nodes are expanded until all leaves are pure or until all leaves contain less than minimum samples split samples)	Maximum depth of the tree
Max_features	**None** (Maximum number of features = existing number of features)	The number of features to consider when looking for the best split
Max_leaf_nodes	**None** (Unlimited)	Number of leaf nodes
Min_impurity_decrease	**0.0**	A node will be split if this split induces a decrease of the impurity greater than or equal to this value (0.0)
Min_impurity_split	**None** (no early stopping)	Threshold for early stopping in tree growth
Min_samples_leaf	**1**	Minimum number of samples required to be a leaf node
Min_samples_split	**2**	Minimum number of samples required to split an internal node
Min_weight_fraction_leaf	**0.0** (Samples have equal weight)	Minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node
Presort	**False** (Not speeding up)	Whether to presort the data to speed up the finding of best splits in fitting
Random_State	**0** (random_state is the seed used by the random number generator)	Random Number Generator
Splitter	**'best'** (choosing best Split)	Strategy used to choose the split at each node

Annex B: Support Vector Machine Parameters

Parameter	***Selected***	***Description***
C	**1**	Penalty parameter C of the error term
Cache_size	**200**	Specify the size of the kernel cache (in MB)
Class_weight	**None** (all classes have weight one)	Set the parameter C of class i to class_weight[i]*C for SVC
Coef0	**0.0** (not significant for 'rbf' kernel)	Independent term in kernel function
Decision_function_shape	**'ovr'**	Return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers
Degree	**3** (ignored)	Degree of the polynomial kernel function ('poly')
Gamma	**'auto'** (uses 1/n_features)	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
Kernel	**'rbf'**	Specifies the kernel type to be used in the algorithm
Max_iter	**-1**	Hard limit on iterations within solver, or -1 for no limit
Probability	**False**	Whether to enable probability estimates
Random_state	**None (**the random number generator is the RandomState instance used by np.random)	The seed of the pseudo random number generator used when shuffling the data for probability estimates
Shrinking	**True**	Whether to use the shrinking heuristic
Tol	**0.001**	Tolerance for stopping criterion
verbose	**False**	Enable verbose output

Annex C: Neural Network MLP Parameters

Parameter	***Selected***	***Description***
activation	**‘relu’** (rectified linear function)	Activation function for the hidden layer
alpha	**0.0001**	Penalty parameter (regularization term)
Batch_size	‘auto’ (batch_size=min(200, n_samples)	Size of minibatches for stochastic optimizers
Beta_1	**0.9**	Exponential decay rate for estimates of first momen vector in adam, should be in [0,1)
Beta_2	**0.999**	Exponential decay rate for estimates of second moment vector in adam, should be in [0, 1)
Early_stopping	**False** (not using early stopping)	Whether to use early stopping to terminate training when validation score is not improving
Epsilon	**1e-0.8**	Value for numerical stability in adam
Hidden_layer_sizes	**(100,)**	The ith element represents the number of neurons in the ith hidden layer (tuple, length)
Learning_rate	**‘constant’** (constant learning rate given by ‘learning_rate_init’)	Learning rate schedule for weight updates
Learning_rate_init	**0.001**	The initial learning rate used. It controls the step-size in updating the weights
Max_iter	**350**	Maximum number of iterations. The solver iterates until convergence (determined by ‘tol’) or this number of iterations
Momentum	**0.9**	Momentum for gradient descent update. Should be between 0 and 1
Nesterovs_momentum	**True**	Whether to use Nesterov’s momentum
Power_t	**0.5**	The exponent for inverse scaling learning rate. Only used when solver=’sgd’
Random_state	**None** (Random State Instance used by np.random)	Random Number Generator

Shuffle	**True**	Whether to shuffle samples in each iteration
Solver	**‘adam’** (stochastic gradient-based optimizer)	The solver for weight optimization
Tol	**0.0001**	Tolerance for the optimization
Validation_fraction	**0.1**	The proportion of training data to set aside as validation set for early stopping. Must be between 0 and 1. Only used if early_stopping is True
Verbose	**False**	Whether to print progress messages to stdout
Warm_start	**False**	When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution