



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Real-Time Travel Mode Detection and Trip Purpose Prediction with Smartphone
Sensing and Machine Learning

Elton Figueiredo de Souza Soares

Orientador

Dr. Carlos Alberto Vieira Campos

Co-orientador

Dr. Sidney Cunha de Lucena

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2019

Real-Time Travel Mode Detection and Trip Purpose Prediction with Smartphone
Sensing and Machine Learning

ELTON FIGUEIREDO DE SOUZA SOARES

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO
DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFOR-
MÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO).
APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

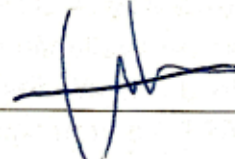
Aprovada por:



D.Sc. Carlos Alberto Vieira Campos — UNIRIO



D.Sc. Sidney Cunha de Lucena — UNIRIO



D.Sc. Carlos Eduardo Ribeiro de Mello — UNIRIO



Ph.D. Shahrokh Valaee — University of Toronto

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2019.

Catálogo informatizada pelo(a) autor(a)

F676 Figueiredo de Souza Soares, Elton
Real-Time Travel Mode Detection and Trip Purpose
Prediction with Smartphone Sensing and Machine
Learning / Elton Figueiredo de Souza Soares. -- Rio
de Janeiro, 2019.
143 f

Orientador: Carlos Alberto Vieira Campos.
Coorientador: Sidney Cunha de Lucena.
Dissertação (Mestrado) - Universidade Federal do
Estado do Rio de Janeiro, Programa de Pós-Graduação
em Informática, 2019.

1. Detecção de Modo de Transporte. 2.
Sensoriamento Móvel. 3. Mobilidade Inteligente. 4.
Sistemas de Transporte Inteligentes. 5. Aprendizado
de Máquina. I. Vieira Campos, Carlos Alberto,
orient. II. Cunha de Lucena, Sidney, coorient. III.
Título.

Dedico essa dissertação à todos que de alguma forma colaboraram para que eu concluísse este trabalho. Seja por terem colaborado diretamente com a realização da pesquisa, seja por terem colaborado indiretamente me ensinando, orientando, apoiando, incentivando e/ou inspirando com seus exemplos de dedicação, perseverança e sabedoria.

Agradecimentos

Agradeço a minha família, especialmente aos meus pais, Jussara e Ecio, por me darem a educação básica necessária para chegar a uma universidade e o apoio necessário para concluir mais esta etapa de minha formação acadêmica e profissional.

Agradeço a UNIRIO, ao corpo docente e servidores pela excelência e qualidade no ensino e serviços prestados.

Agradeço ao meu orientador, Dr. Carlos Alberto Vieira Campos, e co-orientador Dr. Sidney Cunha de Lucena pela paciência na orientação, dedicação e incentivo prestados durante mais esta etapa de minha jornada acadêmica.

Agradeço aos meus colegas de pós-graduação, em especial ao Helio de Paula Moura, Carlos Alvaro de Macedo Soares Quintella, Nicomar Fernandes de Oliveira e Tiago do Vale Saraiva, pela colaboração e apoio dispensados durante estes dois anos de convivência.

Agradeço ao professores membros da banca examinadora, Dr. Carlos Eduardo Ribeiro de Mello e Dr. Shahrokh Valaee, por aceitarem o convite e pela disposição em avaliar e contribuir com este trabalho.

Agradeço a todos os voluntários que participaram dos testes de campo da aplicação desenvolvida neste trabalho. Sem a colaboração de vocês, trabalhos como este não seriam possíveis.

Agradeço à minha eterna namorada e futura esposa, Carla Santos Franco, por estar ao meu lado durante esse desafio e por sempre me incentivar a seguir em frente.

Por fim, gostaria de agradecer à CAPES pelo apoio financeiro que me permitiu desenvolver uma pesquisa de mestrado de alta qualidade.

Soares, Elton Figueiredo de Souza **Real-Time Travel Mode Detection with Smartphone Sensing and Machine Learning**. UNIRIO, 2019. 143 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

A detecção dos modos dos transporte utilizados e a predição dos objetivos de viagem, através de dados de sensores de smartphones, surgiram como dois desafios de pesquisa nos últimos anos. Ambos os problemas foram profundamente investigados isoladamente, enquanto o problema de inferir modo e propósito ao mesmo tempo e, mais especificamente, usando o mesmo algoritmo de pré-processamento foi menos explorado. Além disso, poucos estudos apresentaram soluções capazes executar a detecção de modos de transporte do usuário em tempo real, e um conjunto ainda menor apresentou uma avaliação dessas soluções de maneira realista. Enquanto isso, alguns dos estudos anteriores afirmam as que as soluções de reconhecimento de atividades "de prateleira", não apresentam bom desempenho na tarefa de detecção do modo de transporte, embora muitos deles não apresentem uma evidência quantitativa de seu mau desempenho.

Assim, neste trabalho, foram propostas três técnicas de detecção de modo de transporte em tempo real, utilizando diferentes combinações de sensores de smartphone, e uma técnica para detecção conjunta de modo de transporte e predição de propósito da viagem usando um único algoritmo de pré-processamento, em tempo real. Nós avaliamos as técnicas propostas e uma solução de reconhecimento de atividades empiricamente, através de testes de campo e experimentos de validação cruzada com conjuntos de dados de mobilidade privados e públicos.

Palavras-chave: Detecção de Modo de Transporte, Sensoriamento Móvel, Mobilidade Inteligente, Sistemas de Transporte Inteligentes, Aprendizado de Máquina, Inteligência Artificial.

ABSTRACT

The detection of the travel modes used and, the prediction of trip purposes, through smartphone sensors data have emerged as two research challenges in recent years. Both of these problems have been deeply investigated in isolation, while the problem of inferring mode and purpose at the same time and, more specifically, using the same preprocessing algorithm has been less explored. Also, few studies presented solutions that can execute the detection of user travel modes in real-time, and even fewer have presented the evaluation of these solutions in a realistic manner. Meanwhile, some of the previous studies claim that off-the-shelf activity recognition solutions, do not perform well in the travel mode detection task, although many of them do not present a quantitative evidence of their bad performance.

Thus, in this work, we propose three techniques for real-time travel mode detection, using different combinations of smartphone sensors, and one technique for joint travel mode detection and trip purpose prediction using a single preprocessing algorithm, in real-time. We empirically evaluated the proposed techniques and an off-the-shelf activity recognition solution using field tests and cross-validation experiments with private and public mobility datasets.

Keywords: Travel Mode Detection, Mobile Sensing, Smart Mobility, Intelligent Transportation Systems, Machine Learning, Artificial Intelligence

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Justification	3
1.3	Objectives	3
1.3.1	Main Objective	3
1.3.2	Research Questions	4
1.4	Contributions	4
1.5	Structure	5
2	Theoretical Background	7
2.1	Feature Engineering Techniques for building Classification Models	8
2.1.1	Summary Statistics	8
2.1.1.1	Location	8
2.1.1.2	Spread	9
2.1.1.3	Shape	10
2.1.1.4	Dependence	11
2.1.2	Time and Frequency Domain	12
2.1.3	Principal Component Analysis	13

2.1.4	Recursive Feature Elimination	13
2.2	Machine Learning Techniques for building Classification Models	13
2.2.0.1	Logistic Regression	13
2.2.0.2	K-Nearest Neighbours	14
2.2.0.3	Naive Bayes	15
2.2.0.4	Decision Trees	15
2.2.0.5	Adaptive Boosting	16
2.2.0.6	Random Forest	17
2.2.0.7	Support Vector Machine	17
2.2.0.8	Neural Networks	18
2.2.0.9	Deep Neural Networks	19
2.2.0.10	Recurrent Neural Networks	20
2.2.1	Long-Short Term Memory	21
2.2.2	Gated Recurrent Unit	22
2.3	Automated Machine Learning Techniques for building Classification Models	22
2.3.1	Hyperparameter Optimization	22
2.3.1.1	Grid Search	23
2.3.1.2	Random Search	23
2.3.1.3	Bayesian Optimization	23
2.3.2	Combined Algorithm Selection and Hyperparameter Optimization	23
2.3.3	Global Optimization	24
2.4	Performance Metrics for evaluating Classification Models	25
2.4.1	Accuracy	25
2.4.2	Precision	26

2.4.3	Recall	26
2.4.4	F1-Score	26
2.4.5	Kappa Coefficient	26
3	Real-Time Travel Mode Detection with Location Sensors	27
3.1	Introduction	27
3.2	Related Works	28
3.3	Proposed Solution	31
3.4	Prototype Development	33
3.5	Prototype Evaluation	35
3.5.1	Methods	35
3.5.2	Data	36
3.5.3	Results	37
3.5.4	Discussion	40
3.6	ActivityRecognition API Evaluation	41
3.6.1	Methods	41
3.6.2	Data	45
3.6.3	Results	46
3.6.4	Discussion	47
3.7	Conclusion	47
4	Real-Time Travel Mode and Trip Purpose Prediction with Location Sensors	48
4.1	Introduction	48
4.2	Related Works	49
4.2.1	Travel Mode Detection	49
4.2.2	Trip Purpose Prediction	50

4.2.3	Joint Travel Mode and Trip Purpose Identification	51
4.3	Study Data	52
4.4	Proposed Solution	55
4.4.1	Preprocessing	56
4.4.2	Classification	60
4.5	Performance Evaluation	61
4.5.1	Evaluation Metrics	62
4.5.2	Proposed Solution with Random Search	62
4.5.3	Proposed Solution with Bayesian Optimization	62
4.5.4	Baseline Solution	63
4.6	Discussion	64
4.6.1	Travel Mode Detection	65
4.6.2	Trip Purpose Prediction	65
4.6.3	Time Window Size	66
4.7	Conclusions	66
5	Real-Time Travel Mode Detection with Multiple Sensors	68
5.1	Introduction	68
5.2	Related Works	70
5.3	Travel Mode Detection Technique used in a Public Benchmark Dataset .	73
5.3.1	Travel Mode Detection Technique	73
5.3.2	Public Benchmark Dataset	74
5.3.3	Discussion	75
5.4	Feature Engineering	77
5.5	Evaluation Experiments	77

5.5.1	Obtained results	80
5.5.2	Discussion	82
5.5.2.1	Classification Performance	82
5.5.2.2	Classification Cost	82
5.5.2.3	PCA Impact	83
5.6	Conclusion	84
6	Real-Time Travel Mode Detection with Recurrent Neural Networks	85
6.1	Introduction	85
6.2	Related Works	86
6.3	Proposed Method	89
6.3.1	Generic Framework for Travel Mode Detection	89
6.3.2	Online Travel Mode Detection with LSTM	90
6.3.3	Implementation Design	92
6.4	Experiments	93
6.4.1	Settings	94
6.4.2	Details of Training and Implementation	95
6.4.3	Results Analysis	96
6.4.3.1	Five Second Time Windows	97
6.4.3.2	One Second Time Windows	99
6.4.3.3	Ten Second Time Windows	101
6.4.3.4	Summary and Discussion	103
6.5	Conclusion	105
7	Conclusions and Future Work	107

List of Figures

2.1	Visual representation of a Logistic Regression model for travel mode classification. Adapted from https://commons.wikimedia.org	14
2.2	Visual representation of a KNN model for travel mode classification. Adapted from https://commons.wikimedia.org	15
2.3	Visual representation of a Decision Tree model for travel mode classification. Adapted from https://www.lucidchart.com	16
2.4	Visual representation of a Support Vector Machine model for binary travel mode classification using only Speed and Acceleration as input features. Adapted from https://commons.wikimedia.org	17
2.5	Visual representation of a Multilayer Neural Network model for travel mode classification with one hidden layer. Adapted from https://commons.wikimedia.org	18
2.6	Visual representation of a Deep Feedforward Neural Network model for travel mode classification with three hidden layers. Adapted from https://commons.wikimedia.org	20
2.7	Visual representation of a Recurrent Neural Network model for travel mode classification. Adapted from https://commons.wikimedia.org	21
2.8	Visual representation of a LSTM based RNN for travel mode classification. Adapted from https://commons.wikimedia.org	21
2.9	Visual representation of a GRU based RNN for travel mode classification. Adapted from https://commons.wikimedia.org	22

2.10 Automated Machine Learning workflow implemented in AutoSklearn. Diagram adapted from [48].	25
3.1 Proposed Real-Time Travel Mode Detection Solution. *MLP - Multilayer Perceptron, SVM - Support Vector Machine, BN - Bayesian Net, DT - Decision Table	32
3.2 CityTracks-RT application components.	34
3.3 CityTracks-RT application workflow.	35
3.4 Spatial visualization of users' displacement on the Rio de Janeiro metropoli- tan area during the use of CityTracks-RT application.	36
3.5 Locations captured x locations generated per travel mode (a). Locations captured x locations generated by precision range in meters (b). Cumula- tive frequency of locations collected by precision range (c).	38
3.6 Component diagram of the CityTracks-AWARE application.	43
3.7 Diagram of collection server components.	43
3.8 Sequence diagram of the processing of data collected by the sensors in the CityTracks-AWARE app when sending to the server (Before).	44
3.9 Sequence diagram of the processing of data collected by the sensors in the CityTracks-AWARE app when sending to the server (After).	45
3.10 Inferences made by travel mode collected.	46
4.1 Frequency of location traces collected by travel mode (a) and trip purpose (b).	53
4.2 Projection of the location traces coordinates on the Rio de Janeiro metropoli- tan area (a) and Downtown region (b) map.	55
4.3 Overview of travel mode detection and trip purpose prediction solution.	55
4.4 Locations frequency distribution by measurement precision: (a) All lo- cation traces, (b) Locations within 200 meters threshold, (c) Locations within 100 meters threshold, (d) Locations within 50 meter threshold (e), Locations within 25 meters threshold (f), Locations within 12.5 meters threshold (g).	57

4.5	Scatter plot for travel mode relationships with maximum calculated acceleration (a) and mean measured speed (b). Scatter plots for trip purpose relationships with travel mode (c) and weekday (d).	59
4.6	Steps used to find the best classifiers for travel mode detection and trip purpose prediction.	60
4.7	Performance metrics per window size for travel mode detection (a) and trip purpose (b).	67
5.1	Preprocessing steps of US-TransportationMode travel mode detection technique. Adapted from http://cs.unibo.it/projects/us-tm2017/	74
5.2	Histogram of total number of samples collected by travel mode (a) and user (b).	75
6.1	Proposed offline training and offline/online detection mechanism.	90
6.2	TMD-LSTM model architecture.	91
6.3	Travel Mode Inference using a trained TMD-LSTM.	92
6.4	Cloud-based and In-device real-time travel mode detection with TMD-LSTM.	93
6.5	Sample distributions per travel mode (a) and user (b).	94
6.6	Evaluation experiments process overview. Window size = 1 second, #S = 80566; Window size = 5 seconds, #S = 16041; Window size = 10s, #S = 7967.	95
6.7	RNNs accuracy (a) and size (b) obtained for multiple hyperparameter configurations, using 5 second time windows.	98
6.8	(D)FNNs accuracy (a) and size (b) obtained for multiple hyperparameter configurations, using 5 second time windows.	99
6.9	RNNs accuracy obtained for multiple hyperparameter configurations, using 1 second time windows.	100
6.10	(D)FNNs accuracy obtained for multiple hyperparameter configurations, using 1 second time windows.	101

6.11 RNNs accuracy obtained for multiple hyperparameter configurations, using 10 second time windows.	102
6.12 (D)FNNs accuracy obtained for multiple hyperparameter configurations, using 10 second time windows.	103
6.13 Accuracy and model size of best performing configurations of each ML technique for each window size.	103
6.14 Average training and testing confusion matrices per-fold of best performing neural network configurations with 5, 1 and 10 second time windows. Predicted labels are represented as columns and true labels are represented as rows.	106

List of Tables

3.1	Summary of related works on travel mode detection through smartphone sensors. *Acc - Acelerometer, Gyr - Gyroscope, Mag - Magnetometer, Cellular - Cellular Networks	31
3.2	Data collection summary statistics per user. *#H - Hours of data collection, #C - Number of chunks collected, #L - Number of locations collected, MP - Mean location precision, SD - Precision standard deviation, %C - Percentage of locations captured via FusedLocationAPI, %G - Percentage of locations generated by interpolation.	39
3.3	Confusion matrix of the motorized and non-motorized chunk classification for the SVM classifier.	39
3.4	Confusion matrix of motorized mode classification for the Decision Table classifier.	39
3.5	Confusion matrices of non-motorized chunk classification for Decision Table and Bayesian Net classifiers.	40
3.6	Confusion matrix of walking, biking, bus and car classification with Multilayer Perceptron.	40
3.7	Summary of mean per class accuracy, precision, recall and F1-score of each classifier.	40
3.8	Sensors supported by the CityTracks-AWARE application and their minimum sampling frequencies. * Sampling frequency does not apply because these sensors capture the data reactively.	42
3.9	Data collected by device.	45

3.10	Confusion matrix grouping the data collected within the five activity classes recognized by the ActivityRecognition API.	46
3.11	Performance metrics for each activity class recognized by the ActivityRecognition API. *TP - True Positives, FP - False Positives.	46
4.1	Descriptive statistics of location traces by device.	54
4.2	Number of location samples removed by each filtering technique.	56
4.3	Pearson correlation tests of chunk categorical attributes.	58
4.4	Performance metrics of the best classifiers found through Random Search on the test set for travel mode detection.	62
4.5	Performance metrics of the best classifiers found through Random Search on the test set for trip purpose prediction.	62
4.6	Performance metrics of the best classifiers found through Bayesian Opt. on the test set for travel mode detection.	63
4.7	Performance metrics of the best classifiers found through Bayesian Opt. on the test set for trip purpose prediction.	63
4.8	Average per fold performance metrics of travel mode detection on the 10-fold cross-validation.	63
4.9	Average per fold performance metrics of trip purpose prediction on the 10-fold cross-validation.	64
4.10	10-fold cross-validation performance metrics of the classifiers used in the CityTracks-RT application.	64
4.11	Performance metrics of the classifiers used in the field tests of CityTracks-RT application.	64
4.12	Confusion matrix of best performing travel mode classifier.	65
4.13	Confusion matrix of best performing trip purpose classifier.	66
5.1	Summary of related works.	73
5.2	Sensors considered on each Sensor Set.	75

5.3	US-TransportationMode original performance evaluation results.	76
5.4	Sensors on each cross-validation experiment and respective features extracted from 5 second time windows.	78
5.5	Hyperparameter configurations for each machine learning algorithm used for each sensor set.	79
5.6	Performance metrics of all machine learning algorithms for scenarios 1, 2 and 3. *DT - Decision Tree, RF - Random Forest, SVM - Support Vector Machine, NN - Neural Networks, ASE - AutoSklearn Ensemble.	80
5.7	Performance metrics of all machine learning algorithms for scenarios 4, 5 and 6.	81
5.8	Performance metrics of all machine learning algorithms for scenarios 7, 8 and 9.	81
5.9	Performance metrics of all machine learning algorithms for scenarios 10, 11 and 12.	81
5.10	Mean split fit and score time for each scenario.	84
6.1	Summary of related works. *GPS - Global Positioning System, CDR - Call Data Records **DL Model - Deep Learning Model, CNN - Convolutional Neural Networks, DNN - Deep Neural Networks, LSTM - Long-Short Term Memory, IO-HMM - Input Output Hidden Markov Models, RNN - Recurrent Neural Networks, SAE - Stacked Autoencoders, CGRNN - Control Gate based Recurrent Neural Network	87
6.2	Best features selected from the multiple sensors.	96
6.3	Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 5 second time windows.	97
6.4	Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 1 second time windows.	99

6.5 Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 10 second time windows. 101

1. Introduction

Population growth in urban centers presents many challenges in the development of smarter cities [25]. Urban sensing research has tried to provide means for urban planners to understand cities dynamics, in order to address those challenges, through optimized provision of public services and infrastructure management. Another research field that has a similar objective, with special focus on the mobility component of smart cities [57], is the field of Intelligent Transportation Systems (ITS).

Smartphone popularization has enabled the development of many ITS solutions based on participatory and opportunistic sensing [78]. Its growing set of sensors, computation power and battery autonomy allow the continuous sensing of citizens mobility behaviour, surrounding environment and social interactions. This allowed the emergence of innovative applications based on contextual information (*i.e.*, context-aware applications) extracted by specialized software from sensory data (*i.e.*, virtual sensors [71]).

The identification of users' travel modes (*i.e.*, modes of transportation) through their smartphone sensors is a growing topic of research, with many applications in the field of Smart Mobility and Internet of Things (IoT), specially with respect to ITS [126]. With a dataset of citizens travel modes in hand, governments are allowed to better understand the usage of transportation infrastructures, where knowledge of mobility patterns among citizens can be useful for predicting demand over modes of transportation in urban centers [107]. It also allows retailers and other individual consumer-focused businesses to better characterize their customers and correlate their consumption patterns with mobility behaviour, in addition to other existing contextual information, like socioeconomic attributes.

In particular, online (*i.e.*, *real-time*) travel mode detection can provide context-awareness, which is useful for Location-Based Services (LBS) [106], in order to customize information delivery based on users' needs and possibilities of interaction. It may apply as well

to the use cases described for offline detection, depending on the frequency in which the travel mode information needs to be available for the final application.

A fully autonomous ITS, for example, may utilize citizens travel mode information to increase the number of buses circulating on certain parts of the city in high-demand periods and decrease it when the demand for this transportation mode is low. If one uses offline travel mode detection for this task, it would be able to adapt the system based on historical patterns, such as specific hours of the day when people tend to move on and off to work. However, if online travel mode detection is employed, the system would be able to automatically respond to more infrequent events that may increase or decrease demand for a specific mode of transportation, such as a sports event or a music concert, for example.

1.1 Motivation

Among the several features that may be used to describe mobility patterns, the travel mode and trip purpose specifically contribute to their better understanding, thus benefiting the generation of automatic (or semi-automatic) travel diaries [100], recommendation systems, personal assistants and other mobile data collection applications [117, 118].

Detecting which mode of transportation and trip purpose are being used by citizens through their smartphone sensors is a hard task, if we consider all the range of possible transportation modes and trip purposes available in urban centers. There are many commercial solutions that tackle instances of these problems using historical information from user mobility behaviour and/or rule-based classification schemes, like Google Maps Timeline ¹, for example.

These solutions do not provide, however, a generic technique that can be used to infer the mode and the purpose of user trips using only a small amount of user data and without requiring a connection to a cloud server to process the information captured through smartphone sensors. Besides, most of these solutions are able to classify only a subset of all the modes and purposes that can be inferred from user mobility data, limiting their usefulness for a broader range of use cases.

¹<https://www.google.com.br/maps/timeline>

1.2 Justification

The detection of the travel modes used and, the prediction of trip purposes, through smartphone sensors data have emerged as two research challenges in recent years. Both of these problems have been deeply investigated separately, while the problem of inferring mode and purpose at the same time and, more specifically, using the same preprocessing algorithm has been less explored.

Also, few studies presented solutions that can execute the detection of user travel modes in real-time, and even fewer have presented the evaluation of these solutions in a realistic manner. Meanwhile, some of the previous studies claim that off-the-shelf activity recognition solutions, do not perform well in the travel mode detection task, although many of them do not present a quantitative evidence of their bad performance.

1.3 Objectives

This section presents this work main objective and the research questions that will be addressed in the following chapters of this dissertation.

1.3.1 Main Objective

This work's main objective is to develop solutions for the problems of real-time travel mode detection and trip purpose prediction. These solutions must be able to identify the mode of transportation and the purpose of the trips made by smartphone users, in real-time, with a high accuracy and low resource consumption.

The accuracy of this identification is essential for the context-aware applications that might use these contextual-information for delivering customized messages and services to the user. Also, ITS applications may use the inferred travel modes and trip purposes for infrastructure management and adaptation, which makes precision and recall of those inferences crucial for their success.

The low resource consumption is another critical requirement of the ideal solution, because it cannot consume a high proportion of the computational resources available on the smartphone, as these are already disputed by the other applications and services. Besides, smartphone resources might be very limited, depending on the model and vendor. Therefore, the optimal solution should be able to run on any smartphone device compatible with the context-aware applications it might serve.

1.3.2 Research Questions

The present work will try to answer the following research questions:

1. Can off-the-shelf activity recognition solutions perform real-time travel mode detection satisfactorily with regards to accuracy, precision and recall of the inferences made?
2. Is detecting a person's mode of transportation through statistical features extracted from time series of smartphone location sensors measurements, in real-time, truly possible?
3. How can we detect the mode of transportation and the purpose of trips, in real-time, using statistical features extracted from time series of smartphone location sensors measurements?
4. Can automated machine learning help us generate more accurate classification models for real-time travel mode detection based on smartphone sensor measurements?
5. Does the use of dimensionality reduction techniques upon the features extracted from the time series of multiple smartphone sensor measurements reduce the cost of training and running travel mode classifiers without substantially reducing accuracy, precision and recall?
6. Do higher order statistical features of the time series of smartphone sensors measurements, allow a better identification of travel modes, in real-time, with regards to accuracy, precision and recall?
7. Can recurrent neural networks be used to generate more efficient real-time travel mode detection models than other state-of-the-art machine learning techniques, with regards to classification accuracy and model size trade-off?

1.4 Contributions

The main contributions of this work are:

1. A real-time travel mode detection technique that uses features extracted from time series of smartphone GPS, WiFi and Cellular Network sensors measurements and supervised machine learning algorithms to train travel mode classification models

that rely only on smartphone hardware resources, without the need of internet connectivity.

2. A prototype implementation of the proposed technique as a mobile application on Android platform, namely CityTracks-RT, and the experimental analysis of in the field performance of the proposed solution in the metropolitan region of Rio de Janeiro.
3. A mobile application for multi-sensor data collection and real-time travel mode detection, namely CityTracks-AWARE, using an off-the-shelf activity recognition solution and the experimental analysis of in the field performance of the solution in the metropolitan area of Rio de Janeiro.
4. A method for building real-time travel mode detection and trip purpose prediction models using features extracted from time series of smartphone location measurements and the experimental analysis of the proposed method based on quantitative metrics obtained through cross-validation using real smartphone data collected during CityTracks-RT evaluation experiments.
5. An experimental analysis of the improvement obtained in real-time travel mode detection performance, when using automated machine learning techniques, based on quantitative metrics obtained through cross-validation using real smartphone data from a public travel survey dataset, namely TMDataset [26].
6. A real-time travel mode detection method based on recurrent neural networks and features extracted from time series of multiple smartphone sensor measurements, namely TMD-LSTM, and the experimental analysis of the proposed technique based on quantitative metrics obtained through cross-validation using real smartphone data from TMDataset [26].

1.5 Structure

Chapter 2 is an overview of the main theoretical background regarding the classification models used for travel mode detection and trip purpose prediction, the machine learning algorithms used to build these models, the engineering techniques used to generate features for training these models and the evaluation metrics used for evaluating these models. Chapter 3 answers research questions 1 to 2 and presents the contributions 1 to 5 of this work. Chapter 4 answers research question 3 and presents the contributions 6 to 7. Chapter 5 answers research questions 4 to 6 and presents the contributions 8 to 9.

Chapter 6 answers research question 7 and presents the contributions 10 to 12. Finally, chapter 7 presents the main conclusions and suggestions of future work.

2. Theoretical Background

The task of detecting the travel modes people use in their daily commute, is fundamentally a classification task, in which a computer system must classify which mode of transportation people are using given a set of observations from smartphone sensors, GPS logs or any other source of data. Therefore, in order to build a computer system that is able to perform this task, one must model the patterns inherent to the observations of each transportation mode in order to correctly identify them. During a long time, this modelling was done through manual data analysis or using rule extraction techniques. However, with the advancements in machine learning, more specifically statistical learning, automatic generation of these models based on observations is now possible and feasible. In this work, we explore the use of machine learning techniques for building real-time travel mode detection models.

Therefore, in the following sections we present the main Feature Engineering, Machine Learning and Automated Machine Learning techniques used for building those models as well as the main performance metrics used in their evaluation and comparison. As a disclaimer, the terms online detection and real-time detection are used in the remainder of this work to refer to the process of inferring the modes of transportation in the shortest period of time possible, which should be not be higher than a couple of minutes given the state-of-the-art solutions [126].

It is important to clarify that every supervised machine learning model can perform online classification, as it attributes labels to unseen samples that are fed into the model. Therefore, the main difference between offline and online travel mode detection is the fact that typical offline approaches will process a large amount of travel data to identify each individual trip and tripeg [100] in order to classify which travel modes were used on a whole trip or tripeg, whose duration can range from minutes to hours, or even days. In opposition to that, online detection approaches must be capable of inferring the modes of transportation used in segments or instants of a tripeg. Online detection, however,

should not be confused with online training [113] of machine learning models, as the latest requires the classification model to be updated at every new sample it classifies. Online training is also a desired feature for travel mode detection models, but it involves additional concerns with regards to catastrophic forgetting [61] and concept drift problems [93].

Provided that, we will use the terms online detection and real-time detection to refer to solutions where a machine learning model is trained once with a static dataset, and then used to perform the classification of travel modes used in small segments or instants of trips, based on features extracted from short time series of smartphone sensor measurements.

2.1 Feature Engineering Techniques for building Classification Models

One of the main challenges in solving problems with the use of machine learning algorithms is extracting the most relevant features from the raw data in order to train a classification or prediction model. This challenge is addressed by a process called Feature Engineering, and involves construction, combination, transformation and selection of features [85]. This often requires domain specific knowledge and might be more relevant to the final model performance than the machine learning algorithm itself [35].

2.1.1 Summary Statistics

In descriptive statistics, summary statistics are used to summarize a set of observations, in order to communicate the largest amount of information as simply as possible [103]. Observations can be summarized in terms of:

- location *i.e.*, central tendency
- spread *i.e.*, statistical dispersion
- shape *i.e.*, shape of the distribution
- dependence *i.e.*, statistical dependence

2.1.1.1 Location

In summary statistics, the main measures of location are the arithmetic mean, median, mode, and interquartile mean. The arithmetic mean, or average, is the sum of a set of

numbers divided by the count of numbers in the set [68]. The median is the value separating the higher half from the lower half of a data sample (a population or a probability distribution). For a data set, it may be thought of as the “middle” value [86]. The mode of a set of data values is the value that appears most often. It is the value x_i at which its probability mass function takes its maximum value. In other words, it is the value that is most likely to be sampled [63]. The interquartile mean is a statistical measure of central tendency based on the truncated mean of the interquartile range [137].

Formally, given a sample of numerically ordered observations x_1, x_2, \dots, x_n , where n is the number of observations. The mean is defined by the Equation 2.1:

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} \quad (2.1)$$

If n is odd, this median is defined by Equation 2.2:

$$M = x_{(n+1)/2} \quad (2.2)$$

Otherwise, it is defined by Equation 2.3:

$$M = \frac{x_{n/2} + x_{n/2+1}}{2} \quad (2.3)$$

The interquartile mean is defined by Equation 2.4:

$$x_{IQM} = \frac{2}{n} \sum_{i=\frac{n}{4}+1}^{\frac{3n}{4}} x_i \quad (2.4)$$

2.1.1.2 Spread

In summary statistics, the main measures of spread are the standard deviation, variance, range, interquartile range, absolute deviation and mean absolute difference [68]. The standard deviation is a measure that is used to quantify the amount of variation or dispersion of a set of data values [17]. A low standard deviation indicates that the data points tend to be close to the mean of the set, while a high standard deviation indicates that the data points are spread out over a wider range of values. The variance is the expectation of the squared deviation of a random variable from its mean. It measures how far a set of numbers are spread out from their average value [68]. The range of a set of data is

the difference between the largest and smallest values [145]. The interquartile range is the difference between 75th and 25th percentiles, or between upper and lower quartiles [137]. The absolute deviation of an element of a data set is the absolute difference between that element and a given point. Typically the deviation is reckoned from the central value, being construed as some type of average, most often the median or sometimes the mean of the data set [17]. The mean absolute difference is to the average absolute difference of two independent values drawn from a probability distribution [68].

Formally, given a sample of observations x_1, x_2, \dots, x_n , where n is the number of samples and \bar{x} is the mean, as defined by Equation 2.1. The standard deviation is defined by Equation 2.5:

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \quad (2.5)$$

The variance is defined by Equation 2.6:

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1} \quad (2.6)$$

The mean absolute deviation is defined by Equation 2.7:

$$\text{MAD} = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}| \quad (2.7)$$

The mean absolute difference is defined by Equation 2.8:

$$\text{MD} = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n |x_i - x_j| \quad (2.8)$$

2.1.1.3 Shape

In summary statistics, the main measures of the shape of a distribution are skewness and kurtosis. Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable with respect to its mean value. The skewness value can be positive or negative [20]. When skewness is positive, the shape of the probability distribution is right-skewed with respect to its mean, and if skewness is negative, it is left-skewed. Meanwhile, the kurtosis measure shows how different from a normal probability

distribution the random variable distribution is. When it is positive, the distribution has a sharper peak around the mean and a heavier tail. When it is negative, it has a flatter top and decays more abruptly. The kurtosis measure also provides some information about the tail of the distribution. If it is very large and positive, it indicates that the distribution has a long tail [20].

Formally, given a sample of observations x_1, x_2, \dots, x_n , where n is the number of samples and $\mu = \bar{x}$ is the mean, as defined by Equation 2.1. The skewness is defined by Equation 2.9 [41]:

$$sk = \frac{\mu^3}{\mu_2^{3/2}} \quad (2.9)$$

The kurtosis is defined by Equation 2.10 [41]:

$$ku = \frac{\mu^4}{\mu_2^2} \quad (2.10)$$

2.1.1.4 Dependence

The mutual information of two random variables is a measure of the mutual dependence between the two variables. More specifically, it quantifies the “amount of information” (in units such as shannons, commonly called bits) obtained about one random variable through observing the other random variable. The concept of mutual information is intricately linked to that of entropy of a random variable, a fundamental notion in information theory that quantifies the expected “amount of information” held in a random variable [33].

Formally, the mutual information of two discrete random variables X and Y can be defined by Equation 2.11:

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \quad (2.11)$$

where $p(x,y)$ is the joint probability function of X and Y , and $p(x)$ and $p(y)$ are the marginal probability functions of X and Y , respectively.

In the case of continuous random variables, the mutual information is defined by Equ-

tion 2.12:

$$MI(X;Y) = \int_{y \in Y} \int_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) dx dy \quad (2.12)$$

2.1.2 Time and Frequency Domain

Time domain is the analysis of mathematical functions, physical signals or time series of economic or environmental data, with respect to time. In the time domain, the signal or function's value is known for all real numbers, for the case of continuous time, or at various separate instants in the case of discrete time [82].

In electronics, control systems engineering, and statistics, the frequency domain refers to the analysis of mathematical functions or signals with respect to frequency, rather than time. In summary, a time-domain graph shows how a signal changes over time, whereas a frequency-domain graph shows how much of the signal lies within each given frequency band over a range of frequencies. A frequency-domain representation can also include information on the phase shift that must be applied to each sinusoid in order to be able to recombine the frequency components to recover the original time signal [23].

A given function or signal can be converted between the time and frequency domains with a pair of mathematical operators called transforms. An example is the Fourier transform, which converts a time function into a sum or integral of sine waves of different frequencies, each of which represents a frequency component. The "spectrum" of frequency components is the frequency-domain representation of the signal. The inverse Fourier transform converts the frequency-domain function back to the time function [23].

In using the Laplace, Z-, or Fourier transforms, a signal is described by a complex function of frequency: the component of the signal at any given frequency is given by a complex number. The magnitude of the number is the amplitude of that component, and the angle is the relative phase of the wave. For example, using the Fourier transform, a sound wave, such as human speech, can be broken down into its component tones of different frequencies, each represented by a sine wave of a different amplitude and phase. The response of a system, as a function of frequency, can also be described by a complex function. In many applications, phase information is not important. By discarding the phase information it is possible to simplify the information in a frequency-domain representation to generate a frequency spectrum or spectral density [23].

2.1.3 Principal Component Analysis

Principal Component Analysis (PCA) is a multivariate technique used to decompose and transform a multivariate dataset into a set of successive orthogonal components ordered by the amount of variance of each component [143]. In other words, PCA is a linear transformation of a feature set often used as a dimensionality reduction algorithm. It transforms features of the data into orthogonal vectors called principal components. The idea is to represent high-dimensional data on a low-dimensional subspace while maintaining most of the variation in the data. The principal components are the orthogonal vectors that span this subspace [90].

2.1.4 Recursive Feature Elimination

Recursive Feature Elimination (RFE) [90] is an algorithm used to determine the top untransformed features of a dataset. It is a form of backward elimination, alliteratively removing features until an optimal number is found. First, the algorithm partitions the data into training and testing sets via resampling. The algorithm creates a classification model using the full set of n predictors on the training set and uses this model to predict the test set. Each predictor is ranked according to its contribution to improving the accuracy of the model. Next, it creates a subset composed of the top j features for all integer values of j such that $j \leq n$. Each subset is used to create another classification model employing only the top j features, and this model is adopted to predict the test set. This entire process is repeated until the desired number of folds for cross-validation is reached. After testing all values of j , the algorithm identifies the value with the highest average prediction accuracy. Label this value k (the optimal number of features) and return the top k features [90].

2.2 Machine Learning Techniques for building Classification Models

In order to automatically derive travel mode classification models from smartphone sensor observations, machine learning techniques are used in conjunction with the feature engineering techniques described previously. In this section we present a summary of the main machine learning techniques used in this work.

2.2.0.1 Logistic Regression

The logistic model is a widely used statistical model that uses a logistic function to model a binary dependent variable. Logistic regression technique through which one esti-

mates the parameters of a logistic model. A logistic model has a dependent variable with two possible values, such as stationary/non-stationary or in-vehicle/on-foot. These are represented by an indicator variable, where the two values are labelled “0” and “1”. In the logistic model, the the logarithm of the odds (log-odds) for the value labelled “1” is a linear combination of one or more independent variables, called predictors. The independent variables can each be a binary variable or a continuous variable. The corresponding probability of the value labelled “1” can vary between 0 and 1 (certainly the value “1”). The function that converts log-odds to probability is the logistic function. The unit of measurement for the log-odds scale is called a logit, from logistic unit. The defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each dependent variable having its own parameter [69]. Figure 2.1 presents illustrates how the logistic function can used to differentiate modes of transportation.

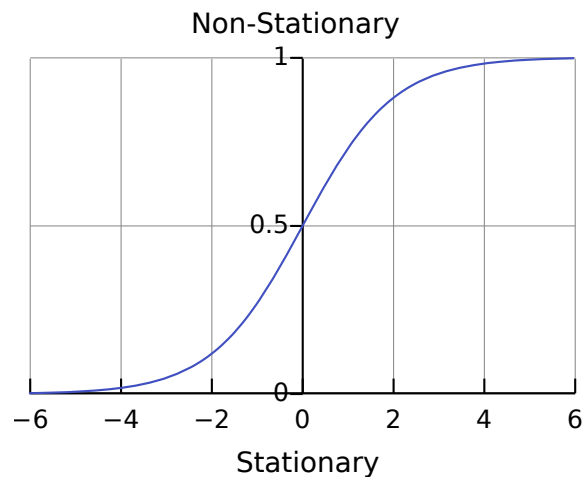


Figure 2.1: Visual representation of a Logistic Regression model for travel mode classification. Adapted from <https://commons.wikimedia.org>.

2.2.0.2 K-Nearest Neighbours

The k-nearest neighbours algorithm (KNN) is a non-parametric method used for classification and regression. In classification, the input consists of the K closest training examples in the feature space and the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its K nearest neighbours. If $K = 1$, then the object is simply assigned to the class of that single nearest neighbour. KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm is among the simplest of all machine learning algorithms. The neighbours are taken from a set of objects for which the class

is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the KNN algorithm is that it is sensitive to the local structure of the data [5]. Figure 2.2 illustrates the classification of samples using a KNN model.

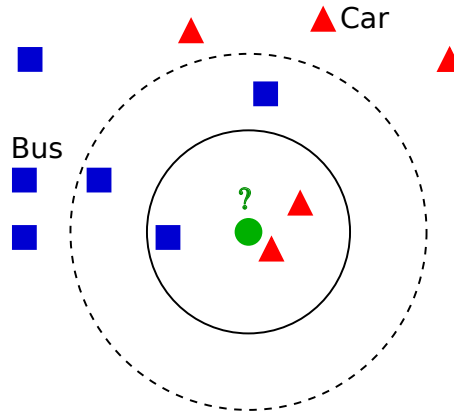


Figure 2.2: Visual representation of a KNN model for travel mode classification. Adapted from <https://commons.wikimedia.org>.

2.2.0.3 Naive Bayes

Naive Bayes (NB) classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. NB has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s and remains a popular (baseline) method for text categorization. NB classifiers are highly scalable, requiring a number of parameters linear in the number of variables in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers. In the statistics and computer science literature, NB models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but NB is not necessarily a Bayesian method [112].

2.2.0.4 Decision Trees

Decision Trees are one of the oldest and simplest machine learning techniques. They can be used for both regression and classification problems and their greatest advantage compared to more modern techniques is their greater interpretability, since their visual representation can be used to support domain expert's decision making [69]. Figure 2.3 presents a visual representation of this model.

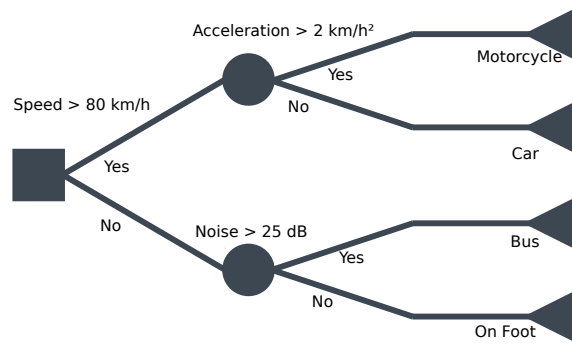


Figure 2.3: Visual representation of a Decision Tree model for travel mode classification. Adapted from <https://www.lucidchart.com>.

The training of a decision tree consists of partitioning the samples from a database into regions. These partitions are usually done through an algorithm called recursive binary splitting. Binary because at each iteration the algorithm divides a region in two and recursive because each subregion is partitioned recursively until a stop condition is reached, such as predetermined maximum number of leaves of the tree or maximum height [69]. This partitioning of the data is done through the values of predictor attributes. The choice of the predictor attributes used in the partitioning is done through the Gini index, entropy or information gain metrics [69]. Once the tree is trained the inferences are made through the average value of the samples (Regression) or the most frequent class (Classification) of each leaf.

2.2.0.5 Adaptive Boosting

Adaptive Boosting or Ada Boost (AB) is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms, namely weak learners, is combined into a weighted sum that represents the final output of the boosted classifier. AB is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner. Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a database. In that sense, AB with decision trees as the weak learners is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AB algorithm about the

relative "hardness" of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples [50].

2.2.0.6 Random Forest

The Random Forest technique is an evolution of decision trees and usually presents higher accuracy due to the use of the Bagging technique in which multiple trees are constructed from different partitions of the database. During the construction of the trees, only a subset of the predictive attributes are considered, which forces a greater variation in the generated trees and reduces the variance of the inferences [69].

The main drawbacks of this technique compared to traditional decision trees are the total loss of interpretability of results and high computational cost for model training. However, in several scenarios this technique has been applied with great success [69].

2.2.0.7 Support Vector Machine

Support Vector Machine is a machine learning technique that attempts to generate a mathematical model capable of dividing the dataset samples into two classes. This mathematical model takes the form of a hyperplane since the number of dimensions of this hyperplane will be equivalent to the number of predictor attributes used. Variations of this technique can be used to for multi-class classification and regression problems [69]. Figure 2.4 presents a visual representation of this model.

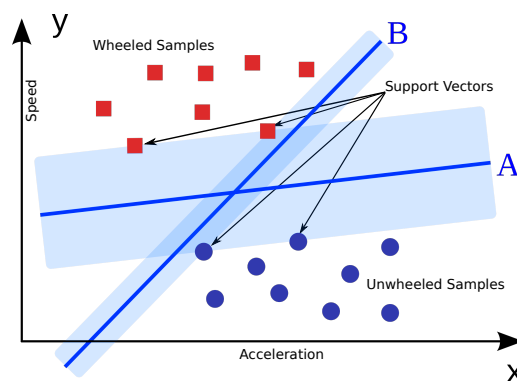


Figure 2.4: Visual representation of a Support Vector Machine model for binary travel mode classification using only Speed and Acceleration as input features. Adapted from <https://commons.wikimedia.org>.

In addition, more sophisticated forms of the Support Vector Machine use kernel functions to separate non-linearly separable samples. For this, the number of sample sizes is increased by combining values of the original attributes [69].

The construction of the hyperplane is based on the points closest to the boundary between the classes, which are called support vectors. Once these points are found, the maximum margin can be established between the samples of each class [69].

The original application of this technique is in problems of binary classification, however, evolutions of the same can be used in multi-class classification problems and even in regression problems [69].

2.2.0.8 Neural Networks

Neural Networks are one of the most popular and most powerful machine learning techniques. They are inspired on the functioning of the human brain and are considered an universal approximator because of their ability to map nonlinear functions using layers of neurons that compute the weighted sum of their inputs applied to an activation function and whose weights are optimized based on a cost metric, such as error rate or accuracy [37]. Figure 2.5 presents a visual representation of this model.

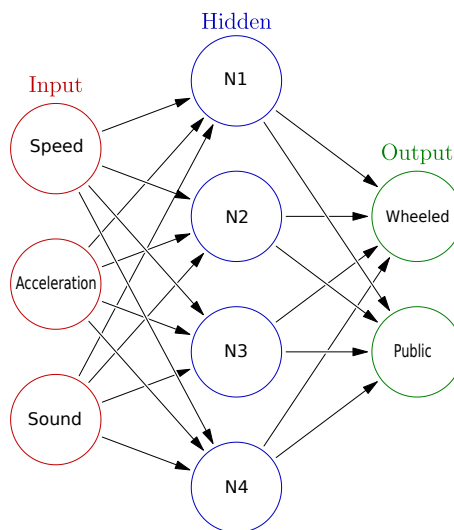


Figure 2.5: Visual representation of a Multilayer Neural Network model for travel mode classification with one hidden layer. Adapted from <https://commons.wikimedia.org>.

Feedforward Neural Networks (FNNs) are based on a logical structure called Perceptron or neuron. The Perceptron consists of a function that performs the weighted sum of input variables. To the result of this sum, a bias value is added and an output function is applied which transforms the result of this sum into a value between 0 and 1 or -1 and 1 [37].

A Multilayer Neural Network (Multilayer Perceptron) may have two or more layers, where the first layer is called a Single-Layer Neural Network and the last layer is called

the output layer. The intermediate layers are called hidden layers and have a key role in generating more complex models [37].

The most common training algorithm of a neural network is the Back-Propagation where the training is performed in iterations in which the error of the inferences of each neuron is evaluated and the weights of its input are adjusted in order to reduce the error in the next iteration [37].

The main disadvantages of neural networks is their long training time, especially in networks with many hidden layers, and their high propensity to overfitting which is what happens when a model fits so much to training data that its inferences only make sense for training samples [37].

To avoid this and other neural network problems, model validation techniques such as a test dataset or cross validation are required. The adjustment of the model is done through its hyperparameters, such as number of hidden layers, number of iterations, learning rate, momentum among others [37].

2.2.0.9 Deep Neural Networks

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers [14]. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives [132]. The extra layers enable composition of features from lower layers, potentially modelling complex data with fewer units than a similarly performing shallow network [14].

Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network didn't accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data [60]. Figure 2.6 presents a

visual representation of a deep feedforward neural networks for travel mode classification.

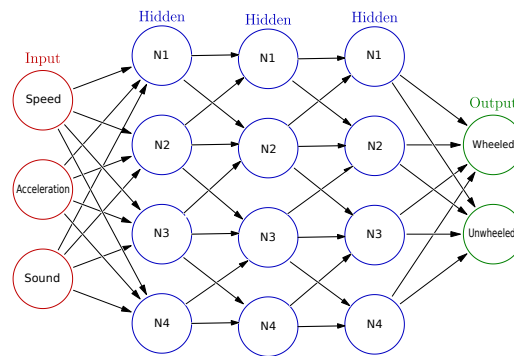


Figure 2.6: Visual representation of a Deep Feedforward Neural Network model for travel mode classification with three hidden layers. Adapted from <https://commons.wikimedia.org>.

2.2.0.10 Recurrent Neural Networks

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behaviour for a time sequence. Unlike feedforward neural networks, RNN can use their internal state (*i.e.*, memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition [16] or speech recognition [114].

A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled. Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks and gated recurrent units [60]. Figure 2.7 presents a visual representation of this model.

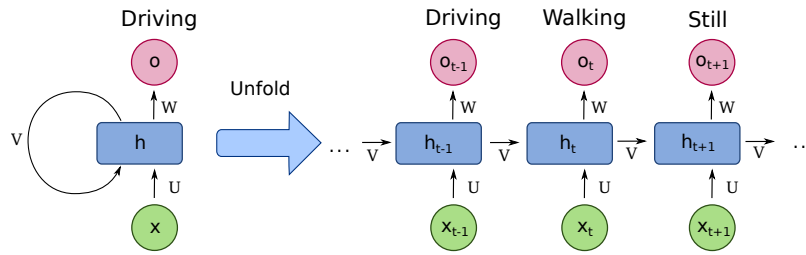


Figure 2.7: Visual representation of a Recurrent Neural Network model for travel mode classification. Adapted from <https://commons.wikimedia.org>.

2.2.1 Long-Short Term Memory

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing/-exploding gradient problem through the use of recurrent gates called “forget” gates [55]. LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. LSTM works even given long delays between significant events and can handle signals that mix low and high frequency components , Many applications use stacks of LSTM RNNs [46] and train them by Connectionist Temporal Classification (CTC) [62] to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences. LSTM can learn to recognize context-sensitive languages unlike previous models based on hidden Markov models [130] and similar concepts. Figure 2.8 presents a visual representation of this model.

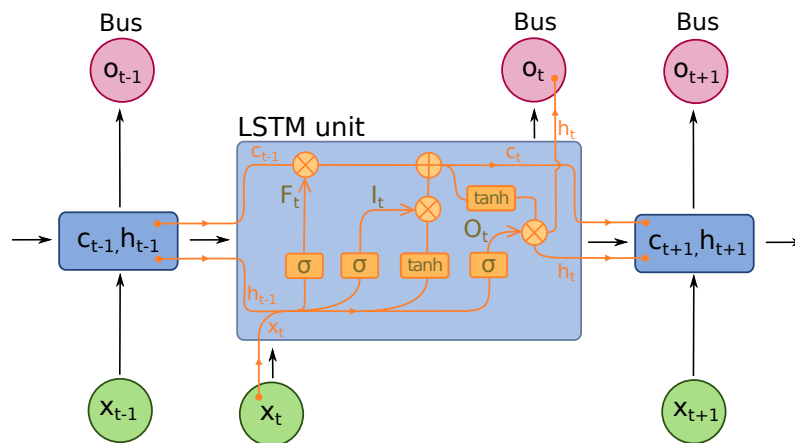


Figure 2.8: Visual representation of a LSTM based RNN for travel mode classification. Adapted from <https://commons.wikimedia.org>.

2.2.2 Gated Recurrent Unit

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks introduced in 2014. They are used in the full form and several simplified variants. Their performance on polyphonic music modelling and speech signal modelling was found to be similar to that of long short-term memory. They have fewer parameters than LSTM, as they lack an output gate [60]. Figure 2.9 presents a visual representation of this model.

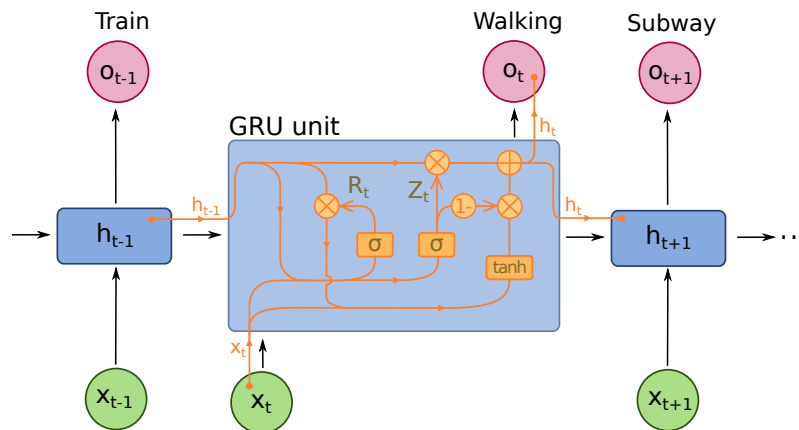


Figure 2.9: Visual representation of a GRU based RNN for travel mode classification. Adapted from <https://commons.wikimedia.org>.

2.3 Automated Machine Learning Techniques for building Classification Models

One of the main challenges in the use of ML algorithms is the configuration of their hyperparameters. The choice of these parameters can directly interfere in the accuracy of the generated model and in the training cost. Traditionally, these choices have been made based on statistical intuition, previous work or default values, as we could notice on most related works mentioned in Table 5.1. However, these choices rarely lead to the best possible performance of these algorithms [48]. This has motivated the development of a new research area, named Automated Machine Learning (AutoML), that tries to develop frameworks that can autonomously identify the optimal configuration of ML algorithms and general feature engineering techniques, such as PCA, that might enhance its performance.

2.3.1 Hyperparameter Optimization

In the last years, several techniques for optimizing hyperparameter choice in an automated way have been proposed, the main ones being Grid Search, Random Search and

Bayesian Optimization.

2.3.1.1 Grid Search

Grid Search technique consists of trying all possible configurations of an algorithm, or a representative subset of them, and choosing the one that presents the best performance according to a predefined metric, such as accuracy, precision, recall or F-score [134]. The main advantage of this technique is that it always finds the best possible configuration and, although it is computationally costly, it is easily parallelized in distributed environments [134]. The main shortcoming of this technique is that it is not adaptive. In other words, the choice of next attempts is not optimized based on previous attempts.

2.3.1.2 Random Search

The Random Search technique uses a uniform probability distribution to choose the configurations that will be tested, allowing a greater variation of hyperparameter values used in a smaller time space [134]. It is more efficient than Grid Search technique in very large search spaces and is also easily parallelized [134]. As Grid Search, the main shortcoming is that it is not adaptive.

2.3.1.3 Bayesian Optimization

Grid and Random Search limitations are the main advantage of Bayesian Optimization [124] and other sequential model-based optimization techniques [67]. Bayesian Optimization analyzes the performance metrics of past configurations to choose new configurations that present the highest expectation of improvement [134]. In this way, this technique is able to find better configurations in a much shorter time interval [124].

2.3.2 Combined Algorithm Selection and Hyperparameter Optimization

To a great extent, choosing a machine learning algorithm can be as challenging as selecting its hyperparameter values. Each algorithm has its own characteristics and might present better performance depending on the nature of the pattern that is being learned. Therefore, in [134] authors of the Auto-WEKA¹ framework formalized the *combined algorithm selection and hyperparameter optimization problem* (in short: CASH).

Given a set of algorithms $\mathcal{A} = \{A^{(1)}, \dots, A^{(k)}\}$ with associated parameter spaces $\Lambda^{(1)}, \dots, \Lambda^{(k)}$ and limited amount of training data $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$, they define CASH as in

¹<https://www.cs.ubc.ca/labs/beta/Projects/autoweka/>

Equation 2.13:

$$A^*\lambda^* \in \underset{A^{(i)} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^k \mathcal{L}(A_{\lambda^{(i)}}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}). \quad (2.13)$$

$\mathcal{L}(A_{\lambda^{(i)}}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)})$ is the loss (*i.e.*, misclassification rate) achieved by A when trained on $\mathcal{D}_{\text{train}}^{(i)}$ and evaluated on $\mathcal{D}_{\text{valid}}^{(i)}$. Constant k is the number of folds used in the k -fold cross-validation process [74], which splits the training data into k equal sized partitions $\mathcal{D}_{\text{valid}}^{(1)}, \dots, \mathcal{D}_{\text{valid}}^{(k)}$ and sets $\mathcal{D}_{\text{train}}^{(i)} = \mathcal{D} / \mathcal{D}_{\text{valid}}^{(i)}$ for $i = 1, \dots, k$.

Utilizing Bayesian Optimization, Auto-WEKA is able to solve this problem in an interactive way. In its n -th iteration, it fits a probabilistic model based on the first $n - 1$ loss function evaluations and uses this model to select the next combination of algorithm and hyperparameters to evaluate. It makes a trade-off between exploration of new parts of the search space with exploitation of regions known to be good [76].

2.3.3 Global Optimization

Given the generality of the Bayesian Optimization technique and the CASH problem, in [134] authors were able to include feature preprocessing algorithms, such as Best First, Greedy Stepwise and Ranker, in the search space as well. Provided that, the Auto-WEKA framework is able to select the best combination of machine learning classifier, hyperparameter values and feature preprocessing technique between the ones available in the WEKA Suite². This approach has been categorized as a solution to the Global Optimization³ of a ML-based system.

Therefore, in [48] authors extended this solution and implemented an open-source framework based on the well known scikit-learn⁴ python library, named AutoSklearn⁵. In this work, they extend the technique proposed in [134] to include the selection, configuration and combination of data preprocessing techniques. They also proposed two major improvements on the AutoML process: a meta-learning component for increased efficiency and an post-processing component for increased robustness. Figure 2.10 provides an overview of their framework.

The meta-learning component compares several meta-features [48] of the current

²<https://www.cs.waikato.ac.nz/ml/weka/>

³<http://aclib.net/acbib/>

⁴<http://scikit-learn.org/>

⁵<https://automl.github.io/auto-sklearn>

dataset with a collection of more than 140 benchmark datasets that were previously analyzed. It identifies the datasets that have a similar set of meta-features and selects their best performing configurations to be utilized in the initial interactions of Bayesian Optimization for the current dataset.

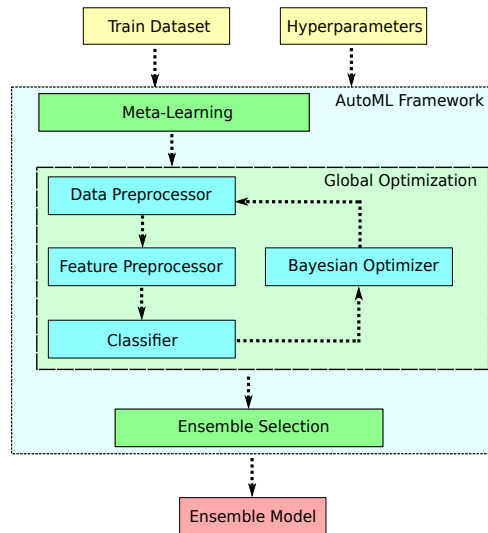


Figure 2.10: Automated Machine Learning workflow implemented in AutoSklearn. Diagram adapted from [48].

The post-processing component utilizes an ensemble selection technique to build an ensemble containing up to 50 best performing models evaluated during the Bayesian Optimization [27]. It starts from an empty ensemble and then iteratively adds the model that maximizes ensemble validation performance [48].

2.4 Performance Metrics for evaluating Classification Models

When evaluating classification models, the most common performance metrics analyzed are accuracy, precision, recall, f1-score and kappa coefficient.

2.4.1 Accuracy

The accuracy metric is obtained dividing the number of chunks correctly classified as belonging (True positives) or not-belonging (True negatives) by the total of inferences made (Total population) [44]. Thus, Accuracy is defined by Equation 2.14:

$$\text{Accuracy} = \frac{\text{True positives} + \text{True negatives}}{\text{Total population}} \quad (2.14)$$

2.4.2 Precision

The precision metric is obtained dividing the number of chunks correctly classified as belonging (True positives) by the total of inferences made for that class (True positives + False positives) [44]. Thus, Precision is defined by Equation 2.15:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \quad (2.15)$$

2.4.3 Recall

The recall metric is obtained dividing of the number of chunks correctly classified as belonging (True positives) by the number of collected chunks belonging to that class (True positives + False negatives) [44]. Recall is defined by Equation 2.16:

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \quad (2.16)$$

2.4.4 F1-Score

Finally, F1-score (F_1), also known as F-score or F-measure, is the harmonic average of each travel mode class precision and recall values [44]. It is defined by Equation 2.17:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.17)$$

2.4.5 Kappa Coefficient

Cohen's kappa, also know as Kappa Coefficient is a statistic that measures inter-annotator agreement [31]. It consists of a score that expresses the level of agreement between two annotators on a classification problem. It is defined by Equation 2.18:

$$k = (p_o - p_e) / (1 - p_e) \quad (2.18)$$

where p_o is the empirical probability of agreement on the label assigned to any sample (*i.e.*, the observed agreement ratio), and p_e is the expected agreement when both annotators assign labels randomly. p_e is estimated using a per-annotator empirical prior over the class labels [7].

3. Real-Time Travel Mode Detection with Location Sensors

3.1 Introduction

The identification of which travel modes are used by citizens in their daily commute, can be useful for many of context-aware applications, specially if this identification is obtained in real-time. Therefore, previous works have proposed solutions for this problem [13, 107, 131, 147, 149] using supervised machine learning algorithms for extracting travel mode patterns from sensors readings.

However, few studies presented solutions that can execute the detection of user travel modes in real-time, and fewer have presented the evaluation of these solutions in a realistic manner. Also, some of the previous studies state that off-the-shelf activity recognition solutions, do not perform well in the travel mode detection task, although many of them do not present a quantitative evidence of their poor performance.

Thus, the present chapter is trying to answer three research questions:

1. Can off-the-shelf activity recognition solutions perform travel mode detection satisfactorily?
2. Is detecting a person mode of transportation through their smartphone sensors, in real-time, truly possible?
3. Does the proposed technique allow real-time travel mode detection in urban centers?

The technique proposed in this chapter uses supervised machine learning algorithms to identify the travel mode of smartphone users based on location traces obtained through embedded sensors, in real-time. A prototype of the solution, that is capable of executing

this detection using only smartphone hardware (*i.e.*, in-device processing), was implemented, and its performance was evaluated through realistic field tests in an urban environment. Additionally, an evaluation of an off-the-shelf activity recognition solution is presented, namely ActivityRecognition API¹, available on Android platform, in the travel mode detection task, in order to justify the development of new techniques. The main contributions of this chapter are:

- Proposal of a real-time travel mode detection technique.
- Development of a prototype of the proposed technique as a mobile application, namely CityTracks-RT, using Android platform and Weka API².
- Evaluation of the proposed solution through realistic field tests with 37 users in the metropolitan region of Rio de Janeiro from Fall/2016 to Spring/2017.
- Development of a prototype application that allows multi-sensor data collection and real-time travel mode detection, namely CityTrack-AWARE, using AWARE framework [47] and ActivityRecognition API.
- Evaluation of ActivityRecognition API performance in the travel mode detection task, through field tests with 6 users in the metropolitan area of Rio de Janeiro during Fall/2017.

Parts of these contributions were published in [125, 126]. The remainder of this chapter is structured as follows. Section 3.2 details related work and their main limitations. Section 3.3 presents the proposed solution. Section 3.4 presents CityTracks-RT, the solution prototype and data collection tool. Section 3.5 presents the evaluation of the proposed solution prototype. Section 3.6 presents the ActivityRecognition API evaluation using the CityTracks-AWARE data collection tool. Section 3.7 presents the main conclusions.

3.2 Related Works

In this section, the state-of-the-art solutions for travel mode detection, both offline and online (*i.e.*, real-time), are described.

In [149], supervised learning through artificial neural networks was applied to identify the travel modes used in each segment of a trip. The reported accuracy for bus trips

¹<https://developers.google.com/location-context/activity-recognition/>

²<https://weka.wikispaces.com/>

detection was greater than in any other study, although this work did not consider a deficiency of the traditional neural network algorithms, called *local optimum*, and the fact that comparing to results from previous studies might be misleading since they were carried out using different datasets, which can present very contrastive quality of data [146]. Neural networks have also achieved superior performance than other machine learning techniques for the offline detection task in [147].

In [121], accelerometer and gyroscope data are used to develop an offline travel mode detection algorithm using Random Forest (RF) [21] for classification. They achieve very high accuracy on six modes classification, but do not propose an online detection technique.

In [43], authors proposed several features based on accelerometer, magnetometer and gyroscope sensors for offline travel mode detection and compare classification performance using Decision Tables (DT) [73], k-Nearest Neighbours (kNN) [4] and Support Vector Machine (SVM) [2]. The best results were achieved using SVM for classification.

In [8], authors also used accelerometer and gyroscope data as well as GPS information to develop an offline travel mode detection algorithm using Multinomial Logistic Regression (MNL) [18], Nested Logit (NL) [141] and Multiple Discriminant Analysis (MDA) [37] for classification. The best performing classifier was the one built using NL with eight variables.

In [107] the CityTracks application was proposed for collecting smartphone users mobility data through mobile crowdsensing. The collected data contained the travel modes used and the user location at roughly each second, which was used to perform Knowledge Data Discovery [45] and propose an offline detection technique that performed location data preprocessing and segmentation, feature extraction and summarization, as well as travel mode classification with supervised machine learning algorithms.

In a similar way, [13] collected labelled samples of GPS, accelerometer and gyroscope sensors data to create a real-time travel mode detection technique based on classifier cascading [3]. They used the magnitude metric [12, 24, 135] to extract orientation independent features from accelerometer and gyroscope readings. One of the limitations of this work is the fact that proposed technique was evaluated in an offline manner, although it performed online classification. Therefore, the simulated real-time detection results can present optimistic metrics which would be hardly achieved in a real use case.

In [131], data from accelerometer, gravity sensor, gyroscope, magnetometer and barometer is collected to develop an energy efficient technique for online detection. By not us-

ing the GPS as a data source the authors were able to achieve a great reduction in energy consumption and a good classification performance during the simulation of online detection. Although this work achieved great results using a large variety of sensors, some of the sensors used are not present in most smartphones models. This could restrict the use of this technique for a small percentage of the citizens, what could limit the impact of the ITS applications that rely on it.

In [28], the authors presented a technique that combines smartphone Hall-Effect magnetometer and accelerometer data to detect the users mode of transportation in real time, by applying advanced signal processing techniques and extracting features from each spectrum of the Fast-Fourier Transform (FFT) [109] of each second of movement, such as entropy, energy, energy ratio and cepstral coefficients. Despite the promising results, the authors did not implement or evaluate a prototype of the proposed technique on a real smartphone platform.

In [90] the authors proposed an online travel mode detection technique that combined smartphone GPS and accelerometer data. They applied the “movelets” technique [9] for preprocessing, Principal Component Analysis [143] and Recursive Feature Elimination (RFE) [88] for dimensionality reduction, k -Nearest Neighbours (k NN) and Random Forest (RF) for classification, and 10-fold cross validation [74] for performance evaluation. The best performing classifier was the one built with RF.

In [10] and [139], the authors augmented GPS data with socioeconomic information from the users to improve offline travel mode detection. In [10] dynamic Bayesian Networks are built for classification and their performance is evaluated on five test datasets and compared with SVM, RF and Multilayer Perceptron (MLP) [111]. In [139], RFs are used to build the main classification model and their performance is compared with MLP and SVM models using 60% of the study data for training and 40% for testing.

In [89] authors performed hypothesis testing of several feature extraction techniques and comparison of different ML approaches including Autoencoders for travel mode detection using GPS traces. Following the deep learning trend, in [34] the authors propose its use for offline travel mode detection based on GPS trajectories, removing the need of feature engineering techniques. They compare multiple Convolutional Neural Network (CNN) [81] architectures and present performance metrics for each configuration using a test dataset that contained 20% of the study data.

Travel mode detection via smartphones can use a wide variety of sensors, however, most state-of-the-art solutions found use only GPS data [10, 34, 89, 139, 147, 149]. Some

solutions used accelerometer readings and cellular data to compensate for GPS signal losses [90, 107] and others used data from the gyroscope, magnetometer, and other less conventional sensors such as barometer and gravity sensor to reduce power consumption [8, 13, 28, 43, 121, 131].

In the solution proposed in this chapter, a combination of GPS, WiFi and cellular networks data was used, since these sensors are available in most of today’s smartphones and allow location tracing with good precision in urban centers.

As illustrated in Table 3.1, none of the related work presented the evaluation of an application for real-time travel mode detection in a real use scenario, what states the originality of this work.

Table 3.1: Summary of related works on travel mode detection through smartphone sensors. *Acc - Acelerometer, Gyr - Gyroscope, Mag - Magnetometer, Cellular - Cellular Networks

Ref.	Classification Algorithm	Detection Type	Sensors Used
[149]	Neural Networks	Offline	GPS
[147]	Bayesian Networks	Offline	GPS
[79]	Random Forest	Offline	GPS
[121]	Random Forest	Offline	Acc, Gyr
[43]	Support Vector Machine	Offline	Acc, Mag, Gyr
[8]	Nested Logit	Offline	GPS, Acc, Gyr
[131]	Hierarchical Classifier	Online	Acc, Gravity Sensor, Gyr, Mag, Barometer
[13]	Cascading of Classifiers	Online	GPS, Acc, Gyr
[107]	Hierarchical Classifier	Offline	GPS, WiFi, Cellular
[28]	Hierarchical Classifier	Online	Hall-Effect Mag, Acc
[90]	Random Forest	Online	GPS, Acc
[10]	Bayesian Networks	Offline	GPS
[139]	Random Forest	Offline	GPS
[89]	Random Forest	Offline	GPS
[34]	Convolutional Neural Networks	Offline	GPS

3.3 Proposed Solution

The solution proposed in this chapter is able to continuously extract features from chunks of location traces at every 90 seconds and infer which travel mode is being used, without the need of expensive segmentation and noise removal techniques.

Instead, it uses lightweight preprocessing techniques, that can be applied while the data is being collected. This approach is also referred to as edge mining, or edge computing, and allows our technique to be executed within smartphone devices, without the need of demanding computing resources from the cloud. Figure 3.1 illustrates the proposed solution. It was based on the data mining technique presented in [107] for offline

detection.

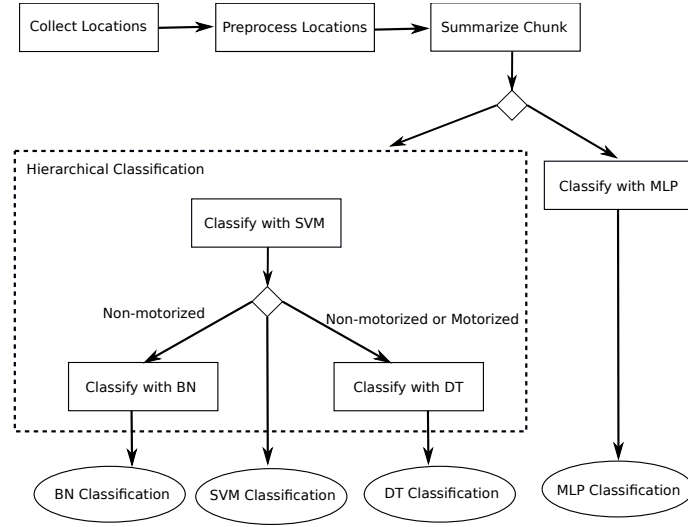


Figure 3.1: Proposed Real-Time Travel Mode Detection Solution. *MLP - Multilayer Perceptron, SVM - Support Vector Machine, BN - Bayesian Net, DT - Decision Table

At maximum frequency of one sample per second, location traces containing altitude, latitude, longitude, precision, and timestamps of the measurement are captured. After each sample collection, a preprocessing routine calculates the distance, based on the Haversine formula [123], instantaneous speed, and acceleration. Samples with speed lower than 0.4 m/s are considered to be equivalent to stops and samples with location precision worse than 200 meters are considered to be invalid, and therefore are discarded.

If the timestamp difference in seconds represented by td_s between two subsequent valid samples, s_{valid}^1 and s_{valid}^2 , is higher than one second, a set of synthetic samples represented by $S_{synt} = \{s_{synt}^1, s_{synt}^2, \dots, s_{synt}^n\}$ is inserted in between them by an interpolation technique. This technique generates n synthetic samples with the same altitude, latitude, longitude, precision, distance and speed of s_{valid}^1 but with acceleration equals zero, where $n = td_s - 1$.

After each one and a half minutes of movement data collection, the samples are grouped into a *chunk* and a summarization routine extracts the maximum speed, maximum acceleration, and number of direction changes in these 90 seconds. These features were selected base on the results presented in by a previous work, that indicated that they could be useful for online detection [108].

When the chunk summarization is concluded, travel mode classification is performed using two main approaches. In one of them, a hierarchical classification scheme is used, which first classifies the chunk into motorized and non-motorized using a Support Vec-

tor Machine (SVM) pre-trained with Sequential Minimal Optimization (SMO), then uses a Bayesian Network or a Decision Table classifier to identify non-motorized chunks as walking or biking, as well as a Decision Table classifier for detecting bus, car or motorcycle chunks from the chunks classified as motorized by the SVM.

This approach was based on the results of our previous work [107] which showed that this hierarchical classification led to better performance than other approaches in the offline detection problem. Other related works [28, 90, 131] have shown that a hierarchical approach could lead to good classification results in online detection also.

The second approach is to use a single Multilayer Perceptron (MLP) classifier, which is responsible for classifying the chunks into walking, biking, bus, car or motorcycle. Neural network based algorithms, such as the Multilayer Perceptron have also presented good results in previous works of offline detection [107, 147, 149].

A preliminary version of the proposed solution was presented in [126].

3.4 Prototype Development

In this section, the architecture and implementation of the solution prototype and data collection tool, named CityTracks-RT, are presented. It was used in the evaluation of the real-time travel mode detection technique proposed in this chapter.

This prototype was implemented on the Android platform and used the FusedLocationAPI³ to collect the locations samples, the Weka API to run the supervised machine learning algorithms and the Gson API⁴ to store collected *chunks* in JavaScript Object Notation (JSON) format.

The proposed solution was implemented using the Object Oriented (OO) programming paradigm [110] with a Model-View-Controller (MVC) design pattern [77].

Therefore, the locations and chunks were defined as Model classes, and the Factory Method and Builder patterns [54] were used to instantiate them while applying the pre-processing and summarization steps detailed in Section 3.3.

A generic JSON Data Access Object (DAO) class [101] has also been implemented. It allowed the in-device persistence of the Model instances using the Gson API. This set of classes can be viewed as the Model layer of the application.

³<https://developers.google.com/>

⁴<https://github.com/google/gson>

The View layer or User Interface (UI), allowed the user to select the current travel mode and to view the timestamps, latitude and longitude of the last captured location. It was implemented using an Android Activity (*CityTracksRTActivity*), that defined the UI logic, and an XML file, that defined the UI layout.

The Control layer was implemented using an Android Service (*CityTracksRTService*). Once this Service is triggered by the *CityTracksRTActivity*, it keeps running in background, even if the app is minimized by the user. This guarantees that the data collection and travel mode detection are not interrupted, unless the user kills the background process manually.

The *CityTracksRTService* is responsible for preprocessing the locations, that are collected through the FusedLocationAPI, by using the location Factory Method, and summarizing the chunk after each one and a half second, by using the chunk Builder. It then identifies the chunk travel mode using the a wrapper class for the Weka API and stores the chunk instances, containing the preprocessed locations, extracted attributes, user informed and inferred labels, using the JSON DAO.

The wrapper class for the Weka API used default hyperparameter values for most of the machine learning algorithms used, with the exception of Multilayer Perceptron algorithm, whose learning rate was set to 0.1, momentum equals 0.2, training time of 2 seconds and three hidden layers between the input and output.

Figure 3.2 contains high-level representation of the interaction between the app components and Figure 3.3 presents the workflow of the CityTracks-RT application indicating which component is responsible for each task in the workflow. The prototype source code and application package (APK) are available at <https://bitbucket.org/eltonfss/citytracks-rt/>.

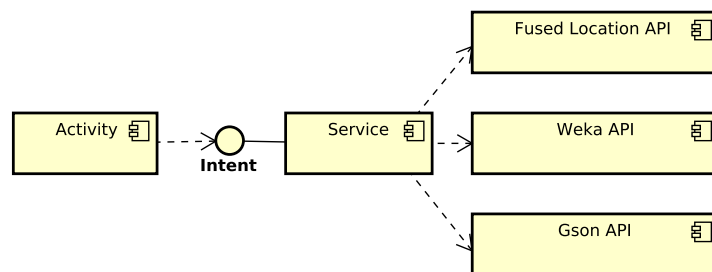


Figure 3.2: CityTracks-RT application components.

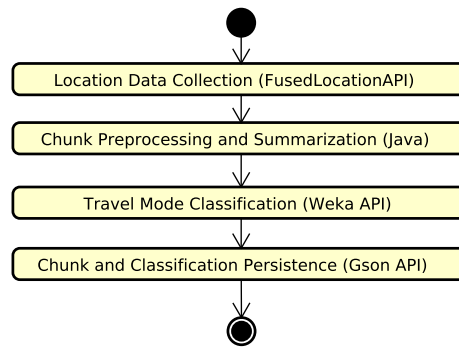


Figure 3.3: CityTracks-RT application workflow.

3.5 Prototype Evaluation

In this section, the performance evaluation of the real-time travel mode detection technique proposed in Section 3.3 is presented. This evaluation was conducted through field tests, in the metropolitan region of Rio de Janeiro, where several volunteers were invited to use our prototype application, CityTracksRT, in their daily commute and to inform their travel modes manually as the ground truth label for our quantitative analysis.

3.5.1 Methods

In total, 37 volunteers have agreed to participate and install the CityTracksRT app. Users received an illustrated manual that instructed them to start the data collection whenever they were commuting through one of the travel modes available in the app and selecting the travel mode option before the movement started. Whenever they switched modes, they should select another travel mode option, if it was available, or stop data collection, otherwise. The data collected was shared via email on the first collection phase and through a web service that was called directly from the CityTracks-RT mobile app on a second phase.

Data collection took place mainly in the South Zone and Downtown of Rio de Janeiro city, since most of the volunteers were students from the Federal University of the State of Rio de Janeiro, which is located in the Urca neighbourhood, within the South Zone and not far from Downtown. Some traces were also collected in satellite cities within Rio de Janeiro (RJ) metropolitan area, such as Duque de Caxias, Nilópolis, Nova Iguaçu and Niterói, and in the route between these cities and Downtown. The spread of the location traces obtained over these geographical regions can be observed in Figure 3.4.

All chunks collected by the users were serialized and stored in JSON format. There-

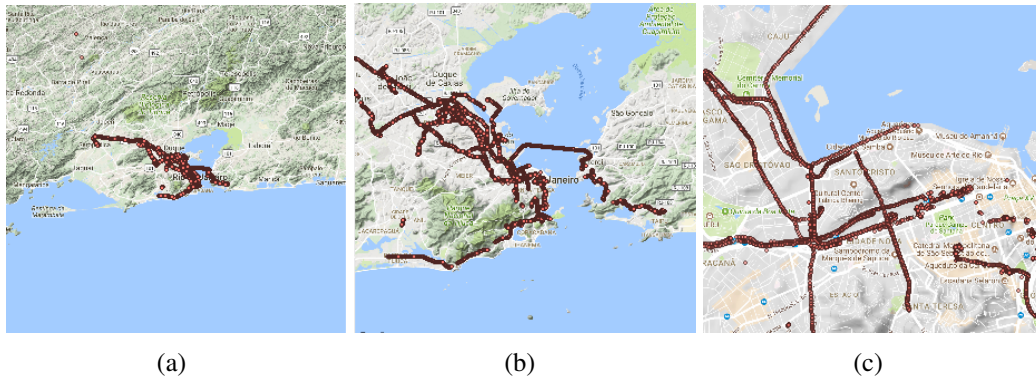


Figure 3.4: Spatial visualization of users' displacement on the Rio de Janeiro metropolitan area during the use of CityTracks-RT application.

fore, a script was developed using the Java programming language and Gson API, in order to restore the chunks information, including all locations coordinates, chunk attributes, inferences made by all classifiers and the ground truth label provided by the user. Thus, this script was used to load all chunks objects in memory as a Java Collection⁵.

A second script, received these chunks as input and calculated the number of True Positives, chunks correctly classified as belonging to a travel mode class, True Negatives, chunks correctly classified as not-belonging to a travel mode class, False Positives, chunks incorrectly classified as belonging to a travel mode class, and False Negatives, chunks incorrectly classified as not-belonging to a travel mode class, for each classifier. These were used to build the confusion matrices that are presented in Section 3.5.3.

Another script calculated the accuracy, precision, recall and F1-score metrics [44] which were used to evaluate and compare supervised machine learning algorithms performance in travel mode classification tasks in previous works [13, 107, 131, 147, 149]. The accuracy can be viewed as a naive metric that summarizes the overall performance while precision and recall metrics allow us to analyze the overall ability of a classifier to correctly classify samples of all classes and less frequent classes, respectively. Finally, the F1-score is the harmonic average of the precision and recall, and is one of the most reliable indicators of good classification performance [105].

3.5.2 Data

In this section the collected data is characterized, by analyzing overall statistical measures and measures by user. The summary statistics by user are presented in Table 3.2.

⁵<https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html>

The location traces obtained had a mean accuracy of 22.1 meters and a total of 2,519 chunks and 226,466 locations were obtained, about 48.7% directly by the sensors via FusedLocationAPI and 51.3% by the interpolation technique described in Section 3.3. The total period of data collection collected sums up to 63 hours, with an average of approximately 1.7 hours of data collected by each user.

The large disparity observed in the number of chunks collected by each volunteer can be associated to the absence of an effective incentive mechanism in the data collection approach. In the approach adopted, users did not receive any concrete or abstract reward for the amount of data they collected.

With respect to the travel mode distribution, more than 80% of the locations were collected by bus or walking, while less than 20% were collected by car and walking, what can be seen in detail on the histogram of Figure 3.5.

Therefore, only users who were truly interested in contributing to this research collected a meaningful amount of data, while others did not become as engaged as them. This suggests that data the final dataset might be biased towards a part of the population, that is more concerned with the outcomes of research activity in general.

The precision threshold of 200 meters was established based on [107] and all locations collected were compliant with it. While analyzing the data collected, it is possible to conclude that reducing this threshold below 200 meters would significantly reduce the number of samples obtained, causing the locations traces to be too sparse and making travel mode detection harder and less efficient.

As presented on Figure 3.5 most of the locations precision fell between 10 and 40 meters, almost 70%, with a small amount of locations being collected with precision below 10 meters and above 40 meters. The importance of this analysis must be emphasized because the precision of locations collected will directly influence the context-aware applications that might depend on the travel mode detection.

3.5.3 Results

In this section the results obtained by each classifier are described, with respect to the metrics defined in Equations 2.14, 2.15, 2.16 and 2.17.

In the hierarchical classification approach, SVM achieved mean per class accuracy of approximately 62.2%, 60.3% precision and 60.2% recall. Therefore, the mean F1-score was about 60.3%. The precision for motorized chunk classification was much higher than

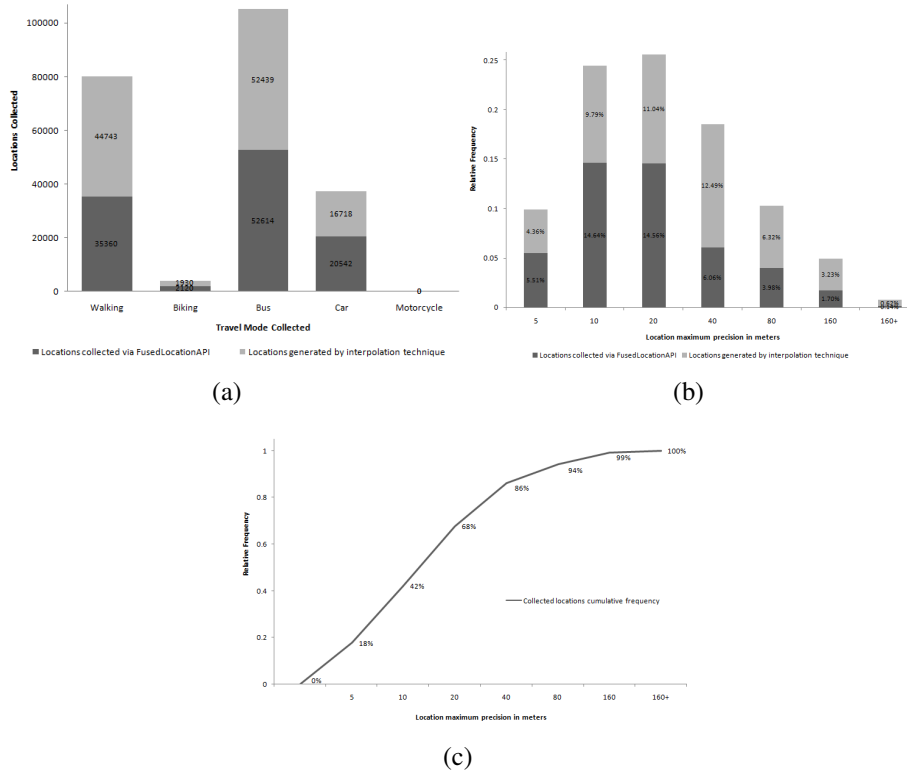


Figure 3.5: Locations captured x locations generated per travel mode (a). Locations captured x locations generated by precision range in meters (b). Cumulative frequency of locations collected by precision range (c).

for non-motorized, while recall achieved similar values for both classes.

This indicates that a high percentage of chunks classified as non-motorized were misclassified motorized chunks. In Table 3.3 the confusion matrix from which those metrics were obtained is presented.

From the 2,519 samples, 1,051 were labelled as motorized and correctly classified by the SVM algorithm. Decision Table classifier achieved mean per class accuracy of 68.0%, 59.2% precision and 58.0% recall. In the Table 3.4 the confusion matrix for this classifier is presented.

Meanwhile, SVM algorithm was able to classify 506 chunks labelled as non-motorized correctly. Therefore, these were classified as walking or biking by a Bayesian Net and a Decision Table classifiers. Approximately 67.0% mean per class accuracy was achieved with Bayesian Net and 66.2% with Decision Table. Precision, recall and F1-score were, respectively, 45.0%, 36.2% and 40.1% for Bayesian Net and 45.0%, 35.8% and 39.8% for Decision Table. In Table 3.5 the confusion matrices of both classifiers are presented.

In comparison to the hierarchical approach, the Multilayer Perceptron classification

Table 3.2: Data collection summary statistics per user. *#H - Hours of data collection, #C - Number of chunks collected, #L - Number of locations collected, MP - Mean location precision, SD - Precision standard deviation, %C - Percentage of locations captured via FusedLocationAPI, %G - Percentage of locations generated by interpolation.

ID	#H	#C	#L	MP	SD	%G	%C
1	8.4	337	30,330	12.4 m	13.9 m	34.6	65.4
2	6.3	253	22,770	34.1 m	24.5 m	76.4	23.6
3	5.5	222	19,980	13.1 m	7.1 m	38.1	61.9
4	5.5	219	19,710	32.3 m	25.0 m	59.9	40.1
5	4.4	176	15,840	14.3 m	23.5 m	87.6	12.4
6	3.1	123	11,070	20.4 m	23.5 m	36.4	63.6
7	2.7	110	9,900	39.0 m	42.5 m	35.7	64.3
8	2.4	95	8,550	5.0 m	6.6 m	31.0	69.0
9	2.2	87	7,830	25.7 m	40.0 m	33.2	66.8
10	2.1	83	7,470	42.7 m	38.7 m	83.8	16.2
11	1.7	69	6,210	15.4 m	22.7 m	37.7	62.3
12	1.7	68	6,120	9.4 m	6.0 m	36.0	64.0
13	1.5	61	5,490	21.9 m	15.7 m	72.0	28.0
14	1.3	52	4,680	26.5 m	15.9 m	77.5	22.5
15	1.3	51	4,590	47.6 m	43.1 m	81.6	18.4
16	1.2	50	4,500	3.5 m	1.6 m	31.4	68.6
17	1.1	44	3,960	14.6 m	22.7 m	38.8	61.2
18	1.0	40	3,345	40.1 m	31.1 m	64.6	35.4
19	1.0	39	3,510	19.7 m	15.3 m	41.7	58.3
20	0.9	36	3,240	9.6 m	16.4 m	36.7	63.3
21	0.9	35	3,150	47.4 m	49.7 m	68.6	31.4
22	0.8	34	3,060	34.4 m	43.7 m	38.0	62.0
23	0.7	30	2,700	20.1 m	35.6 m	41.4	58.6
24	0.7	27	2,430	28.2 m	29.7 m	83.5	16.5
25	0.7	27	2,430	13.6 m	8.6 m	42.0	58.0
26	0.6	23	2,070	11.8 m	25.3 m	37.4	62.6
27	0.5	22	1,991	7.4 m	7.7 m	2.2	97.8
28	0.5	20	1,800	14.4 m	28.8 m	40.1	59.9
29	0.4	16	1,440	48.3 m	47.4 m	67.2	32.8
30	0.3	14	1,260	9.2 m	10.8 m	34.6	65.4
31	0.3	13	1,1170	10.1 m	7.0 m	32.5	67.5
32	0.3	11	990	4.1 m	4.4 m	36.4	63.6
33	0.2	9	810	5.7 m	6.1 m	35.2	64.8
34	0.2	8	720	30.4 m	11.2 m	41.5	58.5
35	0.2	7	630	7.0 m	3.7 m	34.8	65.2
36	0.2	7	630	30.9 m	4.2 m	39.4	60.6
37	0.0	1	90	54.1 m	37.1 m	47.8	52.2
All	63	2519	6121	22.3 m	21.5 m	51.3	48.7

Table 3.3: Confusion matrix of the motorized and non-motorized chunk classification for the SVM classifier.

Classified \ Labelled	Motorized	Non-motorized
Motorized	1051	427
Non-motorized	517	506

Table 3.4: Confusion matrix of motorized mode classification for the Decision Table classifier.

Classified \ Labelled	Bus	Car
Bus	632	109
Car	167	83

Table 3.5: Confusion matrices of non-motorized chunk classification for Decision Table and Bayesian Net classifiers.

Classifier Classified\Labelled	Decision Table		Bayesian Net	
	Walking	Biking	Walking	Biking
Walking	335	38	339	38
Biking	133	0	129	0

achieved 45.6% mean per class accuracy, 37.8% precision, 31.8% recall and 31.0% F1-score. Due to the absence of chunks collected by motorcycle, this mode was not considered in the calculation of these metrics. Table 3.6 presents the confusion matrix for this classifier.

Table 3.6: Confusion matrix of walking, biking, bus and car classification with Multilayer Perceptron.

Classified\Labelled	Walking	Biking	Bus	Car
Walking	463	35	231	168
Biking	96	1	189	37
Bus	269	7	594	107
Car	50	1	96	92

As the number of chunks collected for each mode of transportation was not equally high for all travel mode classes, results obtained should be analyzed with caution, given that class imbalance might have disproportionately increased or decreased the performance metrics obtained. Table 3.7 summarizes the mean per class accuracy, precision, recall and F1-score of each classifier.

Table 3.7: Summary of mean per class accuracy, precision, recall and F1-score of each classifier.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Multilayer Perceptron	45.6%	37.8%	31.8%	31.0%
Support Vector Machine	62.2%	60.3%	60.2%	60.3%
Decision Table	67.1%	52.0%	46.9%	49.3%
Bayesian Net	67.0%	45.0%	36.2%	40.1%

3.5.4 Discussion

Through the field tests conducted, it is possible to verify the performance of the proposed travel mode detection technique in an urban scenario. A total of 2,519 sample chunks were classified using a prototype application, in which the proposed technique was implemented, that executed the detection in real-time, using only smartphone hardware resources.

The large variation in the mean location precision of the chunks collected can be explained by the heterogeneity of the smartphone models used by the 37 volunteers. The mean location precision per user did not exceed 50 meters, what can be considered to be good, but not great. Provided that the predefined precision threshold was set to four times this value, it not possible to affirm that it had a negative impact on the detection performance, but it indicates that better results can be achieved with smartphones whose location precision is better than this.

With respect to the in-device execution of the machine learning algorithms, the classification delay was significantly small and CPU consumption was not too high. The continuous location sampling, however, has proven to be inefficient, causing the CityTracks-RT app to consume a large amount of RAM memory and battery power over time. This indicates that a better implementation would have to utilize an adaptive sampling technique, instead of the fix interval of one second, which was used on the prototype.

3.6 ActivityRecognition API Evaluation

In this section, an evaluation of the ActivityRecognitionAPI, through field tests conducted by another group of volunteers that also travelled around the metropolitan region of Rio de Janeiro, is presented. The results will be analyzed in two stages. In the first one, the data collected by the users will be analyzed, identifying the main travel modes used, number of trips and hours collected by each device and the amount of inferences made for each travel mode. In the second step, the performance of the classification algorithms will be analyzed using the same metrics used on the CityTracks-RT prototype evaluation.

3.6.1 Methods

The data collection tool developed was called CityTracks-AWARE and, in addition to collecting the travel mode informed by the user and the inferences made by ActivityRecognitionAPI, allowed the collection of data of 16 physical and virtual sensors as well as the beginning, end and purpose of the displacements. Table 3.8 lists the sensors collected and the minimum sampling frequencies applied to each of them, noting that it is not possible to guarantee a maximum sampling rate on the Android platform for all sensors.

The application was developed according to the object-oriented software design model named Model-View-Controller [54], where the source code of the application is divided

Table 3.8: Sensors supported by the CityTracks-AWARE application and their minimum sampling frequencies. * Sampling frequency does not apply because these sensors capture the data reactively.

Sensor	Min. Sampling Freq.
Accelerometer	200,000 μ s
Barometer	200,000 μ s
Battery	200,000 μ s
Gravitometer	200,000 μ s
Gyroscope	200,000 μ s
Luximeter	200,000 μ s
Linear Accelerometer	200,000 μ s
Location Sensor	180 s
Magnetometer	200.000 μ s
Network Traffic Sensor	60 s
Network Event Sensor	*
CPU	10 s
Proximity sensor	200,000 μ s
Rotation sensor	200,000 μ s
Screen Sensor	*
Thermometer	200,000 μ s
Activity Recognition	60 s

into 3 main layers. In the model layer are the classes that define the entities of the application and the mechanisms of their persistence. In the view layer are the user interface layout definitions and in the control layer the classes are responsible for processing the user-made requests, using the classes of the model layer.

In addition to using this design pattern, the application logic was split between two modules, based on the development methodology used in the AWARE app itself⁶: *citytracksaware-client* and *citytracksaware-core*. The *client* module contains all the code of the view layer while the *core* module contains all the control and model layer code.

In addition, the core module was subdivided into two more layers. In one of them are contained the models and controls of the sensors accessed through AWARE. In the other are contained the models and controls regarding the trip itself. The purpose of this division was to separate the code that takes care of the integration with the AWARE API and the code that manages the data informed by the user to facilitate the interpretation and maintenance of the code. The application architecture is illustrated in component diagram of Figure 3.6.

In addition, two auxiliary classes, that are used in all layers, have been developed. One allows persistence of any instance of a class in JSON format files through the Gson⁷ API and the other allows the asynchronous sending and re-transmission of HTTP requests to a web server through the Volley API⁸. The first was used to store the travel data in the device during the collection and the second was used to transmit the data collected to the

⁶<https://github.com/denzilferreira/aware-client>

⁷<https://github.com/google/gson>

⁸<https://github.com/google/volley>

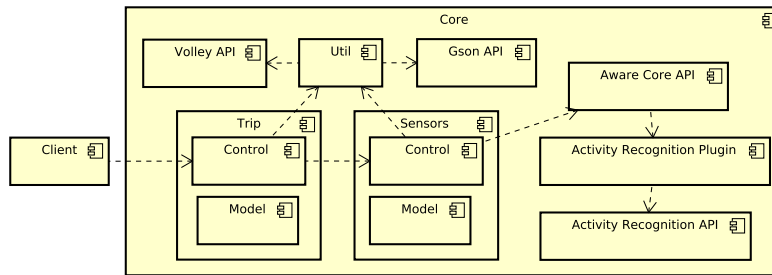


Figure 3.6: Component diagram of the CityTracks-AWARE application.

collection server.

The collection server received the data sent in JSON format by HTTP POST messages and persisted in a MongoDB database⁹ through a web service developed through the PHP-Slim¹⁰ framework using the PHP MongoDB Driver¹¹ to access the database. The application server used was the NGINX¹² and, to allow greater scalability of the application, PHP-FPM¹³ was used as FastCGI process manager. Figure 3.7 illustrates the architecture of collection server with its respective components. The entire environment was configured on a virtual machine running Ubuntu Server¹⁴.

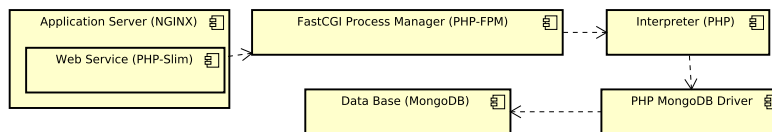


Figure 3.7: Diagram of collection server components.

Although the architecture of the application followed good development practices, during the Alpha tests some performance problems were noticed, especially when it came to sending the data to the server. This was mainly due to the use of the layered architecture, because to ensure the isolation between the layers, the app had to read all the data collected by each sensor in the model layer, and then send them to the server in the control layer. This caused an overload in the RAM of lower-end smartphones, especially in sensors with sampling frequencies less than one second, such as the accelerometer.

To optimize this process, in order to respect the limitations of the mobile devices in which the application would be used, the responsibility of reading the data collected by the sensors was moved to the control layer. This way, each sample collected was read and sent

⁹<https://www.mongodb.com/>

¹⁰<https://www.slimframework.com/>

¹¹<https://docs.mongodb.com/ecosystem/drivers/php/>

¹²<https://nginx.org/>

¹³<https://php-fpm.org/>

¹⁴<https://www.ubuntu.com/server>

immediately, freeing up memory space faster and reducing the total RAM consumption of the device. This change in processing of data is illustrated in Figures 3.8 and 3.9.

In Figure 3.8 the initial approach, in which the *SensorManager* requests all data collected by a sensor to a *SensorDataDAO*, is detailed. The DAO retrieves a collection of *SensorDataModel* instances whose size is the number of samples collected by that sensor. After that, the *SensorManager* iterates through the retrieved collection and requests the upload of each instance through the *UploadService*.

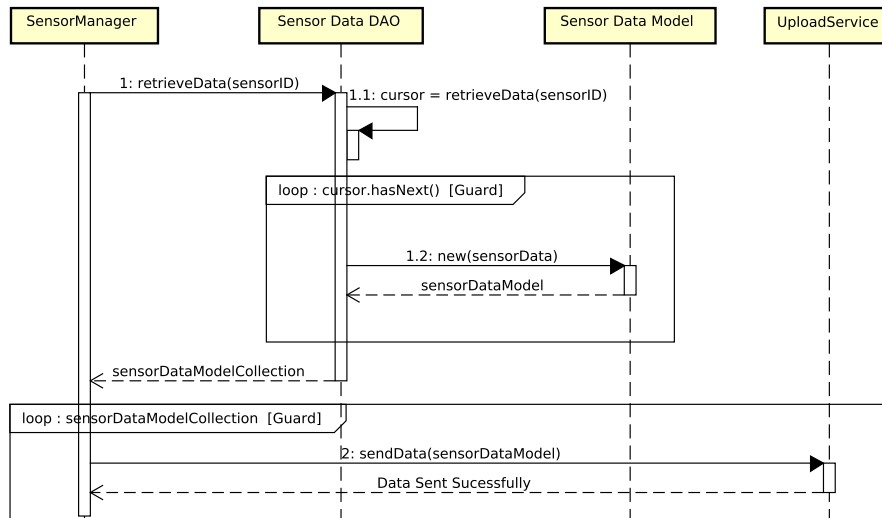


Figure 3.8: Sequence diagram of the processing of data collected by the sensors in the CityTracks-AWARE app when sending to the server (Before).

In Figure 3.9 the optimized approach is detailed, in which the *SensorDataDAO* is replaced by a *DataManager*, that is responsible for creating the *SensorDataModel* instances from sensor samples and sending the upload requests through the *SensorDataUploadService*. Here the *DataManager* accumulates the responsibilities of the *SensorDataDAO* and the *SensorManager* in order to allow a better memory utilization. Instead of reading all sensor samples at once, only one instance is read and uploaded at a time, which reduces the overall memory consumption of the application.

To analyze the data collected and calculate the performance metrics, a program was developed in Python¹⁵ using the Pandas API¹⁶ for reading and manipulating the JSON database exported from the collection server through the MongoDB data export tool¹⁷.

In order to reduce the effect of possible collection errors occurring during the transition between travel modes, the developed program ignores the inferences made within 30 seconds before and after the travel mode changes reported by the user.

¹⁵<https://www.python.org/>

¹⁶<https://pandas.pydata.org/>

¹⁷<https://docs.mongodb.com/manual/reference/program/mongoexport/>

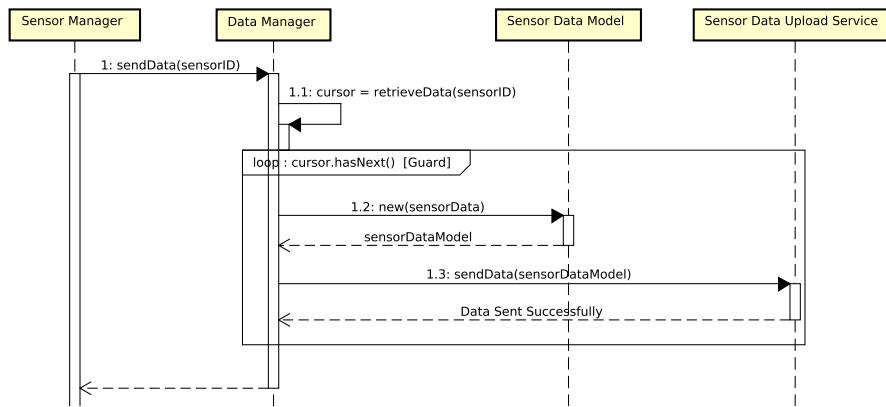


Figure 3.9: Sequence diagram of the processing of data collected by the sensors in the CityTracks-AWARE app when sending to the server (After).

The developed program calculates the confusion matrix based on the collected data and, from it, extracts the metrics used for performance evaluation.

3.6.2 Data

A total of 6 users participated in the data collection, however, due to unknown limitations of the Activity Recognition API implementation, only 3 devices had inferences from it. Therefore, the other devices were discarded from this analysis.

A total of 24 trips were collected, generating more than 48 hours of trip data. Table 3.9 presents the number of trips, hours, inferences and the modalities used by each device. More than 50% of the trips and inferences considered were collected by the same device, what might suggest a significant level of bias in the results.

Table 3.9: Data collected by device.

Device	# Trips	# Hours	# Inferences	Modes
1	4	6,92694	418	Train, Bus, Subway, On Foot, Still
2	4	22,4775	260	Bus, Subway, On Foot Light Rail, Car
3	16	18,9978	1301	Train, Bus, Subway, On Foot, Still

In Figure 3.10 the amount of inferences made by each travel mode collected is presented. It is possible to observe that a great amount of inferences was carried out on foot and a significant amount was made on bus, train, subway and still. This unbalancing of classes can also be considered to be a meaningful factor in the metrics evaluation, since the performance of classes with lower number of samples can be too much above or below true performance.

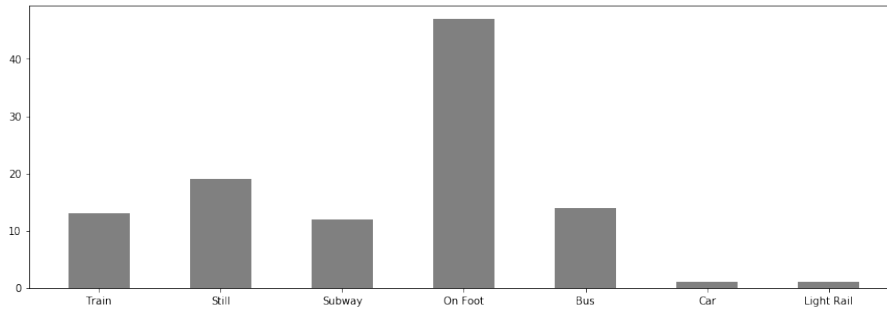


Figure 3.10: Inferences made by travel mode collected.

3.6.3 Results

In order to obtain the performance metrics, a Python script was developed to group all the data collected with motorized travel modes in the “Vehicle” class, since the Activity Recognition API does not make distinctions between vehicles. An error (False Negative and False Positive) or a hit (True Positive and True Negative) was computed for each inference made by the API, and from this information we constructed a confusion matrix indicating the errors and hits for each class, which is presented on Table 3.10.

Table 3.10: Confusion matrix grouping the data collected within the five activity classes recognized by the ActivityRecognition API.

Inferred \ Collected	Other	Tilting	Still	On Foot	Biking	Vehicle
Other	0	0	28	26	0	158
Tilting	0	0	17	123	0	103
Still	0	0	86	127	0	250
On Foot	0	0	8	471	0	83
Biking	0	0	0	8	0	33
Vehicle	0	0	6	10	0	442

The total of True Positives, False Positives, Accuracy, Precision, Recall, and F1-score for each class, as well as the overall average, are presented in Table 3.11. These were calculated using the same formulas that were used in the CityTracks-RT performance evaluation, described in Section 3.5.

Table 3.11: Performance metrics for each activity class recognized by the ActivityRecognition API. *TP - True Positives, FP - False Positives.

Class	TP	FP	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Other	0	212	89.3	0.0	0.0	0.0
Tilting	0	243	87.7	0.0	0.0	0.0
Still	86	377	80.0	18.6	59.3	28.3
On Foot	471	91	80.5	83.8	61.6	71.0
Biking	0.0	41	97.9	0.0	0.0	0.0
Vehicle	442	16	67.5	96.5	41.3	57.9
Average	166.5	163	83.5	33.1	27.0	26.2

3.6.4 Discussion

From the results obtained, it can be observed that the Activity Recognition API recognizes vehicles with high precision but low recall. Therefore, in context-aware applications that are not concerned with the quantity, but rather with the quality of the inferences for this travel mode, it may be useful. In applications where it is important to detect the maximum occurrences of this mode class, it may not perform satisfactorily.

With regards to motion segments “On Foot”, it has reasonable precision and recall, reaching a F1-score of 71%. If all classes inferred are considered, the Activity Recognition API did not perform satisfactorily, justifying the development of new APIs and comparative studies with more users. In addition, the limited set of supported travel mode classes reduces the possibility of using this API in applications directed to intelligent transport systems, since for this type of application, the distinction between motorized travel modes is very important.

3.7 Conclusion

In the present chapter the problem of real-time travel mode detection using GPS location traces has been addressed. This is an important problem for the development of intelligent transportation systems and other smart mobility applications, that may provide faster and personalized services and message delivery, based on this contextual information. Therefore, it has been identified that the only travel mode detection API that is publicly available, on the most popular mobile platform, does not fully meet the requirements of context-aware applications, by design. Also, through field tests, it has been showed that its performance is not optimal for all the scenarios it supports, justifying the need of improvement or the development of a new API for that.

Thus, a technique that allows detecting the travel mode of smartphone users, in real time, is proposed and evaluated. This technique is implemented on a prototype Android application, named CityTracks-RT, that used supervised machine learning models, built with the Weka API in Java. The performance of the prototype was evaluated through the analysis of quantitative metrics obtained via field tests with 37 volunteers in Rio de Janeiro and the results indicate that the proposed technique is capable of detecting the travel modes of smartphone users in urban centers.

4. Real-Time Travel Mode and Trip Purpose Prediction with Location Sensors

4.1 Introduction

Among the several features that may be used to describe mobility patterns, the travel mode and the trip purpose specifically contribute to their better understanding, thus benefiting the generation of automatic (or semi-automatic) travel diaries [100], recommendation systems, personal assistants and other mobile data collection applications [117, 118]. Therefore, the detection of the travel modes used and the prediction of trip purposes through smartphone sensors data have emerged as two research challenges in recent years. Both of these problems have been deeply investigated in isolation, while the problem of inferring mode and purpose at the same time and, more specifically, using the same preprocessing algorithm has been less explored.

Since travel mode detection and trip purpose prediction might need to happen at the same time, the use of a single preprocessing algorithm would imply lower computational cost making this kind of inference more feasible in real-time applications, specially if part of the processing, or all of it, is done within the mobile device, which can have limited computational resources, depending on the vendor and model.

Meanwhile, Knowledge Discovery in Databases (KDD) [45] has been successfully used in different scenarios to discover relevant knowledge from data. It can be understood as the overall process of discovering useful knowledge from data, encompassing steps for data selection, preprocessing, transformation, mining and interpretation. Depending on the domain in which it is applied, this process can make use of very different preprocessing techniques and machine learning (ML) algorithms.

Therefore, the work presented in this chapter intends to advance the state of the art, by proposing a new KDD method that can be used to perform travel mode detection and

trip purpose prediction, based on location traces obtained through smartphone sensors, using a single preprocessing method, reducing the complexity and the total processing cost of a real-time solution. The proposed method explores general and domain-specific preprocessing techniques, as well as supervised ML algorithms, with the use of automated machine learning (AutoML) techniques [64].

The proposed method was evaluated with regards to its performance, using the data collected by 19 smartphone users in the metropolitan area of Rio de Janeiro, during the Spring of 2017 and the results were compared with the ones obtained by an existing solution proposed in [126].

The contributions of this work are three-fold:

1. A KDD method for building real-time travel mode detection and trip purpose prediction models using the same preprocessing steps and AutoML techniques.
2. Evaluation of the models built, using the proposed KDD method, with respect to accuracy, precision, recall and F1-score, using real smartphone data.
3. Comparison of the models built, with the proposed KDD method, with baseline travel mode detection model proposed in a previous work [126].

Parts of these contributions were published in [127]. The rest of this chapter is organized as follows. Section 4.2 provides a summary of the most relevant related works. Section 4.3 presents the dataset used in this study and how it was obtained. Section 4.4 presents the proposed solution. Section 4.5 details the evaluation experiments settings and results. Section 4.6 discusses the results obtained. Finally, Section 4.7 outlines the main conclusions.

4.2 Related Works

In this section the most relevant works about travel mode detection, trip purpose prediction and joint travel mode and trip purpose identification are presented.

4.2.1 Travel Mode Detection

In an early work, the CityTracks [107] application was implemented as an iOS app and its main objective was to collect smartphone users' travel mode data through participatory and opportunistic sensing. The collected data was used to develop a complete

travel detection algorithm that performs the data preprocessing and segmentation, feature extraction and summarization, and travel mode classification through a hierarchical ML model. More recently, the CityTracks-RT [126] application for real-time travel mode detection was implemented, based on an adaptation of the algorithm developed in [107]. This application was implemented as an Android app and field-tested with 19 volunteers on the metropolitan area of the city of Rio de Janeiro, Brazil.

With a similar methodology, the work in [13] collected GPS, accelerometer and gyroscope data, and the travel mode used by smartphone users to create a real-time travel mode detection technique based on cascading of ML classifiers. The proposed technique was implemented in a prototype on the Android platform, called WAID (What Am I Doing). However, one of the limitations of this work was the lack of the prototype performance evaluation, since all the classification analysis was performed before the prototype development.

In [131], the authors collected data from accelerometer, gravity sensor, gyroscope, magnetometer and barometer to elaborate a technique for online travel mode detection with low energy consumption. The non-use of GPS allowed a great reduction in energy consumption and the proposed technique presented a good performance during the simulation of online detection. However, some of the sensors used, such as barometer and gravity sensor, are not available in most of today's smartphones, which restricts the use of this technique in a real scenario.

4.2.2 Trip Purpose Prediction

In [96] authors proposed a trip purpose prediction technique using random forests trained on GPS and accelerometer data collected by 156 participants, taking part in a one-week travel survey in Switzerland completed in 2012¹. Their following work [95] explored the effect of personalized training on trip purpose prediction accuracy and their latest work [94] compared the performance of travel data collection through dedicated GPS devices and smartphones. In their studies they utilize and improve a previously developed travel diary application [119] that performed travel mode detection and trip purpose prediction, although each task was performed based on a different set of features.

In [39] authors propose the use of online location-based search and discovery services, such as Google Places API to improve prediction. They apply nested logit and random forest to identify five trip purposes from trips collected during the 2010 Travel Behaviour Inventory in Minneapolis-St. Paul metropolitan area. A future work suggested by the

¹<http://www.project-peacox.eu/>

authors is the use of location data collected via smartphones (or GPS loggers) for model testing, rather than location data from traditional travel behaviour surveys [39].

Other recent works in this topic explored semi-automated trip purpose inference through participatory sensing [120], combination of GPS trajectories, POIs and social media data [92], data selection from different seasons for model training and testing [59], trip purpose detection with artificial neural networks with particle swarm optimization [148].

4.2.3 Joint Travel Mode and Trip Purpose Identification

In comparison with individual travel mode detection and trip purpose prediction, fewer works proposed techniques that could simultaneously perform both tasks. One of the earliest works to present that type of solution [129] proposed the use of heuristics based on features extracted from GPS logs, such as average speed and maximum speed, combined with GIS information, such as street lines and public transportation stops, to identify the travel modes and purposes of trips made by citizens in Sidney, Australia. The main disadvantage of their technique is that it is highly dependent on GIS information, which might not always be available, accurate and up to date.

In [19] authors proposed the use of GPS coordinates of trip ends combined with POI information derived from GIS data to identify the trip purpose and a mix of features derived from GPS speed and GIS data to identify travel modes. One of the main advantages of the technique proposed in the present work, in comparison to the one proposed by them, is the fact that it utilizes the same features, extracted from GPS logs, for both tasks, reducing the amount of preprocessing required and removing the dependence of GIS data availability.

A more recent work [56] proposed a solution that utilized GPS, Accelerometer, WiFi and Cellular Network extracted features combined with GIS derived features to train a Bayesian model for travel mode detection. The trip purpose identification was performed based on historical mobility patterns which were obtained through the most frequent routes and places each person moved to. The main limitation of this technique was the fact that it was meant to run in an offline manner, meaning that the travel mode detection and, especially, the trip purpose identification could not be done in real time. Also, the architecture proposed performed all data processing and classification in a back-end server, creating a dependency with Internet connectivity availability.

The latest solution was proposed in CTASS [11] and consisted of a digital framework for contextualized travel behaviour advice to cardiac patients that utilized the Moves

app [40] for non-motorized travel mode detection in conjunction of its Health Travel Behaviour (HTB) app that performed trip purpose identification and trip data collection. This work, however, did not detail which features and classification models were used for each detection task, neither presented a quantitative evaluation of its performance.

4.3 Study Data

In order to analyze travel mode and trip purpose together using location traces, a data collection experiment was conducted using an Android application, named CityTracks-RT. Therefore, in this section the operation of this application and the methodology applied during the field tests of this application are described.

The CityTracks-RT app is a data collection tool that performs real-time travel mode detection through five pre-trained classifiers. It stores preprocessed location data, user informed travel modes and inferences made, in the smartphone memory, while periodically transferring them to a centralized server in which the data is aggregated for latter analysis.

It was implemented as a prototype of the travel mode detection solution proposed in a previous work [126], whose operation is illustrated by Figure 3.1.

To allow further investigation of travel mode detection and trip purpose prediction, this application was modified to store and transfer the raw location data collected, and the trips purposes, to the server as well. No changes were made to the travel mode detection algorithm itself.

Each sample stored contained raw numerical attributes, such as latitude, longitude, altitude, precision, timestamp, measured speed and bearing in degrees, as well as calculated attributes, such as the speed, derived from the Euclidean distance between two consecutive traces divided by their timestamp difference, and the acceleration, derived from the calculated speed differences between two traces divided by their timestamp differences.

During the field tests, 19 smartphone users from the metropolitan area of Rio de Janeiro were asked to install the app, start data collection whenever they were commuting through one of the travel modes available on the app and manually inform the travel mode being used as well as the trip purpose.

The travel mode options supported by the app were:

1. Walking, *i.e.*, walking on foot,

2. Biking, *i.e.*, riding a bike,
3. Car, *i.e.*, driving a car,
4. Bus, *i.e.*, taking a bus.
5. Motorcycle, *i.e.*, riding a motorcycle.

The trip purpose options available for selection were:

1. Home, *i.e.*, going his/her home.
2. Work, *i.e.*, going to work place.
3. Education, *i.e.*, going to educational facility, *e.g.*, going to school.
4. Shopping, *i.e.*, going to shopping facility, *e.g.*, going to an outlet.
5. Leisure, *i.e.*, going to recreational facility, *e.g.*, going to the park.
6. Other, *i.e.*, none of the above.

The proportions of collected samples for each travel mode and trip purpose class are presented on Figure 4.1.

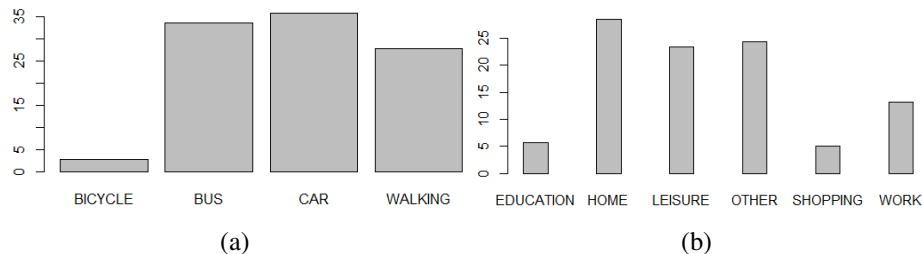


Figure 4.1: Frequency of location traces collected by travel mode (a) and trip purpose (b).

The collected travel modes were fairly distributed between car, bus and walking, since these are the most common ways of transportation in the city where this study was conducted, that were available at the app. A small amount of locations samples was collected by bicycle and none were collected by motorcycle.

With regards to trip purposes, most samples were collected while going home, to some leisure activity or other purpose that was not related to the alternatives present in the app. A smaller, but meaningful amount of samples was collected while going to work, shopping or to some education activity, such as going to the university.

Table 4.1 presents a summary of the data collected by each device. A common challenge on participatory sensing studies is being able to engage all volunteers in the data collection task so as to ensure data quality and quantity. Since the data collection was completely voluntary, meaning no monetary incentives were offered to the study participants, it is possible to notice that some of them were not as engaged as others, which can cause some bias on the final results of this study towards the mobility behaviours of the most committed users.

Table 4.1: Descriptive statistics of location traces by device.

Device ID	Samples #	Avg. Precision	Std. Precision	Travel Modes	Trip Purposes	Hours #
1	2472	8.06	7.19	2	1	1.27
2	3861	16.14	37.65	1	2	1.73
3	3942	21.54	34.52	1	1	4.14
4	1051	19.67	76.42	1	1	0.46
5	8134	4.55	4.91	3	4	2.95
6	93	55.57	40.00	1	1	0.06
7	1782	14.28	43.27	1	2	0.42
8	562	33.52	128.47	1	2	1.08
9	4675	27.38	109.70	1	2	1.00
10	27101	11.89	22.92	4	4	11.93
11	2120	11.35	17.77	3	3	10.32
12	1450	6.82	15.80	2	2	0.41
13	1403	86.35	166.66	2	1	0.80
14	3402	4.60	35.38	2	4	1.76
15	429	29.03	9.70	1	1	0.21
16	5836	32.90	54.93	3	4	2.67
17	1635	13.69	23.98	2	2	0.70
18	129	16.51	6.93	2	2	0.10
19	647	18.44	112.97	2	2	0.03
All	70724	22.75	45.96	4	5	42.04

With regards to the geographical distribution of collected data, Figure 4.2 displays the projection of collected locations coordinates on the Rio de Janeiro metropolitan area map. Locations were mostly collected on the main roads that connect the Downtown area with the North Zone, Niteroi city and South Zone, as well as within the Downtown area itself.

As a disclaimer, most of the volunteers were students from the Federal University of the State of Rio de Janeiro. The remaining volunteers were friends and relatives of the author. The author has also participated in the data collection activity. All volunteers received an illustrated manual that instructed them to start the data collection whenever they were commuting through one of the travel modes available in the app and selecting the travel mode option before the movement started. Whenever they switched modes, they should select another travel mode option, if it was available, or stop data collection, otherwise.

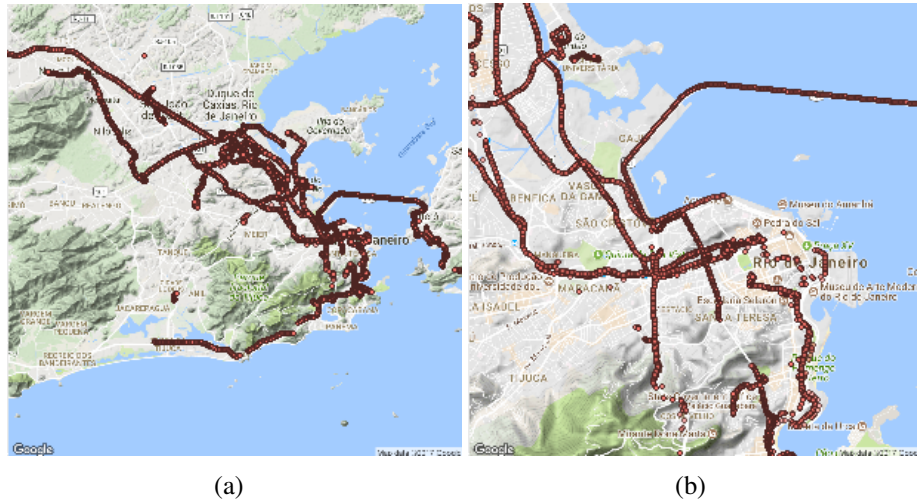


Figure 4.2: Projection of the location traces coordinates on the Rio de Janeiro metropolitan area (a) and Downtown region (b) map.

4.4 Proposed Solution

In this section the proposed solution for travel mode detection and trip purpose prediction is described with detail. First the preprocessing steps, that will be applied to the location traces in order to extract motion segments and relevant features, are explained. Then the AutoML techniques used to identify the best supervised ML algorithms for travel mode detection and trip purpose prediction is detailed. Figure 4.3 presents an overview of the proposed solution.

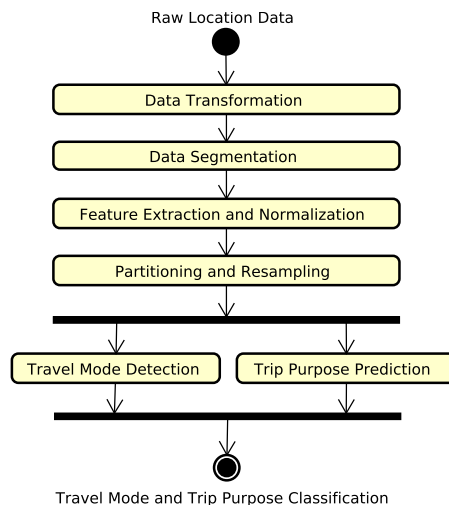


Figure 4.3: Overview of travel mode detection and trip purpose prediction solution.

4.4.1 Preprocessing

In order to allow the use of ML algorithms to try to identify a pattern between the information obtained by the smartphone sensors and the current travel mode and trip purpose some preprocessing steps were applied based on previous techniques [13, 100, 107, 126] and KDD best practices [45].

Data Transformation. The first step was to convert the locations traces stored in JSON files to a flat table containing the columns: *device_id*, *timestamp*, *travel_mode*, *trip_purpose*, *latitude*, *longitude*, *altitude*, *bearing*, *precision*, *calculated_speed*, *calculated_acceleration*, *measured_speed*. A total 70,724 table rows was obtained from the location files.

After that, two general filtering techniques were applied as well as two techniques specially designed for location traces preprocessing. The numbers of samples removed by each filtering technique are presented on Table 4.2.

Table 4.2: Number of location samples removed by each filtering technique.

Filtering Technique	Samples Removed #
Remove location samples with invalid attributes values	0
Remove duplicated location samples	0
Remove locations with precision above 50 meters	2,459
Remove locations within 1 minute of travel mode change	180
Total	2,639

The exclusion of imprecise location traces avoids training the ML models on noisy data and the 50 meter threshold has been used. Higher precision thresholds have been used on previous works [107, 126]. However, as demonstrated by Figure 4.4, the 50 meter threshold allows capturing the most relevant part of the samples distribution with regards to precision and guarantees that the locations considered on the data analysis and ML model training are as reliable as possible.

According to IEEE Spectrum² future smartphones will be able to capture locations with 30-centimeter maximum accuracy instead of today's 5 meter threshold. That indicates that smartphones would be able to capture locations on a much higher average precision, therefore, suggesting the use of a lower boundary on this filtering step.

The exclusion of location traces within 1 minute of each travel mode change attenuates the effect of data collection mistakes, since the data collectors might fail to inform travel mode changes right after or before they happen. In [90] a 20 second window was used, but as some travel modes might impose difficulties on informing the mode changes that quickly the window size of 1 minute was chosen.

²<https://spectrum.ieee.org/semiconductors/design/superaccurate-gps-coming-to-smartphones-in-2018>

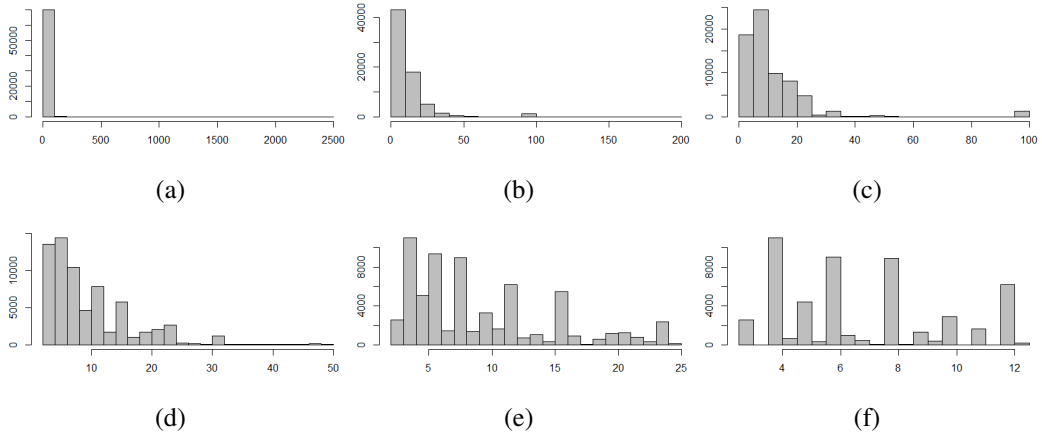


Figure 4.4: Locations frequency distribution by measurement precision: (a) All location traces, (b) Locations within 200 meters threshold, (c) Locations within 100 meters threshold, (d) Locations within 50 meter threshold (e), Locations within 25 meters threshold (f), Locations within 12.5 meters threshold (g).

A total of 2,639 location samples were removed leaving 68,084 location samples left to be used on the next preprocessing steps.

Data Segmentation. The second step was to extract the trip segments [122], also known as triplegs [100], of each trip from each device. A tripleg will correspond to a specific travel mode, as a trip will correspond to a single trip purpose. Also, the trip identification was made by using a timestamp difference threshold of 30 minutes, indicating that location traces with timestamps differences higher than this threshold will be considered as distinct trips. This threshold was used to avoid considering multiple trips collected with the same trip purpose as one single trip, which could possibly distort the results obtained. For the trip segmentation, *i.e.* tripleg identification, no threshold was used. Every time a travel mode change was detected within a trip, a new tripleg was considered to exist.

After that, constrained time segments were extracted from each tripleg using a window size of 60 seconds, denominated chunks [107]. In previous works, multiple window sizes have been used. In [126], the window size used was 90 seconds. The 60 seconds window size was chosen because it is considerably smaller than the 90 seconds of [107, 126] presenting an opportunity to improve the state of the art with respect to detection speed.

A total of 78 trips, 98 triplegs and 2,170 chunks were extracted from the 68,084 remaining location traces and submitted to the next preprocessing step in which several classification features were extracted from each chunk.

Feature Extraction and Normalization. In this step chunk features, that would be

useful to train a classification model, were extracted, using supervised ML, for predicting the travel mode and the trip purpose based on a single chunk. That type of classification has also been referred to as online travel mode detection and real-time trip purpose prediction [13, 39, 126, 131].

In that sense, a subset of the features extracted in previous works [13, 107, 126] was selected and two additional features were included in order to improve classification accuracy. These attributes were the weekday in which each chunk was collected and the period of the day extracted from on its locations timestamps. These two attributes might be very useful since the travel modes and trip purposes can be highly correlated to them. To analyze these correlations as well as the correlation between the travel modes and the trip purposes the Pearson Correlation Test was executed using the R function *cor.test*.

Table 4.3 presents the results of those tests. Degrees of freedom were omitted from the table since they had same value for all tests (2,168).

Table 4.3: Pearson correlation tests of chunk categorical attributes.

Explanatory Variable (x)	Response Variable (y)	Sample Estimated Correlation (r)	T-Test Statistic (t)	Significance Level (p)
Travel Mode	Trip Purpose	-0.188877	-8.9556	<2.2e-16
Weekday	Trip Purpose	0.0312288	1.4548	0.1459
Period of Day	Trip Purpose	-0.01660698	-0.77336	0.4394
Weekday	Travel Mode	-0.1481358	-6.9744	4.063e-12
Period of Day	Travel Mode	0.05862079	2.7342	0.006304

As the only tests that presented a significance value below 0.05 were the ones that analyzed the correlation between the travel mode with trip purpose, and the correlation between the weekday and the period of the day with the trip purpose, it can only be assumed that the true correlation is different than zero for these variables. That being said, the highest estimated correlation (r) was between travel mode and trip purpose, which indicates that these variables are more correlated, within the dataset of this study, than the others. Provided that, it was chosen not to consider them as predictors of one another in the evaluation experiments to reduce the bias of the results obtained. Also, as these travel mode and trip purpose predictions could occur in parallel, it would not make sense to use one as input to the other in the proposed solution.

In Figure 4.5 scatter plots that indicate some intuitive thresholds of feature values for each travel mode are presented as well as the relationships between the categorical variables and the trip purposes collected.

Partitioning and Resampling. In order to generate training and test datasets for travel mode detection and trip purpose prediction, two copies of the original dataset were generated, denominated *travelModeDataset* and *tripPurposeDataset*. To mitigate the bias

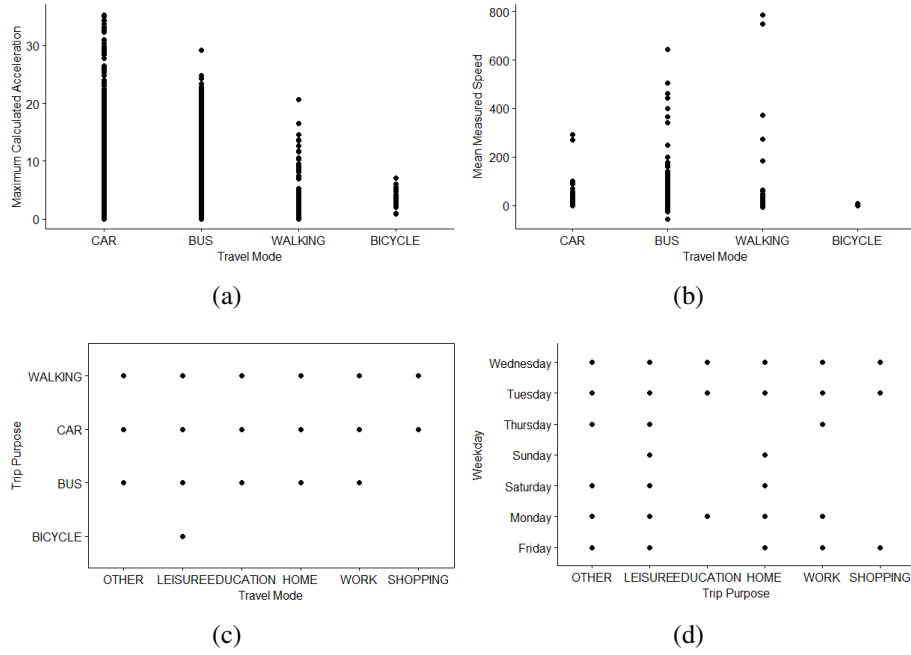


Figure 4.5: Scatter plot for travel mode relationships with maximum calculated acceleration (a) and mean measured speed (b). Scatter plots for trip purpose relationships with travel mode (c) and weekday (d).

generated by the high correlation observed between travel modes and trip purposes the trip purpose column was removed from *travelModeDataset* and the travel mode column was removed from the *tripPurposeDataset*.

This correlation could, however, be explored in a larger study, as the detected travel mode could be used as feature for detecting the trip purpose and vice-versa. After that, subsets of the dataset samples was randomly assigned for training and testing, generating four new datasets *trainingTravelMode*, *testingTravelMode*, *trainingTripPurpose* and *testingTripPurpose*.

Some standard preprocessing techniques were also applied at this step, such as replacing NA values with 0, removing ID columns and normalizing the numerical features to a common order of magnitude, since numerical features with higher order can induce training bias on the ML algorithm, which could distort the classification results.

Since the dataset used in this study is unbalance with respect to travel mode classes, applying ML algorithms directly to them might create some bias towards the classes with the larger amount of samples. Provided that, three different resampling techniques were applied to reduce this bias: up-sampling, down-sampling and synthetic minority over-sampling (SMOTE). These techniques allowed the generation of more balanced datasets and the performance comparison of the ML algorithms with each one of them.

The up-sampling technique replicates samples from the less frequent classes until the number of samples for each class is equal. Meanwhile, the down-sampling technique removes samples from the most frequent classes until their count is equal to the less frequent class. The SMOTE technique is the most used and most effective, since it generates synthetic samples that maintain the feature values distribution of each class. Those techniques were applied through the functions available on the caret package³.

4.4.2 Classification

In the classification phase, instead of basing the ML algorithms and parameterization choices on statistical intuition or previous works, two different AutoML techniques were applied to identify the best ML algorithm and configuration. This method is summarized by Figure 4.6 and is described with detail in the remainder of this section.

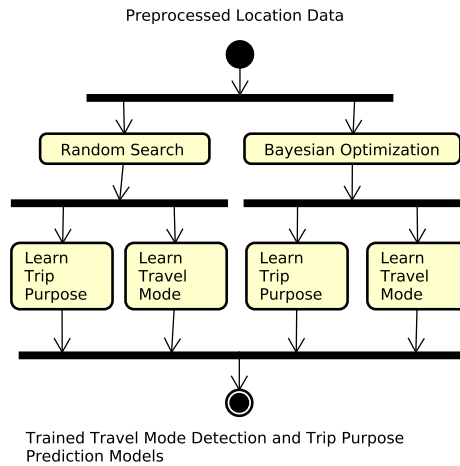


Figure 4.6: Steps used to find the best classifiers for travel mode detection and trip purpose prediction.

Random Search. In order to perform travel mode detection and trip purpose prediction the *h2o.automl* framework was used to do a random search for the best classifier and its most efficient hyper-parameter configuration.

The framework used selects the best classifier based on a single performance metric, whose default option is the log-loss, which is defined by equation:

$$\text{LogLoss} = - \sum_{c=1}^M y_{o,c} \log(p_{o,c}); \quad (4.1)$$

where M is the number of possible class labels, \log is the natural logarithm, y is a bi-

³<https://topepo.github.io/caret/subsampling-for-class-imbalances.html>

nary indicator (0 or 1) of whether class label c is the correct classification for observation o and p is the model’s predicted probability that observation o is of class c , as defined in [60]. The classifiers considered on the search space were Deep Neural Network [60], Distributed Random Forest [58], Generalized Linear Model [99], Gradient Boosting Machine [51], Naive Bayes Classifier [97], Generalized Low Rank Models [136], Stacked Ensemble [144], XGBoost [29], Cox Proportional Hazards [6]. The only parameter that was configured for the search was the maximum search time, which was set to 5 minutes.

Bayesian Optimization. As an alternative for the Random Search, the Bayesian Optimization technique was used through the *AutoWEKA* package for WEKA 3.

This package applies Bayesian Optimization through the SMAC algorithm [76] to find the best classifier and hyper-parameter configurations through exploration of the whole parameter space and exploitation of good configuration spaces in relation to a performance metric, whose default option is the error rate, which is defined by equation:

$$\text{ErRate} = 1 - \sum C_i \neq C_{\max, x} \int_{x \in H_i} P(C_i|x)p(x)dx; \quad (4.2)$$

where x is an instance, C_i is a class into which an instance is classified, H_i is the area/region that a classifier function h classifies as C_i [53].

In contrast with more classic AutoML approaches, such as grid search and random search, Bayesian Optimization is considered to be more efficient on large search spaces. *AutoWEKA* was one of the first implementations of this technique for ML frameworks and still one of the most complete ones, with 28 learners implemented [76].

In order to allow the comparison of this technique with the random search the same time limit was used on both, which was set to 5 minutes.

4.5 Performance Evaluation

In this section the performance of the proposed solution as well as the CityTracks-RT real-time travel mode detection solution are evaluated. First, the travel mode and trip purpose classification performance of the proposed solution with Random Search and Bayesian Optimization is evaluated using 70% of data for training and 30% for testing. Then, the proposed solution performance using Bayesian Optimization is evaluated in a 10-fold cross-validation experiment. Lastly, the performance of the CityTracks-RT solution is evaluated using 10-fold cross-validation as well. In addition, the performance

metrics of the real-time detection performed by CityTracks-RT during the field tests are presented.

4.5.1 Evaluation Metrics

The classification performance metrics considered in this analysis are *accuracy*, *precision*, *recall* and *F1-score* [44]. Those metrics were extracted for each travel mode class and the average per class values are reported for each classifier. The average values are weighted by the number of samples of each class.

4.5.2 Proposed Solution with Random Search

When doing random search, the *h2o-automl* framework selected the best model for each training dataset and their performances were evaluated on the test dataset using metrics derived from the confusion matrix. Tables 4.4 and 4.5 present the best travel mode and trip purpose classifiers, respectively, their overall accuracy, precision, recall and F1-score on the test set, for each resampling technique.

Table 4.4: Performance metrics of the best classifiers found through Random Search on the test set for travel mode detection.

Sampling Technique	Best Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Distributed Random Forest	78	69	67	67
Up-sampling	Gradient Boosting Machine	86	94	61	71
Down-sampling	Gradient Boosting Machine	69	60	53	49
SMOTE	Gradient Boosting Machine	76	65	63	63

Table 4.5: Performance metrics of the best classifiers found through Random Search on the test set for trip purpose prediction.

Sampling Technique	Best Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Gradient Boosting Machine	80	52	51	50
Up-sampling	Distributed Random Forest	76	-	23	-
Down-sampling	Distributed Random Forest	78	45	43	43
SMOTE	Distributed Random Forest	79	50	47	47

4.5.3 Proposed Solution with Bayesian Optimization

When applying Bayesian Optimization through *AutoWEKA*, better results have been achieved, as expected. Tables 4.6 and 4.7 present the performance metrics for the best performing classifiers on the travel mode classification and trip purpose prediction for each resampling technique on the test set.

The large gap observed between the results obtained with Bayesian Optimization in comparison with Random Search, suggests that the 5 minutes search time was not enough

Table 4.6: Performance metrics of the best classifiers found through Bayesian Opt. on the test set for travel mode detection.

Sampling Technique	Best Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Support Vector Machine	78	78	78	78
Up-sampling	Random Forest	86	86	86	86
Down-sampling	Decision Tree	83	85	83	82
SMOTE	Ada Boost	99	99	99	99

Table 4.7: Performance metrics of the best classifiers found through Bayesian Opt. on the test set for trip purpose prediction.

Sampling Technique	Best Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Random Forest	96	96	96	96
Up-sampling	Random Forest	89	90	89	89
Down-sampling	Random Committee	99	99	99	99
SMOTE	Random Forest	98	98	98	98

to allow the Random Search to find good regions of the hyperparameter space while the sequential nature of the Bayesian Optimization procedure allowed it to find the good regions within this same search time.

Therefore, in order to perform a more reliable evaluation of the best classifiers obtained through Bayesian Optimization, a 10-fold cross-validation was executed with these models. Tables 4.8 and 4.9 present average per fold performance metrics for the travel mode and trip purpose classification.

Table 4.8: Average per fold performance metrics of travel mode detection on the 10-fold cross-validation.

Sampling Technique	Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Support Vector Machine	76	76	76	76
Up-sampling	Random Forest	88	88	88	88
Down-sampling	Decision Tree	70	70	70	70
SMOTE	Ada Boost	83	82	83	82

4.5.4 Baseline Solution

The CityTrack-RT travel mode detection algorithm applied a hierarchical classification strategy that used a Support Vector Machine [32] classifier to differentiate between motorized and non-motorized travel modes and a Decision Table classifier to identify which motorized or non-motorized travel mode was being used. Two additional classifiers worked in parallel with the main classification algorithm. One of them was a Bayesian Network classifier [52] which performed only non-motorized travel mode classification. The other one was a Multilayer Perceptron classifier [60] which tried to identify which travel mode was being used without the hierarchical approach.

Table 4.9: Average per fold performance metrics of trip purpose prediction on the 10-fold cross-validation.

Sampling Technique	Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
None	Random Forest	69	69	69	69
Up-sampling	Random Forest	81	80	81	80
Down-sampling	Random Committee	67	66	67	66
SMOTE	Random Forest	77	77	77	77

In order to compare this solution with the one proposed in this chapter, a 10-fold cross-validation is performed using the CityTracksRT models. Table 4.10 summarizes the overall performance metrics obtained by each of the classifiers used in the hierarchical classification strategy, and the Multilayer Perceptron classifier.

Table 4.10: 10-fold cross-validation performance metrics of the classifiers used in the CityTracks-RT application.

Classes #	Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
5	Multilayer Perceptron	51	48	51	48
2	Support Vector Machine	68	47	68	55
4	Decision Table	77	76	77	76
2	Bayesian Networks	91	88	91	87

In addition, Table 4.11 summarizes the performance of the real-time travel mode detection during the field tests of the CityTracks-RT app. The main difference is that in this evaluation, the algorithms were trained only once, with a fix set of samples, which were collected before the field tests.

Table 4.11: Performance metrics of the classifiers used in the field tests of CityTracks-RT application.

Classes #	Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
4	Multilayer Perceptron	34	31	28	29
2	Support Vector Machine	58	61	54	57
4	Decision Table	59	40	42	41
2	Bayesian Networks	88	44	50	47

4.6 Discussion

In this section the proposed method is compared with the CityTracks-RT solution and the main strengths and weaknesses of both techniques are discussed. Also, the impact of varying windows sizes on the travel mode detection and trip purpose prediction performance of the propose technique are analyzed.

4.6.1 Travel Mode Detection

With respect to travel mode detection, the proposed method presented better overall performance on the 10-fold cross-validation experiment reaching a maximum accuracy, precision, recall and F1-Score of approximately 88% while classifying 4 different travel modes with the Up-Sampling technique and Random Forest classifier. All other resampling techniques presented metrics of at least 70% which indicates a good level of generalization reached with any of them.

Table 4.12 presents the confusion matrix of the best performing travel mode classifier, through which it can be observed that the classifier was able to detect bicycle, car and walking segments very effectively, while a significant proportion of Bus segments had been classified as car segments. This is likely due to the similarities between both mode patterns depending on the traffic conditions in which the samples had been collected.

Table 4.12: Confusion matrix of best performing travel mode classifier.

<i>True \ Predicted</i>	Bicycle	Bus	Car	Walking
Bicycle	858	0	0	0
Bus	0	669	123	66
Car	4	83	729	42
Walking	0	61	31	766

The CityTracks-RT solution achieved a higher performance only when classifying non-motorized travel modes with Bayesian Networks reaching a 91% accuracy. Since the hierarchical classification algorithm depended on the Support Vector Machine and Decision Table classifications its overall performance cannot be higher than the 76-77% metrics obtained by the latest. The Multilayer Perceptron was the worst performing classifier, presenting metrics between 48 and 50%, in contradiction with previous performance evaluation results [126].

The performance disparity between CityTracks-RT solution and the proposed solution might be inflated by the fact that no parameter tuning was performed on the former, as the authors allegedly used default WEKA parametrization [126]. By applying AutoML techniques it was possible to automatically test several hyper-parameter configurations on each classifier and choose the best one based on a loss function.

4.6.2 Trip Purpose Prediction

With respect to trip purpose prediction, our method was able to identify the trip purpose with a maximum accuracy, precision, recall and F1-Score of 81% using Up-Sampling technique and Random Forest classifier also. For this classification problem the

Down-sampling technique and the direct classification (*i.e.*, without resampling) did not reach the 70% threshold on any performance metric, which suggests that for this classification problem the SMOTE or Up-Sampling techniques are more appropriate.

Table 4.13 presents the confusion matrix of the best performing trip purpose classifier, through which it can be observed that the classifier was able to predict the purpose of trips to Education, Shopping and Leisure facilities very effectively. The category with the highest miss rate was Home, which was very often confused by the classifier with the Other category. This indicates that the samples of these two categories had similar feature distributions.

Table 4.13: Confusion matrix of best performing trip purpose classifier.

<i>True \ Predicted</i>	Education	Home	Leisure	Other	Shopping	Work
Education	539	7	0	2	0	0
Home	31	358	42	68	14	35
Leisure	12	39	439	21	12	25
Other	29	77	20	380	15	27
Shopping	0	2	0	2	544	0
Work	27	55	36	29	4	397

Since the CityTracks-RT application did not perform trip purpose prediction no comparison can be made on that sense. One interesting aspect that could be explored in future research is the use of location-based search and discovery services to improve trip purpose prediction based on smartphone location traces, as proposed by [39].

4.6.3 Time Window Size

In order to analyze the impact of different windows sizes in the classification performance of the best hyperparameter configurations of the proposed solution, using Up-Sampling and Random Forest classifier, an additional experiment was conducted, using window sizes of 30, 45, 60, 75 and 90 seconds. Through these experiments it can be concluded that the optimal window size for travel mode detection was of 60 seconds, and for trip purpose prediction was of 30 seconds. In Figure 4.7 it can be observed that the performance improvement obtained in the travel mode detection by increasing the window size from 30 to 60 seconds, about 6%, is more significant than the one obtained in trip purpose prediction when reducing the window size from 60 to 30 seconds, about 1%.

4.7 Conclusions

In the present chapter a hybrid solution for real-time travel mode detection and trip purpose prediction was proposed. This solution uses a single preprocessing algorithm

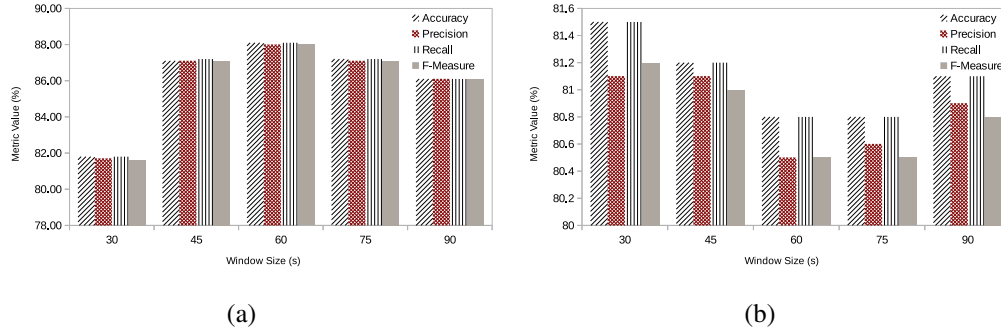


Figure 4.7: Performance metrics per window size for travel mode detection (a) and trip purpose (b).

to extract features that are used to train classification models through supervised ML algorithms.

Four resampling techniques and two AutoML methods were applied to identify the best ML algorithm and its best hyperparameter configurations for each resampling technique. Their performance with respect to accuracy, precision, recall and F1-Score was evaluated using k-fold cross-validation, with k=10.

The results show that the best ML algorithm in most cases is Random Forest and the most indicated resampling technique is Up-Sampling. The most efficient automated ML technique within the five minute search time was Bayesian Optimization and the maximum accuracy reached for travel mode detection and trip purpose prediction on the cross-validation experiments was 88% and 81%, respectively.

Therefore, it can be concluded that the proposed solution is capable of detecting the travel mode and predict the trip purpose of a trip using only 60 seconds of smartphone collected location data, allowing context-aware information systems to use this contextual information to provide personalized services and better predict user behaviour with respect to mobility patterns and transportation preferences.

5. Real-Time Travel Mode Detection with Multiple Sensors

5.1 Introduction

Almost all smartphones and other mobile devices currently include multiple sensors. Allied with the increasing processing power and battery autonomy of these devices, these sensors allow the discovery of important context information such as the travel mode being used, enabling many context-aware applications. Also, the fact that most people that live in urban centers carry their smartphones to almost everywhere they go enables continuous sensing through these devices, which allows greater data collections and ubiquity in the services provided through them.

In the last years, many smartphone-based travel mode detection techniques have been developed. Most of them apply supervised machine learning (ML) algorithms, in conjunction with advanced signal processing techniques (*i.e.*, preprocessing techniques), to extract relevant features from the data collected through sensors that are embedded in these devices [106]. After a literature review, it was noted that although most of these works have proposed innovative preprocessing techniques and ensembles of ML models, few of them explore the usage of dimensionality reduction techniques in conjunction with ML hyperparameter optimization. If applied properly, dimensionality reduction techniques can greatly reduce ML model training cost (*i.e.* *time, CPU and battery*) without compromising classification performance. Meanwhile, ML hyperparameter optimization can improve ML model accuracy by finding the best possible configuration used for training.

In this chapter, an attempt to fill this gap is made by proposing the use of a well known dimensionality reduction technique, the Principal Component Analysis (PCA), and an Automated Machine Learning (AutoML) framework, the AutoSklearn [48], which performs ML hyperparameter optimization, among other features. The improvement that these methods can provide in classification accuracy and training cost by applying them

is evaluated over a travel mode detection technique (known as US-TransportationMode¹) and analyzing the resulting performance improvement for 12 different scenarios. Experiments used real smartphone data from the TMD Dataset ², which a benchmark dataset for travel mode detection built by researchers from University of Bologna. Statistical measures, such as skewness and kurtosis, were extracted along a sequence of time windows, and the improvement observed with their addition to the classification model was evaluated as well. Thus, the research hypothesis investigated in this study are as follows:

- *Hypothesis 1.* AutoML frameworks can generate more accurate classification models for travel mode detection than human experts. Automated Machine Learning has been proved to outperform human experts in several domains [15, 75, 124]. Therefore, while its expected that it would generate better classification models for travel mode detection, none of the related works presented this kind of evaluation and comparison.
- *Hypothesis 2.* PCA can reduce classification cost in a multi-sensor travel mode detection system without significant losses in model accuracy. While in [90] authors have evaluated the impact of PCA for dimensionality reduction, their input data was based only on GPS and accelerometer data. The dataset used in this study contains data from 9 different sensors, from which up to 54 features were extracted what can lead to new and richer results.
- *Hypothesis 3.* Skewness and Kurtosis, extracted from smartphone sensors measurements for a sequence of time windows, can be used as features to classify modes of transportation. Previous work [89] have evaluated this hypothesis to be false in an experiment where only features extracted from GPS traces were considered, at a granularity of one sample/min. In the present work however, a higher sampling rates and a wider array of sensors, such as accelerometer, microphone and gyroscope, was used, what can lead to different conclusions.

In order to verify these hypotheses, performance metrics are analyzed using k-fold cross-validation (k=10) [74] to identify the most suitable combination of feature engineering techniques and ML configurations for the problem of detecting travel modes, in real time, based on smartphone sensors. This is the first work to present evidence that AutoML frameworks can outperform human experts in the combined algorithm selection and hyperparameter optimization of ML classifiers for the travel mode detection problem.

¹<https://github.com/vlomonaco/US-TransportationMode>

²<http://cs.unibo.it/projects/us-tm2017/index.html>

The remainder of this chapter is structured as follows. Section 5.2 brings related works and Section 5.3 analyzes the US-TransportationMode technique, characterizing the TMD Dataset and conceptualizing the ML techniques that were used. Section 5.4 describes feature engineering techniques, including dimensionality reduction using PCA. Section 5.5 describes and discusses the evaluation experiments and their results. Section 5.6 presents conclusions of this chapter.

5.2 Related Works

Table 5.1 summarizes related works, which indexes are shown in column 1. Next three columns show respective number of sensors, number of extracted features and number of detected travel modes for each related work. Column five lists the evaluated ML algorithms: BN - Bayesian Networks, NB - Naive Bayes, SVM - Support Vector Machine, MLP - Multilayer Perceptron, DT - Decision Table, RF - Random Forest, RT - Random Tree, KNN - k-Nearest Neighbours, MNL - Multinomial Logistical Regression, NL - Nested Logit, MDA - Multiple Discriminant Analysis, LR - Logistical Regression, DBN - Dynamic Bayesian Network, CNN - Convolutional Neural Network, BC - Bayesian Classifier and AE - Autoencoder. Column six (AC - Algorithms Comparison) states whether the related work compares ML algorithms performance. Column seven (PE - Parameter Explanation) states whether it elicits which hyperparameter values were used for training. Finally, column eight (PO - Parameter Optimization) states if the related work explicitly used any hyperparameter optimization technique.

In [107], authors applied a KDD process [45] to develop an offline travel mode detection technique using location data collected from GPS, WiFi and Mobile Networks. The main contributions of this work were the developed preprocessing techniques in order to remove noisy samples and to augment data resolution, as well as the use of a hierarchical classification model built with supervised ML algorithms, such as SVM and DT. Performance evaluation used data collected from real smartphones. Main limitations were the use of WEKA's [142] default machine learning algorithms' configurations and the fact that it was not able to perform online travel mode detection. This second gap was filled by [126], which adapted the technique to perform online travel mode detection. The new technique was implemented on a prototype application and its performance was evaluated through field tests. The maximum accuracy was achieved using a Multilayer Neural Network classification model instead of the hierarchical model proposed in [107]. Despite the incremental contribution of this work, the lack of machine learning parameters optimization remained, since it also used default configurations available in WEKA Suite [142].

In [13], authors proposed an online travel mode detection technique using accelerometer and gyroscope data, based on cascading ML classifiers [3]. It was implemented in a prototype application, but performance evaluation was made through 10-fold cross-validation instead of field tests. The main gap is the fact that authors did not clarify which parameter configurations were used for machine learning algorithms and whether any optimization technique was applied.

In [131], authors developed an energy efficient online travel mode detection technique using data from accelerometer, gravity sensor, gyroscope, magnetometer and barometer. They proposed the use of Primal Estimated Sub-Gradient Solver (Pegasos) with SVM to allow online learning as well. This technique showed great results in simulation experiments with training sets of varying sizes, but neither cross-validation or field tests were performed. Also, authors did not clarify which parameters configurations were used for SVM itself and if some form of hyperparameter optimization was performed.

In [43], authors propose several features based on accelerometer, magnetometer and gyroscope sensors for offline travel mode detection and compare classification performance using DT, kNN and SVM. The main limitation is that it neither specified which hyperparameter values were used for these classifiers or disclaimed if any optimization technique was used. In [121], accelerometer and gyroscope data are used to develop an offline travel mode detection algorithm using RF. Authors state that most hyperparameter were set to default values of the "randomForest" package implemented in R³. The only exception was the number of trees, which was set to 100. Main limitations are the fact that it did not use any optimization technique to identify best hyperparameter values and that it does not compare the RF classifier performance with any other algorithm. In [8], authors also use accelerometer and gyroscope data as well as GPS information to develop an offline travel mode detection algorithm using MNL, NL and MDA for classification. Although authors clarify which hyperparameter values were used, it is not clear whether any optimization technique was used for selecting them.

In [28], authors present a technique that combines smartphone Hall-Effect magnetometer and accelerometer data to detect users' mode of transportation in real time. It applies advanced signal processing techniques and extracts features from each spectrum of the Fast-Fourier Transform(FFT) of each second of movement, such as entropy, energy ratio and spectral coefficients for each second of movement. Authors decided to build a two-layer classifier using RF to distinguish stationary and non-stationary periods, and NN to identify which travel mode was being used on non-stationary periods. Although

³<https://cran.r-project.org/web/packages/randomForest/randomForest.pdf>

this work clarified which parameters were used on each model and applied grid search to optimize ML algorithm selection and hyperparameter configuration, it did not present the parameters that were used for the grid search itself.

In [90], authors proposed an online travel mode detection technique that combined GPS and accelerometer. They applied the “movelets” [9] technique for preprocessing, PCA and Recursive Feature Elimination (RFE) for dimensionality reduction, k NN and RF for classification, and 10-fold cross validation for performance evaluation. With respect to hyperparameter optimization, authors performed a grid search based on accuracy for the best k values of the k NN model, with k ranging from 1 to 100. For the RF model, authors clarify which hyperparameters were used but do not elicit whether any optimization was performed. Any comparison between different ML algorithms hyperparameter configurations was presented.

In [10] and [139], authors augmented GPS data with socioeconomic information from users to improve offline travel mode detection. In [10], dynamic Bayesian Networks were built for classification and their performance was evaluated on five datasets and compared to SVM, RF and MLP, whose hyperparameters were optimized using exhaustive grid search, although their values are not informed in the paper. In [139], RFs are used to build the main classification model and their performance is compared to MLP and SVM models. Authors do not inform which hyperparameters values were used on each ML algorithm and do not clarify whether any optimization technique was used.

In [89], authors compared several feature extraction techniques of different ML approaches, including Autoencoders for travel mode detection using GPS traces. They did not, however, clarify which hyperparameters values were used, neither elicit the use of optimization techniques for their choices. Following the deep learning trend, in [34] authors propose its use for offline travel mode detection based on GPS trajectories, removing the need of feature engineering techniques. They compare multiple CNN architectures and present performance metrics for each configuration. Although this work has elicited most of the used hyperparameter values, default parameters were used for the back-propagation algorithm (*i.e.* Adam [72]) and no comparison was made between different parametrization of this algorithm.

After reviewing the literature, it is clear that most works in travel mode detection research do not clarify the hyperparameter used for machine learning algorithms training, what can greatly decrease the level of research reproducibility on each of them. Also, even fewer works explicitly apply any hyperparameter optimization technique, which could greatly increase performance of the models built. Finally, a short amount of related works

Table 5.1: Summary of related works.

Ref.	#Sensors	#Features	#Modes	ML Algorithms	AC	PE	PO
[107]	3	3	5	BN, NB, SVM MLP, DT, RF, RT, K-Means, KNN, Ada Boost	✓	✗	✗
[126]	3	3	4	MLP, SVM, DT, BN	✓	✓	✗
[13]	3	12	7	RF, DT, BN, RT, SVM, NB, Cascading, Bagging, Boosting, Voting, Stacking	✓	✗	✗
[131]	5	61	6	BN, Pegasos (SVM)	✓	✗	✗
[43]	3	14	5	DT, KNN, SVM	✓	✗	✗
[121]	2	31	4	MNL, NL, MDA	✓	✗	✗
[28]	2	136	7	SVM, LR, Boosting, RF, NN	✓	✗	✓
[90]	2	140	5	KNN, RF	✓	✓	✗
[10]	1	10	4	DBN, SVM, RF, MLP	✓	✗	✓
[139]	1	7	5	RF, MLP, SVM	✓	✗	✗
[89]	1	20	4	BC, RF, MLP, AE	✓	✗	✗
[34]	1	4	5	CNN, KNN, SVM, DT, RT, MLP	✓	✗	✓

apply dimensionality reduction to reduce ML training cost.

5.3 Travel Mode Detection Technique used in a Public Benchmark Dataset

The dataset used in this study is the TMD Dataset, made available by researchers from University of Bologna in order to be used as a benchmark for travel mode detection through smartphones. In this section is described the travel mode detection technique that was implemented by these researchers, namely US-TransportationMode, and made available with this dataset for further improvements and research reproducibility. Also, this dataset is characterized with descriptive statistics and the main limitations on the performance evaluation methods used by the authors are discussed, as well as improvements that provided in the present work.

5.3.1 Travel Mode Detection Technique

The US-TransportationMode technique has an initial data preprocessing phase in which data cleaning operations are performed, such as deleting measures from the sensors to exclude and make the values of the sound and speed sensors positive.

Furthermore, sensors that returned a single data value as the result of sense were directly used, while sensors that returned more than one value could not be used directly, since most of them were based on a coordinate system which was dependent on smart-

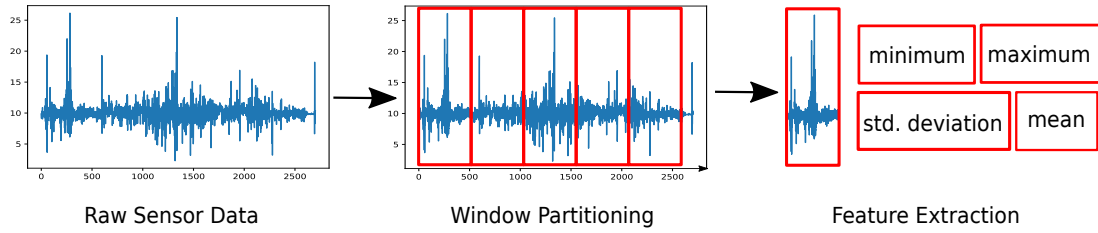


Figure 5.1: Preprocessing steps of US-TransportationMode travel mode detection technique. Adapted from <http://cs.unibo.it/projects/us-tm2017/>.

phone orientation.

Some sensors in a smartphone return a single value and others return multiple values. The US-TransportationMode technique uses an orientation-independent metric called magnitude [12, 24, 135] that is applied to sensors based on a coordinate system that return multiple values.

Given the values of sensor s on the x , y , and z axes, which are represented by $v_{x,s}$, $v_{y,s}$ and $v_{z,s}$, the magnitude is obtained by Equation 5.1:

$$\text{magnitude}(s) = |v_s| = \sqrt{v_{x,s}^2 + v_{y,s}^2 + v_{z,s}^2} \quad (5.1)$$

The technique divides the dataset into 5 seconds non-overlapping time windows and four features (*i.e.*, maximum, minimum, mean and standard deviation values) are extracted from all sensors. Figure 5.1 illustrates the preprocessing steps using the accelerometer readings of one of the volunteers as an example.

After preprocessing phase is completed, extracted features are fed into a supervised machine learning algorithm for training and prediction using scikit-learn python library⁴. Four ML algorithms were used: Decision Trees [69], Random Forest [69], Support Vector Machine [69] and Neural Networks [37].

5.3.2 Public Benchmark Dataset

In order to build the TMD Dataset, smartphone sensors data were collected by fifteen volunteers and classified into five different travel modes: walking, car, still, train and bus. The dataset is composed of a total of 226 labelled files representing activities corresponding to more than 31 hours of data: 26% of data is annotated as walking, 25% as driving a car, 24% as standing still, 20% as being on train, and 5% as being on bus. Figure 5.2

⁴<http://scikit-learn.org/stable/>

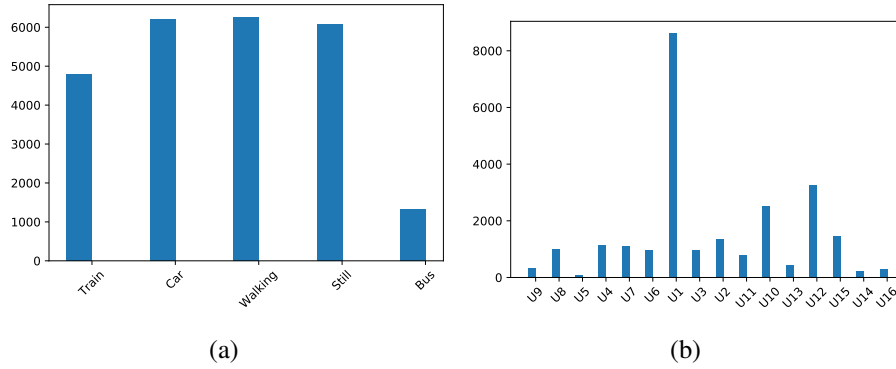


Figure 5.2: Histogram of total number of samples collected by travel mode (a) and user (b).

Table 5.2: Sensors considered on each Sensor Set.

Sensor Set	Sensors
1	Accelerometer, Sound and Gyroscope
2	Accelerometer, Sound, Orientation, Linear Acceleration, Gyroscope Uncalibrated, Gyroscope, Game Rotation Vector and Rotation Vector
3	Accelerometer, Sound, Orientation, Linear Acceleration, Gyroscope Uncalibrated, Gyroscope, Game Rotation Vector, Rotation Vector and Speed

illustrates the amount of samples collected per mode and by each user.

For the data collection task, volunteers used an Android application that allowed them to record their name, to start and stop the data collection and to label the travel mode being used. The application also sampled 23 different sensors with a maximum frequency of 20 Hz. Raw sensed data sampled on the device was then saved to be later transferred to researchers via a USB connection. Since many devices did not support all kinds of sensors, most of them were excluded and 9 sensors remained on the final dataset: Accelerometer, Sound (Microphone), Orientation, Linear acceleration, Speed (GPS), Gyroscope, Rotation vector, Game rotation vector and Gyroscope uncalibrated.

5.3.3 Discussion

For the US-TransportationMode technique, authors evaluated and compared travel mode detection performance using three sensor configurations described in Table 5.2.

The performance of each classifier and sensor set combination was evaluated using the accuracy metric, which is classically defined by Equation 2.14.

A positive inference means that an instance in a given time window is classified as belonging to a given class, and a negative inference means it is classified as not belonging

to that class. In other words, the accuracy gives the fraction of correct classifications. During the evaluation experiments, authors used 50% of collected data for model training and 50% for testing, reaching a maximum accuracy of 96% using all sensors available and the Random Forest classifier, as observed on Table 5.3.

Table 5.3: US-TransportationMode original performance evaluation results.

Algorithm	Sensor Set	Accuracy (%)
Decision Tree	1	82
Decision Tree	2	86
Decision Tree	3	91
Random Forest	1	88
Random Forest	2	93
Random Forest	3	96
Support Vector Machine	1	85
Support Vector Machine	2	93
Support Vector Machine	3	95
Neural Network	1	85
Neural Network	2	92
Neural Network	3	95

The main weakness of this performance evaluation is the fact that only the accuracy metric is analyzed, which might not give enough insights about the classifiers' specificity and sensitivity. For such, the precision, recall and F1-Score metrics should be also analyzed [105]. Besides, a KDD-based approach [45] would suggest the use of k-fold cross-validation (with k=10) to provide more reliable results. In the present work, new results for this technique are obtained using the mentioned metrics and validation techniques.

With respect to the travel mode detection technique itself, the main issues are the fact that only low-order statistical features [66] were extracted, without applying any kind of feature selection or dimensionality reduction. In addition, no hyperparameter optimization technique was used on the ML model training, which could greatly improve model accuracy.

In the present work, it is proposed to improve this technique with the use of PCA for dimensionality reduction, Skewness and Kurtosis metrics for identifying high-order statistical patterns and AutoML for combined algorithm selection and hyperparameter optimization. Also, the performance of the proposed technique, that extends the US-TransportationMode technique, is evaluated and compared with the original one in order to identify the main benefits and drawbacks of both techniques.

5.4 Feature Engineering

In this work, two new features are extracted from the time windows obtained by US-Transportation mode preprocessing algorithm, which are the Skewness and Kurtosis measures. These high-order statistical measures are extracted from each 5 second window of each sensor reading, expecting that they might present different patterns for different transportation modes, therefore improving classification performance of ML models.

Also, PCA is applied over the extracted features of each 5 seconds time window. A simple heuristic strategy was applied in order to select the number of components to be used, which consisted of selecting the first n components where n was equal to the number of sensors that were used. For example, in the smallest sensor setting, 12 features were originally extracted from 3 sensors (*i.e.*, Accelerometer, Gyroscope, Sound). By applying PCA, the first 3 components were extracted of those 12 features, reducing the total number of predictor variables by 9 (*i.e.*, 75%), which reduces computational and time costs associated with model fitting.

During performance evaluation experiments, the classification accuracy and classification cost with and without the use of PCA are compared, in order to evaluate its trade-off for the travel mode detection task.

5.5 Evaluation Experiments

In this section the methods used on the evaluation experiments and the obtained results are described. In order to show the effects of the proposed modifications on the US-TransportationMode travel mode detection technique, 10-fold cross-validation experiments were executed on 12 different scenarios for each of the four ML algorithms used on the original evaluation of the US-TransportationMode technique. Besides, the AutoSklearn was used to generate ensembles for each scenario, reaching the execution of a total of 60 experiments.

Table 5.4 presents the features and sensor sets used on each different scenario and Table 5.5 shows the hyperparameter configurations used for each machine learning algorithm. These sensor sets and configurations were exactly the same used in the US-TransportationMode original evaluation, except for the AutoSklearn configuration.

The AutoSklearn configuration only specifies parameters for the Global Optimization process. Based on those configurations, the algorithm automatically evaluates and selects

Table 5.4: Sensors on each cross-validation experiment and respective features extracted from 5 second time windows.

Scenario	Sensor Set	Features
1	1	Minimum, Maximum, Mean and Standard Deviation of all sensors
2	2	Minimum, Maximum, Mean and Standard Deviation of all sensors
3	3	Minimum, Maximum, Mean, Standard Deviation of all sensors
4	1	3 Principal Components extracted from all sensors Minimum, Maximum, Mean and Standard Deviation
5	2	8 Principal Components extracted from all sensors Minimum, Maximum, Mean and Standard Deviation
6	3	9 Principal Components extracted from all sensors Minimum, Maximum, Mean and Standard Deviation
7	1	Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis of all sensors
8	2	Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis of all sensors
9	3	Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis of all sensors
10	1	3 Principal Components extracted from all sensors Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis
11	2	8 Principal Components extracted from all sensors Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis
12	3	9 Principal Components extracted from all sensors Minimum, Maximum, Mean, Standard Deviation, Skewness and Kurtosis

configurations that will be used to build final ensemble models. Since configuration details are too extensive to be put within this chapter, we made them publicly available at the TMD-AutoML repository⁵. We have shared all source codes used in our experiments, and also the data analysis, within this repository.

These configurations were generated by executing the AutoSklearn Global Optimization on 50% of the dataset samples, which were randomly selected. Then, during the k-fold cross-validation, the ensembles were retrained using only samples of the selected folds, ignoring samples that were used to build the ensemble. This ensures that a fair comparison of the performances between the AutoSklearn generated ensembles and the other ML algorithms configurations is provided during the evaluation.

For performance comparisons, the accuracy, precision, recall and F1-Score metrics were evaluated for each classifier in each scenario. The accuracy definition is provided in Equation 2.14.

All the above metrics - accuracy, precision, recall and F1-Score - are defined and obtained for each transportation mode class. However, as commonly done in several related works, the average of each metric will be obtained for all classes.

In order to evaluate the impact of the applied dimensionality reduction techniques, the

⁵<https://bitbucket.org/eltonfss/tmd-automl>

Table 5.5: Hyperparameter configurations for each machine learning algorithm used for each sensor set.

Learning Algorithm	Sensor Set	Hyperparameter Configuration
Decision Table	All	criterion=gini, splitter=best max_depth=None, min_samples_split=2 min_samples_leaf=1, min_weight_fraction_leaf=0.0 max_features=None, random_state=None max_leaf_nodes=None, min_impurity_decrease=0.0 min_impurity_split=None, class_weight=None presort=False
Random Forest	All	n_estimators=100, criterion=gini max_depth=None, min_samples_split=2 min_samples_leaf=1, min_weight_fraction_leaf=0.0 max_features=auto, max_leaf_nodes=None min_impurity_decrease=0.0, min_impurity_split=None bootstrap=True, oob_score=False, n_jobs=1 random_state=None, verbose=0, warm_start=False class_weight=None
Support Vector Machine	1	C=180, kernel=rbf, degree=3 gamma=auto, coef0=0.0, shrinking=True probability=False, tol=0.001, cache_size=200 class_weight=None, verbose=False, max_iter=-1 decision_function_shape=ovr', random_state=None
Support Vector Machine	2 and 3	C=100
Neural Networks	1	hidden_layer_sizes=900, activation=relu, solver=adam, alpha=0.0001, batch_size=auto learning_rate=constant, learning_rate_init=0.001 power_t=0.5, max_iter=600, shuffle=True random_state=None, tol=-1, verbose=False warm_start=False, momentum=0.9 nesterovs_momentum=True, early_stopping=False validation_fraction=0.1, beta_1=0.9, beta_2=0.999 epsilon=1e-08
Neural Networks	2	hidden_layer_sizes=880
Neural Networks	3	hidden_layer_sizes=600
AutoSklearn	All	time_left_for_this_task=300, per_run_time_limit=360 initial_configurations_via_metalearning=25 ensemble_size=50, ensemble_nbest=50 seed=1, ml_memory_limit=3072 include_estimators=None exclude_estimators=None include_preprocessors=None exclude_preprocessors=None resampling_strategy='holdout' resampling_strategy_arguments=None configuration_mode='SMAC'

Table 5.6: Performance metrics of all machine learning algorithms for scenarios 1, 2 and 3. *DT - Decision Tree, RF - Random Forest, SVM - Support Vector Machine, NN - Neural Networks, ASE - AutoSklearn Ensemble.

Classifier	Scenario	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Fitting (ms)	Scoring (ms)
DT	1	63	63	63	63	82	4
DT	2	65	64	65	64	238	4
DT	3	69	69	69	68	253	4
RF	1	71	70	71	70	1,506	92
RF	2	75	75	75	74	2,368	88
RF	3	79	78	79	78	2,486	84
SVM	1	67	66	67	66	2,443	385
SVM	2	72	71	72	71	1,468	443
SVM	3	77	77	77	76	1,381	439
NN	1	67	66	67	65	95,492	27
NN	2	72	71	72	70	103,909	28
NN	3	77	76	77	75	79,801	22
ASE	1	86	86	86	86	160,940	792
ASE	2	94	94	94	94	277,529	1,565
ASE	3	96	96	96	96	236,905	1,315

average fitting and scoring times on each cross-validation experiment was also evaluated. The fitting time is the time spent for training the classifier and the scoring time is the time spent during classification of unseen samples, which is used to generate the performance metrics.

5.5.1 Obtained results

In this section, the performance metrics obtained during the experiments are presented on shape of tables.

Table 5.6 presents the cross-validation performance of each classifier using the original features of the US-TransportationMode technique, which is related to scenarios 1, 2 and 3 (see Table 5.5). Table 5.7 presents the cross-validation performance of each classifier using features obtained from PCA reduction of the original feature sets of the US-TransportationMode technique, which is related to scenarios 4, 5 and 6. Table 5.8 presents cross-validation performances of each classifier are related to scenarios 7, 8 and 9, which use the skewness and kurtosis features in addition to the original features of the US-TransportationMode technique. Table 5.9 is related to scenarios 10, 11 and 12, which considered features obtained from PCA reduction of the combination of skewness and kurtosis features in addition to the original features of the US-TransportationMode technique.

Table 5.7: Performance metrics of all machine learning algorithms for scenarios 4, 5 and 6.

Classifier	Scenario	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Fitting (ms)	Scoring (ms)
DT	4	52	52	52	51	36	5
DT	5	61	61	61	61	64	3
DT	6	66	66	66	65	73	4
RF	4	60	59	60	58	808	100
RF	5	70	70	70	69	1,231	91
RF	6	74	74	74	73	1,611	88
SVM	4	53	54	53	51	2,740	360
SVM	5	62	61	62	61	1,686	377
SVM	6	69	69	69	68	1,337	308
NN	4	55	54	55	53	78,638	24
NN	5	64	64	64	63	81,214	24
NN	6	71	71	71	70	58,115	17
ASE	4	71	71	71	71	221,589	490
ASE	5	84	84	84	84	179,840	710
ASE	6	89	89	89	89	149,542	601

Table 5.8: Performance metrics of all machine learning algorithms for scenarios 7, 8 and 9.

Classifier	Scenario	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Fitting (ms)	Scoring (ms)
DT	7	65	65	65	64	124	3
DT	8	65	65	65	64	347	3
DT	9	71	71	71	70	357	3
RF	7	73	72	73	72	1,750	83
RF	8	75	74	75	73	2,603	82
RF	9	80	80	80	79	2,719	80
SVM	7	59	60	59	57	3,995	626
SVM	8	66	66	66	64	4,811	1,015
SVM	9	71	72	71	70	5,264	1,076
NN	7	68	67	68	67	86,258	24
NN	8	69	67	69	67	98,461	26
NN	9	74	73	74	72	71,040	18
ASE	7	89	89	89	89	142,765	205
ASE	8	95	95	95	95	217,956	230
ASE	9	97	97	97	97	144,981	100

Table 5.9: Performance metrics of all machine learning algorithms for scenarios 10, 11 and 12.

Classifier	Scenario	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Fitting (ms)	Scoring (ms)
DT	10	47	47	47	47	29	4
DT	11	57	57	57	57	65	4
DT	12	63	62	63	61	80	4
RF	10	53	53	53	52	823	102
RF	11	67	67	67	66	1,276	96
RF	12	72	72	72	71	1,683	89
SVM	10	53	55	53	52	3,239	363
SVM	11	59	59	59	58	1,933	386
SVM	12	66	66	66	65	1,520	318
NN	10	54	55	54	53	80,017	25
NN	11	61	61	61	60	80,680	24
NN	12	69	69	69	68	57,074	17
ASE	10	61	61	61	60	268,661	667
ASE	11	82	82	82	82	146,096	493
ASE	12	87	87	87	87	157,042	620

5.5.2 Discussion

In this section a discussion of the results obtained from the evaluation experiments is provided as well as the main advantages and disadvantages of the proposed modifications on US-TransportationMode technique and whether the research hypothesis of the present work were proven to be true or not, based on collected performance metrics.

5.5.2.1 Classification Performance

As observed on the evaluation experiments, with respect to accuracy, precision, recall and F1-Score, the performance of all traditional classifiers dropped significantly in comparison with the original evaluation of US-TransportationMode technique. One of the main surprises was the Decision Tree classifier, which performed poorly on almost all scenarios, reaching a maximum performance of 71% in accuracy and 70% in F1-Score. Meanwhile, Random Forest performed reasonably well on most scenarios reaching a maximum performance of 80% in accuracy and 79% in F1-Score. Support Vector Machine and Neural Networks did not perform as good as Random Forest, although their maximum accuracy and F1-Score values were very close to Random Forest's ones, with 77% and 76-75%, respectively.

The overall winners were the AutoSklearn Ensembles, whose average performance was above 80% in accuracy, precision, recall and F1-Score. The maximum performance was reached for scenario 9, in which the classifier obtained a 97% score on all metrics using the skewness and kurtosis features proposed in this work, in addition to the minimum, maximum, average and standard deviation features proposed on the US-TransportationMode technique. This performance was only 1% better than in scenario 3, where only the original features were used.

The results show a small but significant improvement with the use of the two additional features (skewness and kurtosis). Another interesting fact about this best performing classifier was the fact that it was the smallest ensemble generated by AutoSklearn, using only a rule based classifier and a Gradient Boosting Machine. Provided that, it can be concluded that *Hypothesis 1* can be considered to be true for the dataset used in this study, while *Hypothesis 3* should be further evaluated through a higher number of experiments and data samples.

5.5.2.2 Classification Cost

With regard to time cost for both training and scoring, AutoSklearn Ensembles were the worst performing classifiers since they required multiple models to be trained and

evaluated on each cross-validation fold. This can pose some restrictions with respect to the usage of these classifiers on real-time applications, depending on the way they are implemented.

The second worst performing classifier with respect to fitting time was the Neural Networks, which was also the second best performing classifier with respect to scoring time, a behaviour that is expected from this kind of machine learning algorithm [133]. Support Vector Machines were the second worst with respect to scoring time, being even worse than AutoSklearn ensembles on some scenarios. This could be explained by the fact that SVMs are essentially two-class classifiers and their adaptation for multi-class problems can become quite complex depending on the number of classes and parameters that are used [2].

Therefore, the overall winners on this evaluation perspective were Decision Trees and Random Forests. If this aspect is to be considered as an important one on ML algorithm selection, Random Forest would be the best pick, since it has also shown good classification accuracy on most scenarios.

5.5.2.3 PCA Impact

As observed on Tables 5.6, 5.7, 5.8 and 5.9, applying PCA significantly reduced classification cost and accuracy on most scenarios. In some applications, this trade-off might be worth since fitting a model with all available features might not be feasible on constrained devices, such as smartphones, for example. However, extracting components from features introduces an additional preprocessing cost, which was measured through the fold splitting time on the experiments of this work.

Table 5.10 shows the average splitting, fitting and scoring time on scenarios with (4,5,6,10,11,12) and without PCA (1,2,3,7,8,9). It can be observed that applying PCA did not significantly increase the average splitting time, neither reduced the average fitting and scoring time. This might be a symptom that additional studies are required to better apply this dimensionality reduction technique, which demands a deeper understanding of the mathematical properties of its algorithm and of the data being analyzed. Therefore, although it was not possible to state that *Hypothesis 2* was true, it still not possible to assure that it is false.

Table 5.10: Mean split fit and score time for each scenario.

Scenario	Mean Split Time (ms)	Mean Fit Time (ms)	Mean Score Time (ms)
1, 2, 3	253	64,453	353
4, 5, 6	255	51,902	213
7, 8, 9	322	52,229	238
10, 11, 12	352	53,348	214

5.6 Conclusion

This chapter proposed and evaluated the use of AutoML and feature engineering techniques to enhance travel mode detection algorithms. The evaluation experiments showed that these methods can greatly improve classification accuracy, reaching a maximum of 97% accuracy, precision, recall and F1-Score on the TMD Dataset using the AutoSklearn Global Optimization framework. This is the first work to present evidence that AutoML frameworks can outperform human researchers in the combined algorithm selection and hyperparameter optimization of ML classifiers for the travel mode detection problem.

6. Real-Time Travel Mode Detection with Recurrent Neural Networks

6.1 Introduction

Identifying the travel modes (*i.e.*, modes of transportation) used by citizens on their daily commute is useful for Intelligent Transportation Systems (ITS), since they can use this information to better adapt transportation infrastructure in peak demand periods for specific modes. An ubiquitous sensing device that has been explored in recent works to allow this type of information extraction is the smartphone, which is now capable of collecting multiple sensory data and executing sophisticated machine learning (ML) inference [13, 126].

Many works have explored the use traditional ML algorithms and ensemble methods, combined with general and domain specific feature extraction techniques. However, most traditional ML algorithms require hard assumptions about the form of the function that maps the input features to the detected travel modes [60].

Meanwhile, deep learning based techniques can automatically learn abstract representations of their input data enabling the development of more general travel mode detection models. Since most of them are based on the Neural Networks (NN) model, they are capable of approximating any Borel measurable function [49] and the only assumption required is that the pattern that we are trying to learn is composed by multiple abstract patterns, which can be considered to be true for most ML problems [60].

Several travel mode detection techniques, that used deep learning algorithms, have been proposed during the last decade. Most of them used classical deep learning architectures such as Convolutional Neural Networks (CNN) [34, 83], Deep Neural Networks (DNNs) [42, 89, 98, 140] and Recurrent Neural Networks (RNNs) [70, 84, 128, 138]. However, none of them explored the use of RNNs in conjunction with time and frequency

domain feature extraction to enable the development of flexible travel mode detection solutions based on multiple smartphone sensor readings.

Also, few of them proposed online detection mechanisms and none have provided a general solution that could be used by developers and researchers in the development and deployment of context-aware applications. Thus, in this chapter generic framework for travel mode detection is proposed, leveraging Long-Short Term Memory (LSTM) cells for building flexible online travel mode detection models. Therefore, the contributions of this work are:

- TMDFramework, a generic framework that formalizes offline training and offline/online travel mode inference.
- TMD-LSTM, an online travel mode detection method that allows detecting modes of transportation in real-time using sensor features time series and LSTM based RNNs.
- An architecture for the implementation of cloud-based and in-device detection approaches, using TMD-LSTM.
- Performance evaluation of TMD-LSTM using a public dataset of smartphone sensor data acquired during trips in the city of Bologna, Italy.
- Analysis of the detection delay (*i.e.*, time window size) influence on the classification accuracy and model size.

6.2 Related Works

In this section, we review the main related works that applied deep learning techniques for travel mode detection. Most of them using smartphone collected data, with a few exceptions that were still worth reviewing given their contribution to the state-of-the-art. The earliest works were published in 2016 and the latest in 2018, which might indicate that this research branch still in its early development. Table 6.1 summarizes the related works presented in this section.

In [38] authors proposed technique that uses a fully-connected Deep Neural Network (DNN) with Stacked Denoising Autoencoder (SDA) to extract high-level features from trajectory images generated from GPS logs. This works remains to be the only one in which authors tried to develop a travel mode detection technique that utilized images as input for the deep learning model.

Table 6.1: Summary of related works. *GPS - Global Positioning System, CDR - Call Data Records **DL Model - Deep Learning Model, CNN - Convolutional Neural Networks, DNN - Deep Neural Networks, LSTM - Long-Short Term Memory, IO-HMM - Input Output Hidden Markov Models, RNN - Recurrent Neural Networks, SAE - Stacked Autoencoders, CGRNN - Control Gate based Recurrent Neural Network

Ref.	Data	DL Model	Dataset	Detection
[34]	GPS Logs	CNN	Microsoft GeoLife	Offline
[140]	GPS Logs	CNN + DNN	Microsoft GeoLife	Offline
[89]	GPS Logs	DNN	Microsoft GeoLife + OpenStreetMap	Offline
[84]	CDR Logs	IO-HMM + LSTM	US Mobile Carrier	Offline
[98]	Trip Data	DNN	Preference data for Swiss Metro	Offline
[83]	Accelerometer	CNN	Research dataset	Online
[70]	GPS Logs	RNN	Research dataset	Offline
[42]	Accelerometer Magnetometer Gyroscope	DNN	HTC dataset	Online
[152]	GPS Logs	SAE + DNN	Microsoft GeoLife	Offline
[138]	Accelerometer	CGRNN	HTC dataset	Online
[128]	GPS Logs	LSTM	Research dataset	Offline
[38]	Trajectory Images	DNN	Microsoft GeoLife + Research dataset	Online

In [128] authors developed DeepTransport, a system system that uses a multi-task deep Long Short-Term Memory (LSTM) architecture to learn and infer mobility and travel mode choices patterns at macro scale using GPS records and transportation network data as input. They implemented an end-to-end system that managed large-scale data collection, pre-processing, learning, visualization and evaluation providing a useful tool for urban planners that wish to predict and improve mobility at a citywide level.

In [138] authors proposed a technique that uses a Control Gate based Recurrent Neural Network (CGRNN) to learn travel mode patterns from accelerometer data obtained through the HTC transportation mode dataset [150]. The use of accelerometer data as input to the deep learning model has also been explored in a latter work [83] in which the authors proposed a technique that uses a deep learning architecture based on the Convolution Neural Network (CNN) model to detect travel modes based on accelerometer magnitude.

In [152] authors proposed a technique that uses a DNN with Stacked Autoencoder (SA) to extract higher-level features from human crafted features obtained from GPS data,

such as travel distance, average speed, average acceleration, head direction change and us/subway stop closeness. A similar approach has been explored in [42], but instead of GPS authors used a combination of accelerometer, magnetometer and gyroscope features as input to a feed-forward DNN to perform online detection in smartphones. The data from this study was also obtained through the HTC transportation mode dataset [150].

In [70] authors presented a technique that extracts point-and-segment-based features from GPS records and performs discretization and basis expansion to feed them into a Maxout GRU RNN. They import the concept of embedding from natural language processing in order convert continuous features into vector representations that can better capture the general characteristics of the input data.

In [98] authors presented a technique for predicting travel mode choices based on historical trip preference data from the Swiss Metro (SM) using DNN with a Function for Availability of Alternatives (FAA). Although this work did not present a deep learning technique for travel mode detection based on smartphones, the proposed DNN architecture and FAA could be explored in future work of this area.

Another related work, that was reviewed for similar reasons was [84], in which the authors propose a framework that builds up on cell phone data processing and activity based inferences of travel purposes with an Input-Output Hidden Markov Model (IO-HMM), followed by a LSTM network to learn travelers mobility sequences. Although this study was conducted using CDR logs, which are not easily accessible, and data was synthetically labeled, through a probabilistic method, this work produced innovative tooling for travel demand exploration and traffic simulation.

In [89] authors performed hypothesis testing of several feature extraction techniques and comparison of different ML approaches including Autoencoders for travel mode detection using GPS traces. The use of Autoencoders has also been explored in [140] where the authors evaluate a technique that uses a Sparse Autoencoder (SA) to extract point-level deep features (PLDF) from point-level handcrafted features (PLHF) and a CNN to aggregate the PLDF and generate trajectory-level deep features (TLDF). As a result, they propose a DNN architecture to detect the transportation mode using trajectory-level handcrafted features (TLHF) and the TLDF.

The latest deep learning architecture for travel mode detection using smartphone data was proposed in [34]. The authors develop a technique that uses a CNN to learn travel mode patterns from speed, acceleration/deceleration, jerk, and bearing rate extracted from GPS trajectories obtained through the Microsoft Geolife dataset [151] which has also been

used in [140], [89], [152] and [38].

After reviewing the related works described in this section, we conclude that the concept of representational learning has been deeply explored, with several deep learning architectures being proposed using DNN, CNN, RNN and Autoencoders as building blocks. Most of the proposed models achieved good performance results in the evaluation experiments, although some works of them applied simplistic evaluation metrics [83, 138, 140] and a reduced number of travel mode classes [42, 98, 138].

Some common characteristics that can be noted are the use of GPS data as input [34, 38, 70, 89, 128, 140, 152] with the Microsoft GeoLife dataset [151] as a benchmark [34, 38, 89, 140, 152].

Also, few works focused on online detection, and none of them on online learning, since most of the were focused on transportation planning, which does not require fast identification of the travel mode being used as would be needed in real-time context-aware applications such as Location Based Services (LBS) [106]. To that end, the only works that presented an online detection solution [38, 42, 83, 138] did not present a working prototype, neither a publicly accessible API (Application Programming Interface) or library that could be used for real smartphone applications or future studies.

Some other aspects that were not explored in the existing literature, were the usage of a larger set of smartphone sensors as input for the deep learning model, with a maximum of three sensors being used [42]. This kind of architecture could allow the development of a model that is resilient to sensor failure and malfunctioning. Also, the use of transfer learning to accelerate personalized model training has not been explored, what could lead to better performance in individualized travel mode detection.

6.3 Proposed Method

In this section we describe the proposed framework and method in details.

6.3.1 Generic Framework for Travel Mode Detection

TMDFramework consists of an open-source generic framework that will allow developers and researchers to build travel mode detection models using any combination of sensors features, including raw features, as well as any number of travel modes. Formally, if we consider $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \dots, \mathbf{s}_N\}$ the set of N input feature vectors and $\mathbf{t} =$

$\{t_1, t_2, t_3, \dots, t_M\}$ the set of M travel modes assigned to each feature vector in S , TMD-Framework will generate a detection model using a supervised machine learning algorithm, for any number of N and M .

The generated model can be used for both, online and offline travel mode detection. Let F' be a set of new input feature vectors, the detection model will produce the output vector t' which will contain the inferred travel modes for each input feature vector in S' . In the case of online detection, S' will contain only one input feature vector, which corresponds to the most recent sensor readings, and, therefore, the detection model will produce the single valued vector t' , which corresponds to the currently inferred travel mode. Figure 6.1 illustrates the proposed model in offline training and online/offline detection procedures based on these terms.

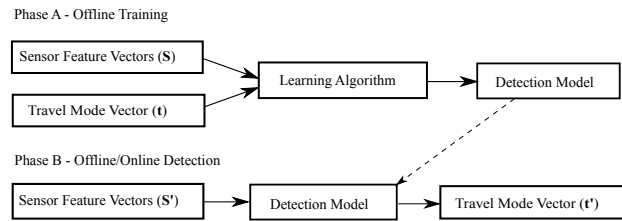


Figure 6.1: Proposed offline training and offline/online detection mechanism.

Provided that, TMDFramework will allow the generation of models based on features derived from GPS readings, as well as raw data and features obtained from accelerometer, gyroscope, magnetometer and any other smartphone sensor. The set of travel modes that can be detected should depend only on the availability of labeled training data, being the learning algorithm L flexible enough to deal with any set of modes that are fed during the training procedure.

Most importantly, TMDFramework should provide an API for mobile platforms, such as Android and iOS, enabling its users to load previously trained models for real-time inference or to train new ones based on newly collected data.

6.3.2 Online Travel Mode Detection with LSTM

TMD-LSTM consists of a travel mode detection method developed within the TMD-Framework architecture that uses a deep RNN model based on Standard LSTM cells to learn a travel mode detection model on any set of input features. LSTM is one of the most popular and efficient methods for reducing the effects of vanishing and exploding gradients when using recurrent connections within neural networks [115].

The LSTM approach changes the structure of hidden units from "sigmoid" or "tanh" to

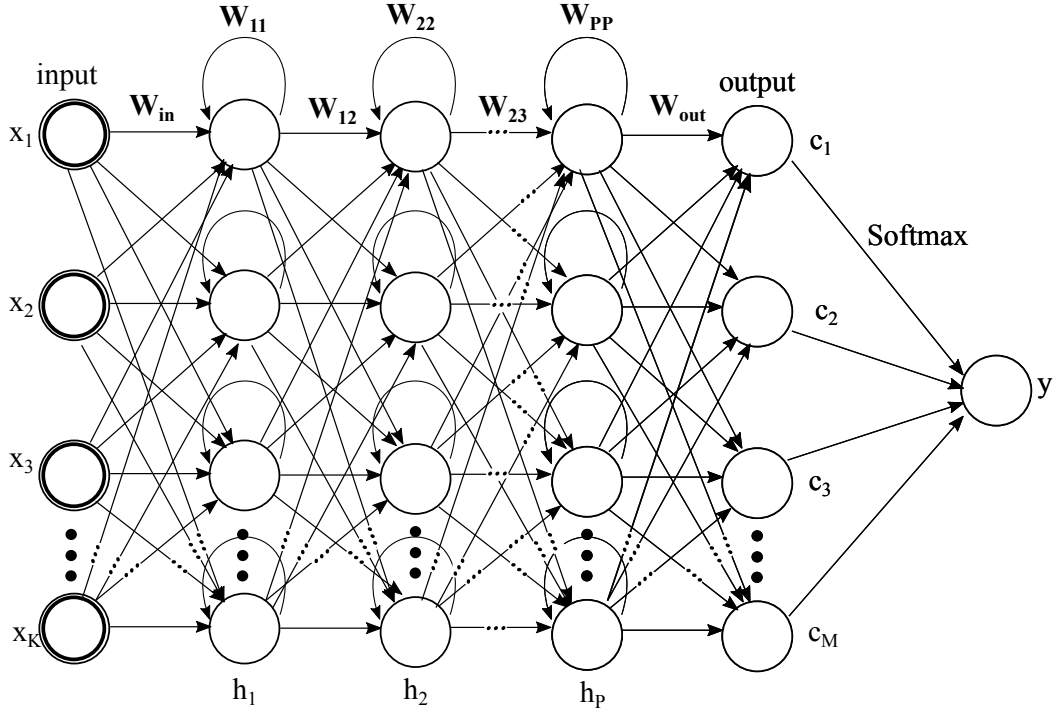


Figure 6.2: TMD-LSTM model architecture.

memory cells, in which their inputs and output are controlled by gates. These gates control flow of information to hidden neurons and preserve extracted features from previous timesteps [65, 80].

The motivation for using a deep RNN model is to be able to extract higher level representations of the input features, while learning sequential dependencies between the input samples. The first can be obtained through a deep architecture, while the second is provided by the recurrent connections within the hidden layers and its training procedure.

Figure 6.2 presents a simplified version of the TMD-LSTM model architecture, where K is the number of input features, M is the number of detected modes and P is the number of hidden layers. The input features are represented as $(x_1, x_2, x_3, \dots, x_K)$ and the weight matrix of the connections between the input layer and the first hidden layer, h_1 , is represented as \mathbf{W}_{in} . The weight matrices of the connections between $h_1, h_2, h_3, \dots, h_P$ are represented as $\mathbf{W}_{12}, \mathbf{W}_{23}, \dots, \mathbf{W}_{(P-1)P}$ and the weight matrices of the recurrent connections of each hidden layer are represented as $\overline{\mathbf{W}}_{11}, \overline{\mathbf{W}}_{22}, \dots, \overline{\mathbf{W}}_{PP}$. The weight matrix of the connections between the last hidden layer and the output layer, h_P , is represented as \mathbf{W}_{out} . The logits of the network are represented as $(c_1, c_2, c_3, \dots, c_M)$. These are used as input for a softmax function that generated the travel mode prediction y . The hidden biases and the internal structure of the LSTMs cells are not represented for simplicity.

Formally, the input gate of an LSTM cell is defined as $g_t^i = \sigma(\mathbf{W}_{Ig^i}x_t + \mathbf{H}g^i h_{t-1} +$

$\mathbf{W}_{g^c g^i} g_{t-1}^c + \mathbf{b}_{g^i}$), where $\mathbf{W}_{I g^i}$ is the weight matrix from the input layer to the input gate, $\mathbf{W}_{H g^i}$ is the weight matrix from hidden state to the input gate, $\mathbf{W}_{g^c g^i}$ is the weight matrix from cell activation to the input gate, and \mathbf{b}_{g^i} is the bias of the input gate [115]. The forget gate is defined as $g_t^f = \sigma(\mathbf{W}_{I g^f} x_t + \mathbf{W}_{H g^f} h_{t-1} + \mathbf{W}_{g^c g^f} g_{t-1}^c + \mathbf{b}_{g^f})$, where $\mathbf{W}_{I g^f}$ is the weight matrix from the input layer to the forget gate, $\mathbf{W}_{H g^f}$ is the weight matrix from the hidden state to the forget gate, $\mathbf{W}_{g^c g^f}$ is weight matrix from the cell activation to the forget gate, and \mathbf{b}_{g^f} is the bias of the forget gate [115]. The cell gate is defined as $g_t^c = g_t^i \tanh(\mathbf{W}_{I g^c} x_t + \mathbf{W}_{H g^c} h_{t-1} + \mathbf{b}_{g^c}) + g_t^f g_{t-1}^c$, where $\mathbf{W}_{I g^c}$ is the weight matrix from the input layer to the cell gate, $\mathbf{W}_{H g^c}$ is the weight matrix from the hidden state to the cell gate, and \mathbf{b}_{g^c} is the bias of the cell gate [115]. The output gate is defined as $g_t^o = \sigma(\mathbf{W}_{I g^o} x_t + \mathbf{W}_{H g^o} h_{t-1} + \mathbf{W}_{g^c g^o} g_t^c + \mathbf{b}_{g^o})$, where $\mathbf{W}_{I g^o}$ is the weight matrix from the input layer to the output gate, $\mathbf{W}_{H g^o}$ is the weight matrix from hidden state to the output gate, $\mathbf{W}_{g^c g^o}$ is the weight matrix from cell activation to the output gate, and \mathbf{b}_{g^o} is the bias of the output gate [115]. Finally, the hidden state is computed as $h_t = g_t^o \tanh(g_t^c)$.

6.3.3 Implementation Design

In this section we describe the implementation design of a real-time travel mode detection solution based on TMDFramework and TMD-LSTM. The solution described considers that the input features that are fed to the TMD-LSTM model are extracted from time windows of sensor readings. In Figure 6.3 we present an overview of the real-time travel mode inference using this architecture.

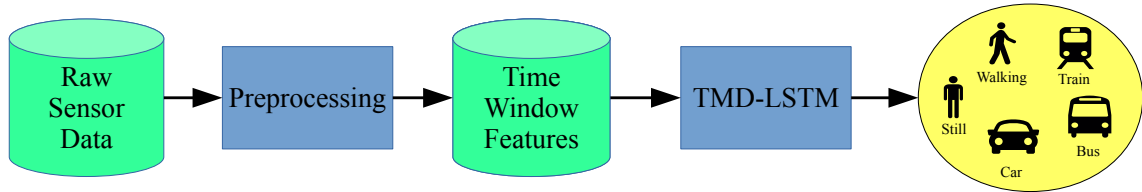


Figure 6.3: Travel Mode Inference using a trained TMD-LSTM.

The training and inference using the TMD-LSTM model can be done exclusively via cloud infrastructure, were the sensory data collected by the user smartphone is sent to a cloud server that identifies the corresponding mode of transportation and retrieves it to the mobile client. This approach will likely consume less processing power from smartphones, although it will require internet connectivity and a considerable amount of data uploading, which can reduce the smartphone battery autonomy and, therefore, the continuous sensing period.

An alternative to this approach is to use the cloud servers for model training only, and

deploying those models in the users' smartphones for in-device travel mode inference. This approach is likely to consume more processing and storage resources than the cloud-based approach, but it will not rely on internet connectivity for inference, neither require continuous uploading of the sensory data for obtaining travel mode inferences. Nonetheless, the sensory data can be uploaded to the server periodically to allow the customization of the travel mode detection models of each user. The training and deployment of these new models can be scheduled in a way that reduces the impact of the data uploading in the smartphone battery autonomy and data usage. One possible strategy is to store the collected samples in the smartphone persistent memory during daily commute and upload the samples to the server for model update only when the smartphone is connected to a charger and has WiFi [102] connectivity. The server should be able to train and deploy the updated model while the smartphone is charging as well.

The cloud-based and in-device approaches illustrated in Figure 6.4.

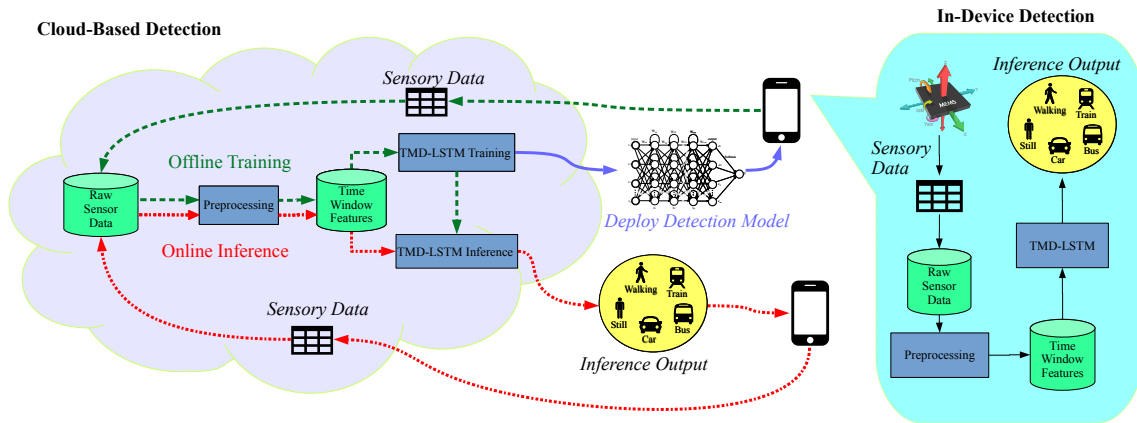


Figure 6.4: Cloud-based and In-device real-time travel mode detection with TMD-LSTM.

Both approaches could be made available using the same cloud infrastructure and mobile API. Data collected through Cloud-based or In-device detection could be integrated for model training and context-aware applications could select the optimal periods for leveraging each approach, as for example, activating in-device detection when internet connectivity is dropping.

6.4 Experiments

In this section we evaluate the online detection feature of our framework through cross-validation experiments using a public benchmark dataset [26].

6.4.1 Settings

Experiments were conducted using the TMDataset, which contains mobility data collected by 16 volunteers in Bologna, Italy. It contains 226 labeled files corresponding to 31 hours of data: 26% Walking, 25% Car, 24% Still, 20% Train, 5% Bus. A total of 23 physical and virtual sensors were sampled with a maximum frequency of 20 Hz.

In Figure 6.5 the number of samples per mode and user, which are not fully balanced, are presented. Therefore, all data from user one (U1) is discarded, to reduce bias, and SMOTE is applied on the training set of each cross-validation fold to balance the number of samples for each mode during training. From the initial 23 sensors sampled, 6 were discarded due to poor data quality and/or lack of samples. Therefore a total of 16 physical and virtual sensors were left for the experiment: Accelerometer, Linear Acceleration, Gravity, Orientation, Gyroscope Uncalibrated, Gyroscope, Game Rotation Vector, Rotation Vector, Proximity, Sound, Light, Pressure, Speed, Step Counter, Magnetic Field Uncalibrated, Magnetic Field. The magnitude metric [135] was extracted from all three axis sensors of this list, in order to make the learning and inference independent of the orientation of the device during data collection.

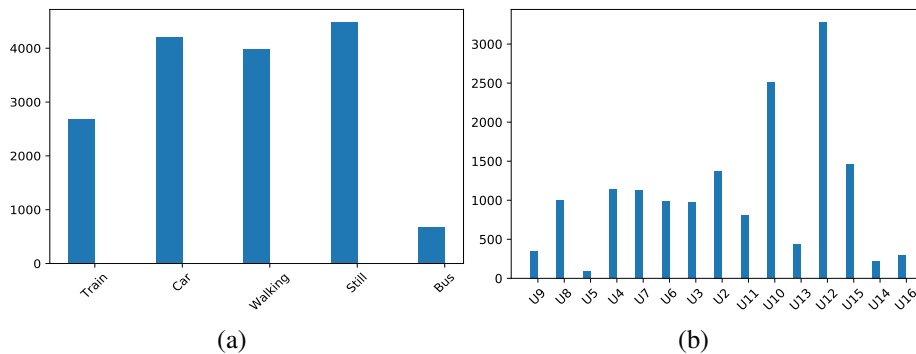


Figure 6.5: Sample distributions per travel mode (a) and user (b).

After segmenting the sensor readings in time series of 5 seconds, 20 features were extracted from the series of each sensor, 13 from the time domain: min, max, mean, standard deviation, median, skewness, kurtosis, range, variance, entropy, 1st-quartile, 3rd-quartile and interquartile range; 7 from the frequency domain, obtained through the Fast Fourier Transform [22] of each series: highest magnitude and its frequency, total spectrum and phase, spectral density, entropy and centroid. A total of 320 features was extracted from each window with forward filling for missing features.

The features extracted were scaled based on the interquartile range and the best features were selected using Recursive Feature Elimination (RFE) with Random Forest (RF) [91]

and K-Best [104] feature selection based on Mutual Information for Classification [33]. Table 6.2 lists a total of 126 features selected, being 87 from the time domain and 39 from the frequency domain.

The features extracted are used in 10-fold cross-validation experiments with 12 different ML algorithms: AutoML, Naive Bayes (NB), AdaBoost (AB), Support Vector Machine (SVM), Decision Tree (DT), RF, K-Nearest Neighbours (KNN), Feedforward NN (FNN), Deep FNN (DFNN), RNN and Logistic Regression (LR). The RNN model corresponds to the TMD-LSTM method proposed in Section 6.3.2. The FNN and DFNN models correspond to networks with up to 1 hidden layer, and networks with at least one two hidden layers, respectively. The term (D)FNN will be used to reference both models from now on. Figure 6.6 presents an overview of the evaluation experiments process.

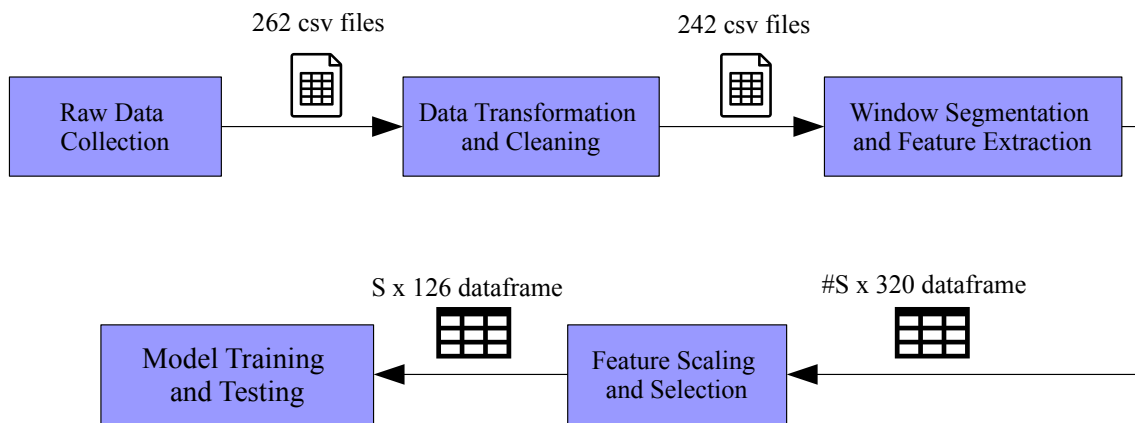


Figure 6.6: Evaluation experiments process overview. Window size = 1 second, #S = 80566; Window size = 5 seconds, #S = 16041; Window size = 10s, #S = 7967.

After running the experiments with 5 second time windows, these were reproduced using time windows of 1 and 10 seconds as well, in order to evaluate the impact of different window sizes on performance.

6.4.2 Details of Training and Implementation

The (D)FNN and RNN models were implemented using Tensorflow [1] Python library and trained using Adaptive Gradient Algorithm (AdaGrad) [36] over batches of 128 samples in 1000 iterations with exponential decay learning rate initialized to 0.1, 0.01 and 0.001 with decay rate of 96%. For regularization, L2 norm is applied with $\beta = 0.01$. The dataset is not shuffled in the to allow the RNNs to identify sequential patterns within the samples. The RNN architecture implemented used Stateful LSMT Cells [30], that used the number of samples in each training batch as the maximum number of timesteps, t , preserved in the cell's memory.

Table 6.2: Best features selected from the multiple sensors.

Sensor	Time Domain Features	Frequency Domain Features (FFT)
Gyroscope	Range, Max, Entropy, Mean, Min, Median, Variance, Interquartile Range, Standard Deviation, 1st-Quartile, 3rd-Quartile	Total Spectrum, Highest Magnitude, Spectral Density, Spectral Entropy
Gyroscope Uncalibrated	1st-Quartile, Median, 3rd-Quartile, Entropy, Variance, Max, Standard Deviation, Mean, Min, Interquartile Range, Range	Spectral Density, Total Spectrum, Highest Magnitude Spectral Entropy
Magnetic Field	1st-Quartile, 3rd-Quartile, Min, Max, Mean, Median, Entropy	Highest Magnitude, Total Spectrum, Spectral Entropy
Magnetic Field Uncalibrated	1st-Quartile, Median, 3rd-Quartile, Max, Mean, Min, Entropy	Highest Magnitude, Spectral Density, Total Spectrum, Spectral Entropy
Pressure	1st-Quartile, 3rd-Quartile Mean, Min, Median, Max	Total Spectrum, Spectral Density, Highest Magnitude
Accelerometer	Min, Mean, Standard Deviation, 1st-Quartile, 3rd-Quartile, Variance, Range, Max, Median, Entropy, Interquartile Range	Spectral Centroid, Highest Magnitude, Spectral Density, Spectral Entropy, Total Spectrum
Linear Acceleration	Variance, Entropy, 1st-Quartile, Median, 3rd-Quartile, Interquartile Range, Max, Min, Mean, Standard Deviation	Spectral Density, Total Spectrum, Spectral Entropy, Highest Magnitude
Orientation	1st-Quartile, Median, 3rd-Quartile, Min, Mean, Max, Interquartile Range, Range, Variance, Entropy, Standard Deviation	Total Spectrum, Highest Magnitude, Spectral Entropy
Gravity	Entropy, Mean, 1st-Quartile, Median, 3rd-Quartile	Spectral Density, Highest Magnitude, Total Spectrum, Spectral Entropy
Rotation Vector	Entropy, Max, Mean, Min, 1st-Quartile, Median, 3rd-Quartile	Highest Magnitude, Spectral Entropy
Game Rotation Vector	Entropy	Total Spectrum, Highest Magnitude, Spectral Entropy

For the (D)FNN models, multiple configurations with ReLu and Leaky ReLu activation are evaluated. For the RNN models, configurations with and without dropout are tested as well. The number of hidden layers in each configuration varied from 1 to 4 hidden layers and the number of units in each layer was derived from the number of input features. The 1st hidden layer had 126 units and the subsequent layers had half the number of units of its preceding layers, rounded down.

6.4.3 Results Analysis

In this section the analysis of the experiment results is presented in four steps. First, the experiment results with 5 second time windows are presented. After that, the results with 1 and 10 seconds are analyzed. Lastly, an overall analysis is made considering the

results with the three window sizes.

6.4.3.1 Five Second Time Windows

In Table 6.3 it can be observed that the only models that reached an average accuracy equal or higher than 90% when using the 5 second time windows were the ones generated by AutoML, DT, RF and RNNs. From these, only AutoML, RF and RNNs obtained an average precision, recall and f1-score equal or higher than 90% and only AutoML and RF obtained a kappa coefficient equal or higher than 90%.

Table 6.3: Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 5 second time windows.

ML Alg.	A(%)	P(%)	R(%)	F1(%)	Kappa(%)	Size
AutoML	96	95	95	95	95	117.9 MB
NB	54	50	52	47	41	10.5 KB
AB	76	70	75	71	69	69.5 KB
SVM	74	75	69	74	65	13.8 MB
DT	91	87	88	87	88	159.9 KB
RF	95	93	93	93	94	3.6 MB
KNN	87	81	86	83	83	41.2 MB
FNN	84	84	84	84	80	100.2 KB
DFNN	83	83	82	83	78	114.0 KB
LR	26	26	28	23	09	5.8 KB
RNN	90	90	90	90	87	314.5 KB

With respect to the model size, the lowest average size reported was from the LR models, but these were also the worst performing models with respect to classification accuracy. Provided that, the focus of this analysis will be on the comparison of the average model size of the models with highest average accuracy: AutoML, RF, DT and RNN.

The average model size for the AutoML models was above 100MB, which might represent a huge chunk of the RAM memory of a low end smartphone. When considering the RF models, the average size drops to 3.6MB which still considerably large, but much more suitable for smartphones.

Nonetheless, the RNN and DT models present an even better trade-off between classification accuracy and model size, reaching at least 90% accuracy with average model size below 1MB. In conjunction with the computational cost of the ML model, the model size can be a critical aspect for the in-device processing of the travel mode detection algorithm. In that sense, RNNs present average 90% accuracy, precision, recall and F1-Score with average model size below 400KB.

With regards to the computational cost, LSTM based RNN time complexity is $O(W)$, where W is the set of weights of the network, while DT is $O(N)$, where N is the number of samples used in training, and RF is $O(MN)$, where M is the number of trees in the

forest [65]. Since the RNN complexity is determined by a constant W , its computational cost does not increase with the number of samples used for training and therefore is more suitable for in-device processing than DT and RF [87].

Figure 6.7 illustrates the impact of different numbers of layers and initial learning rate in RNNs accuracy and model size. Also, the variation of accuracy and model size when using dropout of 0.5 or no dropout can be observed. In general, increasing the number of layers above 3 hidden layers did not lead to better accuracy and the model size increased linearly with respect to the number of layers. The use of dropout of 0.5 caused a small increase in model size and did not provide better accuracy in most configurations. The use of an initial learning rate below 0.01 did not lead to better results in most configurations as well.

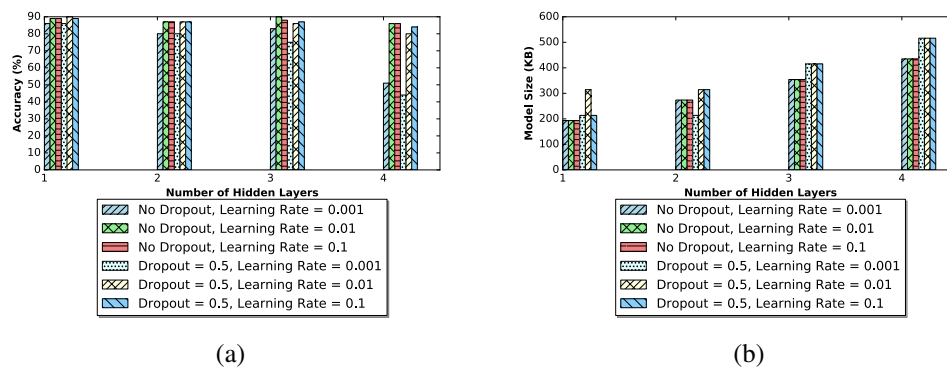


Figure 6.7: RNNs accuracy (a) and size (b) obtained for multiple hyperparameter configurations, using 5 second time windows.

Figure 6.14 presents the average train and test confusion matrices of the best performing RNN configuration, with 3 hidden layers, no dropout and initial learning rate of 0.01. The models built with this configuration were able to learn the patterns of the five travel modes present in the evaluation dataset very well, as can be observed on the training set confusion matrices. However, when analyzing the test matrices, it can be concluded that the Still and Walking patterns presented better generalization than the ones of Car, Train and Bus modes, although the true positive rate for these classes was above 80%. The generalization performance refers to the accuracy of the NN in classification of unseen data, which is represented by the test set [116].

In addition to the previous results, Figure 6.8 illustrates the impact of different numbers of layers and initial learning rate in (D)FNNs accuracy and model size. Also, the variation of accuracy and model size when using ReLu or Leaky-ReLu as hidden layer activation function can be observed.

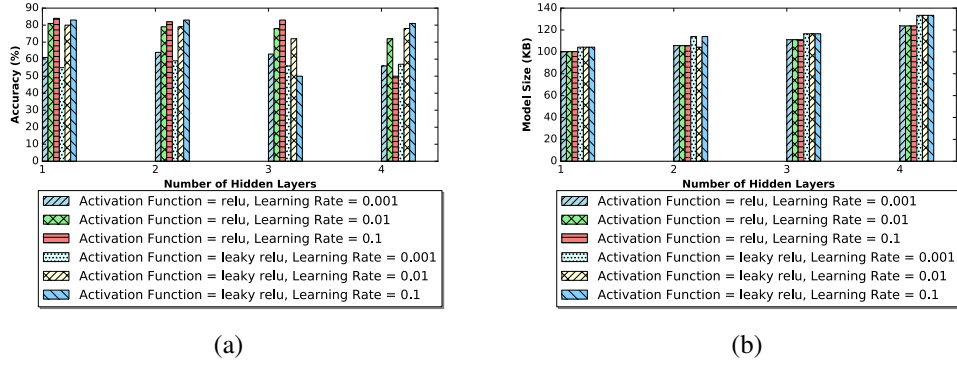


Figure 6.8: (D)FNNs accuracy (a) and size (b) obtained for multiple hyperparameter configurations, using 5 second time windows.

For FNNs, the best performance was reached when using 1 hidden layer, ReLu hidden activation and initial learning rate of 0.1, while for DFNNs the best performance was reached with 2 hidden layer, Leaky ReLu hidden activation and initial learning rate of 0.1. The average training and testing confusion matrices of the best performing FNNs and DFNNs configurations are presented in Figure 6.14 also.

6.4.3.2 One Second Time Windows

In Table 6.4 we observe that the only models that reached an average accuracy equal or higher than 90% when using the 1 second time windows were the ones generated by AutoML and RF. From these, none have obtained an average precision, recall and f1-score equal or higher than 90% and only AutoML obtained a kappa coefficient equal to 90%.

Table 6.4: Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 1 second time windows.

ML Alg.	A(%)	P(%)	R(%)	F1(%)	Kappa(%)	Size
AutoML	92	89	90	89	90	713.9 MB
NB	54	57	46	47	38	10.4 KB
AB	73	67	72	67	65	69.6 KB
SVM	34	44	33	29	16	15.8 MB
DT	87	82	84	83	82	1.1 MB
RF	91	88	89	89	88	26.0 MB
KNN	85	79	83	81	81	204.8 MB
FNN	67	68	67	67	59	100 KB
DFNN	70	71	70	70	63	105.7 KB
LR	34	34	36	29	18	5.8 KB
RNN	77	78	77	77	71	273.8 KB

With respect to the model size, the lowest average size reported was, again, from the LR models, but these were also the worst performing models with respect to classification accuracy, again. Provided that, the focus of this analysis will be on the comparison of the average model size of the models with highest average accuracy: AutoML and RF.

The average model size for the best AutoML models was above 700MB, which might represent an even greater chunk of the RAM memory of a low end smartphone, in comparison to the 100MB of the models generated with 5 second time windows. When considering the RF models, the average size drops to 26MB which is more than seven times larger than the models generated for 5 second time windows.

Provided that, it could be observed that the DT models present an better trade-off between classification accuracy and model size, reaching 87% accuracy with average model size of 1.1MB. In this configuration, RNNs have presented a modest performance, with an average accuracy of 77% and model size bellow 400KB.

Figure 6.9 illustrates the impact of different numbers of layers and initial learning rate in RNNs accuracy. In general, increasing the number of layers or using dropout of 0.5 did not lead to better accuracy. The model sizes varied in the same way as the experiments with 5 second time windows as they cannot be influenced by this variable.

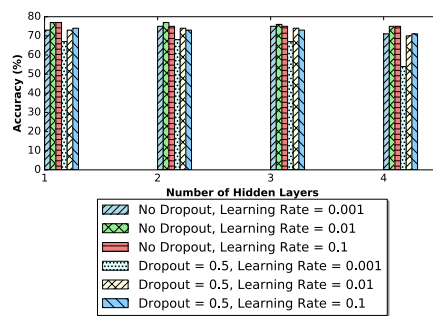


Figure 6.9: RNNs accuracy obtained for multiple hyperparameter configurations, using 1 second time windows.

Figure 6.14 presents the average train and test confusion matrices of the best performing RNN configuration, respectively, with 2 hidden layers, no dropout and initial learning rate of 0.01. The models built with this configuration were able to learn the patterns of the five travel modes present in the evaluation dataset, as can be observed on the training set confusion matrices. However, when analyzing the test matrices, it can be concluded that the Bus and Walking patterns presented better generalization than the ones of Still, Car and Train modes, although the true positive rate for these classes was above 70%.

In addition to the previous results, Figures 6.10 illustrate the impact of different numbers of layers and initial learning rate in (D)FNNs accuracy and model size, respectively. Also, the variation of accuracy and model size when using ReLu or Leaky-ReLu as hidden layer activation function can be observed.

For FNNs, the best performance was reached when using 1 hidden layer, ReLu hidden activation and initial learning rate of 0.1, while for DFNNs the best performance was

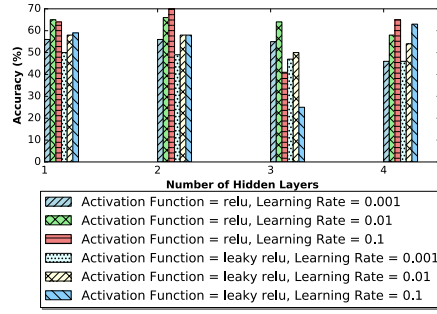


Figure 6.10: (D)FNNs accuracy obtained for multiple hyperparameter configurations, using 1 second time windows.

reached with 2 hidden layer, ReLu hidden activation and initial learning rate of 0.1. The average training and testing confusion matrices of the best performing FNNs and DFNNs configurations are presented in Figure 6.14 also.

6.4.3.3 Ten Second Time Windows

In Table 6.5 we observe that the only models that reached an average accuracy equal or higher than 90% when using the 10 second time windows were the ones generated by AutoML, DT, RF and KNN. From these, only AutoML, RF and KNN obtained an average precision, recall, f1-score and kappa coefficient equal or higher than 90% .

Table 6.5: Average accuracy (A), precision (P), recall (R), F1-score (F1) , kappa coefficient (Kappa) and model size (Size) per-fold obtained by the best configurations of each ML algorithm tested using 10 second time windows.

ML Alg.	A(%)	P(%)	R(%)	F1(%)	Kappa(%)	Size
AutoML	96	95	95	95	95	62.5 MB
NB	47	39	51	40	36	10.5 KB
AB	78	73	77	73	71	69.6 KB
SVM	88	85	84	84	84	4.6 MB
DT	91	87	88	88	88	131.1 KB
RF	96	94	94	94	94	1.8 MB
KNN	94	90	93	91	92	20.5 MB
FNN	82	82	82	82	77	114.0 KB
DFNN	82	82	82	82	77	104.3 KB
LR	84	79	83	80	80	5.8 KB
RNN	89	89	89	89	86	273.8 KB

With respect to the model size, the lowest average size reported was from the LR models again, but , surprisingly, these were not the worst performing models with respect to classification accuracy in this configuration, reaching an average of 84%. Nonetheless, the focus of this analysis will be on the comparison of the average model size of the models with highest average accuracy: AutoML, RF, DT and KNN.

The average model size for the AutoML models was around 60MB, which might represent a better trade-off when compared to AutoML generated models for 1 and 5 second time windows. When considering the KNN models and RF models, the average

size drops to 20.5MB and 1.8MB, respectively, which still considerably large, but much more suitable for smartphones. The DT models were the ones that presented the best trade-off in these experiments with average 91% accuracy and 131.1KB model size.

Nonetheless, the best performing RNN models also presented a good trade-off between classification accuracy and model size, reaching average 89% accuracy with average model size below 300KB. Figure 6.11 illustrate the impact of different numbers of layers and initial learning rate in RNNs accuracy and model size, respectively. Also, the variation of accuracy and model size when using dropout of 0.5 or no dropout can be observed. In general, increasing the number of layers or using dropout of 0.5 did not lead to better accuracy. The model sizes varied in the same way as the experiments with 5 second time windows as they cannot be influenced by this variable.

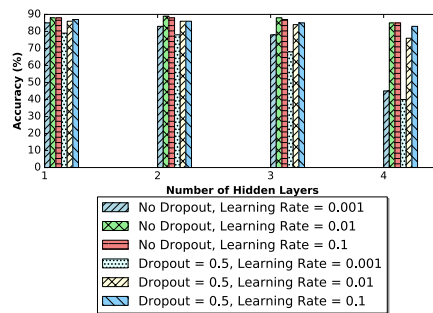


Figure 6.11: RNNs accuracy obtained for multiple hyperparameter configurations, using 10 second time windows.

Figure 6.14 presents the average train and test confusion matrices of the best performing RNN configuration, respectively, with 2 hidden layers, no dropout and initial learning rate of 0.01. The models built with this configuration were able to learn the patterns of the five travel modes present in the evaluation dataset very well, as can be observed on the training set confusion matrices. However, when analyzing the test matrices, it can be concluded that the Still and Walking patterns presented better generalization than the ones of Car, Train and Bus modes, although the true positive rate for these classes was above 80%.

In addition to the previous results, Figure 6.12 illustrate the impact of different numbers of layers and initial learning rate in (D)FNNs accuracy, respectively. Also, the variation of accuracy when using ReLu or Leaky-ReLu as hidden layer activation function can be observed.

For FNNs, the best performance was reached when using 1 hidden layer, Leaky ReLu hidden activation and initial learning rate of 0.1, while for DFNNs the best performance was reached with 2 hidden layer, Leaky ReLu hidden activation and initial learning rate

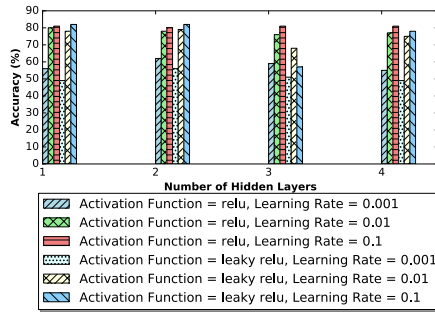


Figure 6.12: (D)FNNs accuracy obtained for multiple hyperparameter configurations, using 10 second time windows.

of 0.1. The average training and testing confusion matrices of the best performing FNNs and DFNNs configurations are presented in Figure 6.14 also.

6.4.3.4 Summary and Discussion

In this section, a summary and discussion of the performance evaluation results is provided. In Figure 6.13 the average accuracy and model size of the best performing configurations of each ML technique for each window size are presented.

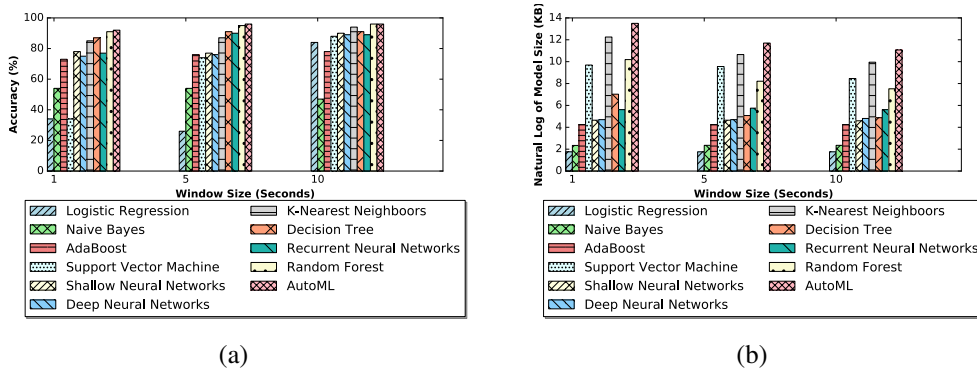


Figure 6.13: Accuracy and model size of best performing configurations of each ML technique for each window size.

With respect to classification accuracy, AutoML consistently presents superior performance across 1, 5 and 10 window size experiments. However, it also presents the largest model sizes across all experiments with its largest model consuming about 200 times more memory than the largest RNN models, which indicates that the later might be more indicated for in-device processing. The RF models consistently presented good results as well, with accuracy varying from 91% to 96% and model size varying from 26MB to 1.8MB. This means, however, that the largest RF model consumed on average 85 more memory than the largest RNN model, which also indicates that RNNs present an advantageous trade-off when compared to RFs.

The models generated with DT presented the best trade-off between classification accuracy and model size, which varied from 87% to 91% and 1.1MB to 131.1KB, respectively. When comparing these models the ones generated with RNNs, whose classification accuracy and model size varied from 77% to 90% and 314.5KB to 273.8KB, respectively, one might think that these could be more suited for in-device real-time travel mode detection. But when the computational complexities and the precision, recall, f1-score and kappa coefficient of the best performing configurations of both models are taken in consideration, RNNs have an advantage.

The models generated with FNNs and DFNNs presented modest performance, with average classification accuracy between 70% and 80%, when using 1 and 5 window sizes, but achieved great results, with average classification accuracy of around 90% when using 10 second time windows. Their lowest and largest model sizes were 99KB and 122.5KB, respectively. Models generated with AB presented similar results for 1 and 5 second window sizes but their performance did not increase substantially when using 10 second windows.

The worst performing models for 1 second windows, were the ones generated by SVM, LR and NB. When using 5 second windows, NB and LR obtained similar performance, but SVM presented a significant improvement. For 10 second windows however, both SVM and LR presented accuracy above 80%, which results in a huge gap between their performances with 1 and 10 second windows. In the case of LR, this can be explained by the fact that the models generated by these technique are incapable of capturing non-linear relationships between the input features, and larger window sizes can lead to more linear relationships.

The models generated with KNN also presented good results, with accuracy varying from 85% to 94% and model size varying from 20MB to more than 200MB. The main disadvantage of this technique, however, was the large model sizes it presented when compared to DT and RNN. Therefore, the results indicate that most of the models achieve decent performance with 10 second window sizes and although AutoML and RF achieve the highest accuracy, precision, recall, f1-score and kappa coefficient across all scenarios, DT and RNN present a better trade-off between classification performance and model size. Also, RNN have reached the best performance when using 5 second time windows, with 90% accuracy and model size of 314.5KB.

In order to evaluate how the samples of user one (U1), which was discarded from the experiments by the reasons stated in Subsection 6.4.1, could influence detection performance, we have executed an additional cross-validation experiment using the best per-

forming RNN configuration and a dataset containing all samples of U1. The average train and test configurations obtained in this experiment are presented in Figure 6.14.

It can be observed that the performance dropped considerably for all travel modes during training, with the exception of car and walking, which maintained a true positive rate above 90%. The same pattern can be observed in the test confusion matrix, although the model did not reach a 90% true positive rate in any of the travel mode classes. This indicates that U1 samples did not contribute for improving the model performance neither in training or testing and they should, therefore, be discarded from the evaluation experiments.

6.5 Conclusion

In this chapter, the problems of offline and online travel mode detection are addressed. These are relevant to many context-aware applications in smart cities. These applications include ITS, health care monitoring, location-based services and others. This work proposed TMDFramework and TMD-LSTM, that uses RNNs to learn travel mode patterns through multiple sensing modalities, and evaluated its online detection capabilities using a public benchmark dataset. The experimental results obtained indicate that the method proposed is capable of detecting modes with up to 90% accuracy and a lower memory consumption and computational cost than other ML approaches with comparable performance.



Figure 6.14: Average training and testing confusion matrices per-fold of best performing neural network configurations with 5, 1 and 10 second time windows. Predicted labels are represented as columns and true labels are represented as rows.

7. Conclusions and Future Work

In this chapter, the main conclusions of this work are summarized as well as the next steps to be taken for future research. This work has addressed four instances of the problem of real-time travel mode detection:

- Real-Time Travel Mode Detection with Location Sensors - Chapter 3.
- Real-Time Travel Mode and Trip Purpose Prediction with Location Sensor - Chapter 4.
- Real-Time Travel Mode Detection using Multiple Smartphone Sensors - Chapter 5.
- Real-Time Travel Mode Detection with Recurrent Neural Networks - Chapter 6.

In Chapter 3, a technique that allows detecting the travel mode of smartphone users, in real time, was proposed and evaluated. This technique was implemented on a prototype Android application, named CityTracks-RT, that used supervised machine learning models, built with the Weka API in Java. The performance of the prototype was evaluated through the analysis of quantitative metrics obtained via field tests with 37 volunteers in Rio de Janeiro and the results indicate that the proposed technique is capable of detecting the travel modes of smartphone users in urban centers.

In Chapter 4 a hybrid solution for real-time travel mode detection and trip purpose prediction was proposed. This solution uses a single preprocessing algorithm to extract features that are used to train classification models through supervised ML algorithms. Its performance was evaluated through the analysis of quantitative metrics obtained via k-fold cross-validation. The maximum accuracy reached for travel mode detection and trip purpose prediction was 88% and 81%, respectively. Therefore, it can be concluded that the proposed solution is capable of detecting the travel mode and predict the trip purpose of a trip using only 60 seconds of smartphone collected location data, allowing

context-aware information systems to use this contextual information to provide personalized services and better predict user behaviour with respect to mobility patterns and transportation preferences.

In Chapter 5 the use of AutoML and feature engineering techniques to enhance travel mode detection with multiple sensors is proposed. The evaluation experiments showed that these methods can greatly improve classification accuracy, reaching a maximum of 97% accuracy, precision, recall and f-measure on the TMD Dataset using the AutoSklearn Global Optimization framework. This is the first work to present evidence that AutoML frameworks can outperform human researchers in the combined algorithm selection and hyperparameter optimization of ML classifiers for the travel mode detection problem.

Finally, in Chapter 6, TMDFramework and TMD-LSTM are proposed. TMD-LSTM uses RNNs to learn travel mode patterns through multiple sensing modalities, and its online detection capabilities are evaluated using a public benchmark dataset. The experimental results obtained indicate that the method proposed is capable of detecting modes with up to 90% accuracy and a lower memory consumption and computational cost than other ML approaches with comparable performance.

As future research the reproduction of the evaluation experiments using iOS devices, in order to assess the difference in detection performance obtained through these and the Android devices, can be considered. Additionally, the implementation of all the components of the generic framework proposed in chapter 6 is needed. Finally, further evaluation experiments with RNNs and other deep learning techniques can be considered, combining handcrafted features and raw data as input for training and predictions.

Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- [2] Shigeo Abe. *Support Vector Machines for Pattern Classification*. Springer Publishing Company, Incorporated, 2nd edition, 2012.
- [3] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.
- [4] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [5] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [6] Per Kragh Andersen and Richard David Gill. Cox’s regression model for counting processes: a large sample study. *The annals of statistics*, pages 1100–1120, 1982.
- [7] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008.
- [8] Behrang Assemi, Hamid Safi, Mahmoud Mesbah, and Luis Ferreira. Developing and validating a statistical model for travel mode identification on smartphones. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):1920–1931, 2016.
- [9] Jiawei Bai, Jeff Goldsmith, Brian Caffo, Thomas A. Glass, and Ciprian M. Crainiceanu. Movelets: A dictionary of movement. *Electronic Journal of Statistics*, 6:559–578, 2012.
- [10] Thanos Bantis and James Haworth. Who you are is how you travel: A framework for transportation mode detection using individual and environmental characteristics. *Transportation Research Part C: Emerging Technologies*, 80:286–309, 2017.

- [11] Tooba Batool, Yves Vanrompay, An Neven, Davy Janssens, and Geerts Wets. Ctass: a framework for contextualized travel behavior advice to cardiac patients. *Procedia Computer Science*, 113:303–309, 2017.
- [12] L. Bedogni, M. Di Felice, and L. Bononi. By train or by car? detecting the user’s motion type through smartphone sensors data. In *2012 IFIP Wireless Days*, pages 1–6, Nov 2012.
- [13] Luca Bedogni, Marco Di Felice, and Luciano Bononi. Context aware android applications through transportation mode detection techniques. *Wireless Communications and Mobile Computing*, 16(16), 2016.
- [14] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
- [15] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.
- [16] R Bertolami, H Bunke, S Fernandez, A Graves, M Liwicki, and J Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), 2009.
- [17] J Martin Bland and Douglas G Altman. Statistics notes: measurement error. *Bmj*, 312(7047):1654, 1996.
- [18] Dankmar Böhning. Multinomial logistic regression algorithm. *Annals of the Institute of Statistical Mathematics*, 44(1):197–200, 1992.
- [19] Wendy Bohte and Kees Maat. Deriving and validating trip purposes and travel modes for multi-day gps-based travel surveys: A large-scale application in the netherlands. *Transportation Research Part C: Emerging Technologies*, 17(3):285–297, 2009.
- [20] Jean-Yves Le Boudec. *Performance Evaluation of Computer and Communication Systems*. EFPL Press, 2011.
- [21] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
- [22] E Oran Brigham and E Oran Brigham. *The fast Fourier transform and its applications*, volume 448. prentice Hall Englewood Cliffs, NJ, 1988.

- [23] S Allen Broughton and Kurt Bryan. *Discrete Fourier analysis and wavelets: applications to signal and image processing*. John Wiley & Sons, 2018.
- [24] A. Bujari, B. Licar, and C. E. Palazzi. Movement pattern recognition through smartphone’s accelerometer. In *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 502–506, Jan 2012.
- [25] Francesco Calabrese, Laura Ferrari, and Vincent D. Blondel. Urban sensing using mobile phone network data: A survey of research. *ACM Computing Survey*, 47(2):1–20, 2014.
- [26] Claudia Carpineti, Vincenzo Lomonaco, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Custom dual transportation mode detection by smartphone devices exploiting sensor diversity. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 367–372. IEEE, 2018.
- [27] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the Twenty-first International Conference on Machine Learning, ICML ’04*, pages 18–, New York, NY, USA, 2004. ACM.
- [28] Ke-Yu Chen, Rahul C. Shah, Jonathan Huang, and Lama Nachman. Mago: Mode of transport inference using the hall-effect magnetic sensor and accelerometer. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):8:1–8:23, June 2017.
- [29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.
- [30] Dongkeun Cho, Howon Kim, et al. Stateful lstm for classification on/off event of household appliances. *Proceedings of the Korean Institute of Communication Sciences Conference*, pages 15–16, 2018.
- [31] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- [32] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.
- [33] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

- [34] Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation Research Part C: Emerging Technologies*, 86:360–371, 2018.
- [35] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.
- [36] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [37] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.
- [38] Yuki Endo, Hiroyuki Toda, Kyosuke Nishida, and Akihisa Kawanobe. Deep feature extraction from trajectories for transportation mode estimation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 54–66. Springer, 2016.
- [39] Alireza Ermagun, Yingling Fan, Julian Wolfson, Gediminas Adomavicius, and Kirti Das. Real-time trip purpose prediction using online location-based search and discovery services. *Transportation Research Part C: Emerging Technologies*, 77:96–112, 2017.
- [40] Kelly R Evenson and Robert D Furberg. Moves app: a digital diary to track physical activity and location. *Br J Sports Med*, 51(15):1169–1170, 2017.
- [41] BS Everitt and A Skrondal. *The cambridge dictionary of statistics*. Cambridge, Cambridge, 2002.
- [42] Shih-Hau Fang, Yu-Xaing Fei, Zhezhuang Xu, and Yu Tsao. Learning transportation modes from smartphone sensors based on deep neural network. *IEEE Sensors Journal*, 17(18):6111–6118, 2017.
- [43] Shih-Hau Fang, Hao-Hsiang Liao, Yu-Xiang Fei, Kai-Hsiang Chen, Jen-Wei Huang, Yu-Ding Lu, and Yu Tsao. Transportation modes classification using sensors on smartphones. *Sensors*, 16(8):1324, 2016.
- [44] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.
- [45] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

- [46] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, 2007.
- [47] Denzil Ferreira, Vassilis Kostakos, and Anind K Dey. Aware: mobile context instrumentation framework. *Frontiers in ICT*, 2:6, 2015.
- [48] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.
- [49] David Heaven Fremlin. *Measure theory*, volume 4. Torres Fremlin, 2000.
- [50] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.
- [51] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [52] Nir Friedman, Dan Geiger, Moises Goldszmidt, G. Provan, P. Langley, and P. Smyth. Bayesian network classifiers. In *Machine Learning*, pages 131–163, 1997.
- [53] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Academic press, 2013.
- [54] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.
- [55] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.
- [56] Karst T Geurs, Tom Thomas, Marcel Bijlsma, and Salima Douhou. Automatic trip and mode detection with move smarter: First results from the dutch mobile mobility panel. *Transportation research procedia*, 11:247–262, 2015.
- [57] Rudolf Giffinger and Nataša Pichler-Milanović. *Smart cities: Ranking of European medium-sized cities*. Centre of Regional Science, Vienna University of Technology, 2007.

- [58] Pall Oskar Gislason, Jon Atli Benediktsson, and Johannes R Sveinsson. Random forests for land cover classification. *Pattern Recognition Letters*, 27(4):294–300, 2006.
- [59] Lei Gong, Ryo Kanamori, and Toshiyuki Yamamoto. Data selection in machine learning for identifying trip purposes and travel modes from longitudinal gps data collection lasting for seasons. *Travel Behaviour and Society*, 2017.
- [60] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [61] Ben Goodrich and Itamar Arel. Unsupervised neuron selection for mitigating catastrophic forgetting in neural networks. In *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 997–1000. IEEE, 2014.
- [62] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.
- [63] Damodar N Gujarati. *Basic econometrics*. Tata McGraw-Hill Education, 2009.
- [64] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macia, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, et al. Design of the 2015 chlearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.
- [65] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [66] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.
- [67] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [68] Harold R Jacobs. *Mathematics: A human endeavor*. Macmillan, 1994.
- [69] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

- [70] Xiang Jiang, Erico N de Souza, Ahmad Pesaranghader, Baifan Hu, Daniel L Silver, and Stan Matwin. Trajectorynet: An embedded gps trajectory representation for point-based classification using recurrent neural networks. *arXiv preprint arXiv:1705.02636*, 2017.
- [71] Sanem Kabadayi, Adam Pridgen, and Christine Julien. Virtual sensors: Abstracting data from physical sensors. In *Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, pages 587–592. IEEE Computer Society, 2006.
- [72] D Kinga and J Ba Adam. A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [73] Ron Kohavi. The power of decision tables. In Nada Lavrac and Stefan Wrobel, editors, *Machine Learning: ECML-95*, pages 174–189, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [74] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95*, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
- [75] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.
- [76] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 17:1–5, 2016.
- [77] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.
- [78] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, Sept 2010.
- [79] Zahra Ansari Lari and Amir Golroo. Automated transportation mode detection using smart phone applications via machine learning: Case study mega city of tehran. In *Proceedings of the Transportation Research Board 94th Annual Meeting, Washington, DC, USA*, pages 11–15, 2015.

- [80] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- [81] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
- [82] YW Lee, TP Cheatham, and JB Wiesner. Application of correlation analysis to the detection of periodic signals in noise. *Proceedings of the IRE*, 38(10):1165–1171, 1950.
- [83] Xiaoyuan Liang and Guiling Wang. A convolutional neural network for transportation mode detection based on smartphone platform. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 338–342. IEEE, 2017.
- [84] Ziheng Lin, Mogeng Yin, Sidney Feygin, Madeleine Sheehan, Jean-Francois Paiement, and Alexei Pozdnoukhov. Deep generative models of urban mobility. *IEEE Transactions on Intelligent Transportation Systems*, 2017.
- [85] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.
- [86] Michel Loève. *Probability theory: foundations, random sequences*. van Nostrand Princeton, NJ, 1955.
- [87] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.
- [88] Nelmarie Louw and SJ Steel. Variable selection in kernel fisher discriminant analysis by means of recursive feature elimination. *Computational Statistics & Data Analysis*, 51(3):2043–2055, 2006.
- [89] Heikki Mäenpää, Andrei Lobov, and Jose L Martinez Lastra. Travel mode estimation for multi-modal journey planner. *Transportation Research Part C: Emerging Technologies*, 82:273–289, 2017.
- [90] Bryan D Martin, Vittorio Addona, Julian Wolfson, Gediminas Adomavicius, and Yingling Fan. Methods for real-time prediction of the mode of travel using smartphone-based gps and accelerometer data. *Sensors*, 17(9):2058, 2017.

- [91] Bryan D Martin, Vittorio Addona, Julian Wolfson, Gediminas Adomavicius, and Yingling Fan. Methods for real-time prediction of the mode of travel using smartphone-based gps and accelerometer data. *Sensors*, 17(9):2058, 2017.
- [92] Chuishi Meng, Yu Cui, Qing He, Lu Su, and Jing Gao. Travel purpose inference with gps trajectories, pois, and geo-tagged social media data. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 1319–1324. IEEE, 2017.
- [93] Leandro L Minku, Allan P White, and Xin Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on knowledge and Data Engineering*, 22(5):730–742, 2010.
- [94] Lara Montini, Sebastian Prost, Johann Schrammel, Nadine Rieser-Schüssler, and Kay W Axhausen. Comparison of travel diaries generated from smartphone data and dedicated gps devices. *Transportation Research Procedia*, 11:227–241, 2015.
- [95] Lara Montini, Nadine Rieser-Schüssler, and Kay W Axhausen. Personalisation in multi-day gps and accelerometer data processing. In *14th Swiss Transport Research Conference (STRC)*, 2014.
- [96] Lara Montini, Nadine Rieser-Schüssler, Andreas Horni, and Kay Axhausen. Trip purpose identification from gps tracks. *Transportation Research Record: Journal of the Transportation Research Board*, pages 16–23, 2014.
- [97] Kevin P Murphy. Naive bayes classifiers. *University of British Columbia*, 18, 2006.
- [98] Daisik Nam, Hyunmyung Kim, Jaewoo Cho, and R Jayakrishnan. A model based on deep learning for predicting travel mode choice. In *Proceedings of the Transportation Research Board 96th Annual Meeting Transportation Research Board, Washington, DC, USA*, pages 8–12, 2017.
- [99] John Ashworth Nelder and R Jacob Baker. *Generalized linear models*. Wiley Online Library, 1972.
- [100] Philippe Nitsche, Peter Widhalm, Simon Breuss, Norbert Brändle, and Peter Maurer. Supporting large-scale travel surveys with smartphones—a practical approach. *Transportation Research Part C: Emerging Technologies*, 43:212–221, 2014.
- [101] Clifton Nock. *Data access patterns: database interactions in object-oriented applications*. Addison-Wesley Boston, 2004.
- [102] Bob O’hara and Al Petrick. *IEEE 802.11 handbook: a designer’s companion*. IEEE Standards Association, 2005.

- [103] Judy Pearsall and Patrick Hanks. *The new Oxford dictionary of English*. Clarendon Press, 1998.
- [104] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [105] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2011.
- [106] Adrian C. Prelicean, Gyözö Gidófalvi, and Yusak O. Susilo. Transportation mode detection – an in-depth review of applicability and reliability. *Transport Reviews*, 37(4):442–464, 2017.
- [107] C. A. de M. S. Quintella, Leila C.V. Andrade, and Carlos Alberto v. Campos. Detecting the transportation mode for context-aware systems using smartphones. In *IEEE Intelligent Transportation Systems (ITSC)*, 2016.
- [108] Carlos Alvaro de M. S. Quintella. Aplicação de aprendizado de máquina para inferência de modo de transporte em traces de smartphones, 2013.
- [109] Lawrence R Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall*, 777 p., 1975.
- [110] Tim Rentsch. Object oriented programming. *ACM Sigplan Notices*, 17(9):51–57, 1982.
- [111] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.
- [112] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [113] David Saad. *On-line learning in neural networks*, volume 17. Cambridge University Press, 2009.
- [114] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

- [115] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.
- [116] Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, and Joseph Barfett. Image augmentation using radial transform for training deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, Calgary, Alberta, Canada*, 2018.
- [117] Johannes Schobel, Rüdiger Pryss, Winfried Schlee, Thomas Probst, Dominic Gebhardt, Marc Schickler, and Manfred Reichert. Development of mobile data collection applications by domain experts: Experimental results from a usability study. In *International Conference on Advanced Information Systems Engineering*, pages 60–75. Springer, 2017.
- [118] Johannes Schobel, Marc Schickler, Rüdiger Pryss, and Manfred Reichert. Process-driven data collection with smart mobile devices. In *International Conference on Web Information Systems and Technologies*, pages 347–362. Springer, 2014.
- [119] Johann Schrammel, Marc Busch, and Manfred Tscheligi. Peacock-persuasive advisor for co2-reducing cross-modal trip planning. In *PERSUASIVE (Adjunct Proceedings)*, 2013.
- [120] Toru Seo, Takahiko Kusakabe, Hiroto Gotoh, and Yasuo Asakura. Interactive online machine learning approach for activity-travel survey. *Transportation Research Part B: Methodological*, 2017.
- [121] Muhammad Awais Shafique and Eiji Hato. Travel mode detection with varying smartphone data collection frequencies. *Sensors*, 16(5), 2016.
- [122] Li Shen and Peter R Stopher. Review of gps travel survey and gps data-processing methods. *Transport Reviews*, 34(3):316–334, 2014.
- [123] Roger W Sinnott. Virtues of the haversine. *Sky Telescope*, 68, Issue 2:159, 1984.
- [124] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.
- [125] Elton F. de S. Soares, Carlos Alvaro de M. S. Quintella, and Carlos Alberto V. Campos. Citytracks-rt: Uma aplicação para detecção do modo de transporte em tempo real

- nos centros urbanos. *Simpósio Brasileiro de Sistemas de Informação (SBSI 2017)*, 2017.
- [126] Elton F. de S. Soares, Carlos Alvaro de M. S. Quintella, and Carlos Alberto V. Campos. Towards an application for real-time travel mode detection in urban centers. *IEEE Vehicular Technology Conference*, 2017.
- [127] Elton F. de S. Soares, Kate Revoredo, Carlos Alvaro de M. S. Quintella, Carlos Alberto V. Campos, and Fernanda Baião. A hybrid solution for real-time travel mode detection and trip purpose prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2019.
- [128] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. In *IJCAI*, pages 2618–2624, 2016.
- [129] Peter R Stopher, Qingjian Jiang, Camden FitzGerald, et al. Processing gps data from travel surveys. *2nd international colloquium on the behavioural foundations of integrated land-use and transportation models: frameworks, models and applications*, Toronto, 2005.
- [130] Ruslan L Stratonovich. *Conditional Markov processes and their application to the theory of optimal control*. American Elsevier, 1968.
- [131] Xing Su, Hernan Caceres, Hanghang Tong, and Qing He. Online travel mode identification using smartphones with battery saving considerations. *IEEE Transactions on Intelligent Transport. Systems*, 17(10), 2016.
- [132] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.
- [133] Pang-Ning Tan et al. *Introduction to data mining*. Pearson Education India, 2007.
- [134] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 847–855, New York, NY, USA, 2013. ACM.
- [135] Philip J Troped, Marcelo S Oliveira, Charles E Matthews, Ellen K Cromley, Steven J Melly, and Bruce A Craig. Prediction of activity mode with global posi-

- tioning system and accelerometer data. *Medicine and science in sports and exercise*, 40(5):972–978, 2008.
- [136] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.
- [137] Graham Upton and Ian Cook. *Understanding statistics*. Oxford University Press, 1996.
- [138] Toan H Vu, Le Dung, and Jia-Ching Wang. Transportation mode detection on mobile devices using recurrent nets. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 392–396. ACM, 2016.
- [139] Bao Wang, Linjie Gao, and Zhicai Juan. Travel mode detection using gps data and socioeconomic attributes based on a random forest classifier. *IEEE Transactions on Intelligent Transportation Systems*, 2017.
- [140] Hao Wang, GaoJun Liu, Jianyong Duan, and Lei Zhang. Detecting transportation modes using deep neural network. *IEICE TRANSACTIONS on Information and Systems*, 100(5):1132–1135, 2017.
- [141] Chieh-Hua Wen and Frank S Koppelman. The generalized nested logit model. *Transportation Research Part B: Methodological*, 35(7):627–641, 2001.
- [142] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.
- [143] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.
- [144] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.
- [145] George Woodbury. *An introduction to statistics*. Cengage Learning, 2001.
- [146] Linlin Wu, Biao Yang, and Peng Jing. Travel mode detection based on gps raw data collected by smartphones: a systematic review of the existing methodologies. *Information*, 7(4):67, 2016.
- [147] Guangnian Xiao, Zhicai Juan, and Chunqin Zhang. Travel mode detection based on gps track data and bayesian networks. *Computers, Environment and Urban Systems*, 54:14–22, 2015.

- [148] Guangnian Xiao, Zhicai Juan, and Chunqin Zhang. Detecting trip purposes from smartphone-based travel surveys with artificial neural networks and particle swarm optimization. *Transportation Research Part C: Emerging Technologies*, 71:447–463, 2016.
- [149] Fei Yang, Zhenxing Yao, and Peter J Jin. Gps and acceleration data in multi-mode trip data recognition based on wavelet transform modulus maximum algorithm. *Transportation Research Record*, pages 90–98, 2015.
- [150] Meng-Chieh Yu, Tong Yu, Shao-Chen Wang, Chih-Jen Lin, and Edward Y Chang. Big data small footprint: The design of a low-power classifier for detecting transportation modes. *Proceedings of the VLDB Endowment*, 7(13):1429–1440, 2014.
- [151] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256. ACM, 2008.
- [152] Xiaolu Zhu, Jinglin Li, Zhihan Liu, Shangguang Wang, and Fangchun Yang. Learning transportation annotated mobility profiles from gps data for context-aware mobile services. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 475–482. IEEE, 2016.