DETERMINATION OF TRANSPORTATION MODES USING MOBILE PHONES

A Graduate Project Report submitted to Manipal University in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Electronics and Communication Engineering

Submitted by

Debasmita Ghose

Reg. No: 130907158

Under the guidance of

Mr. Siew Kei Lam
(External)
Assistant Professor,
School of Computer Science and Engineering,
Nanyang Technological University, Singapore

&

Mrs. Aparna U.
(Internal)
Assistant Professor,
Department of Electronics and Communication Engineering,
Manipal Institute of Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

(A Constituent College of Manipal University)

MANIPAL – 576104, KARNATAKA, INDIA

JUNE - JULY 2017

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

(A Constituent College of Manipal University)

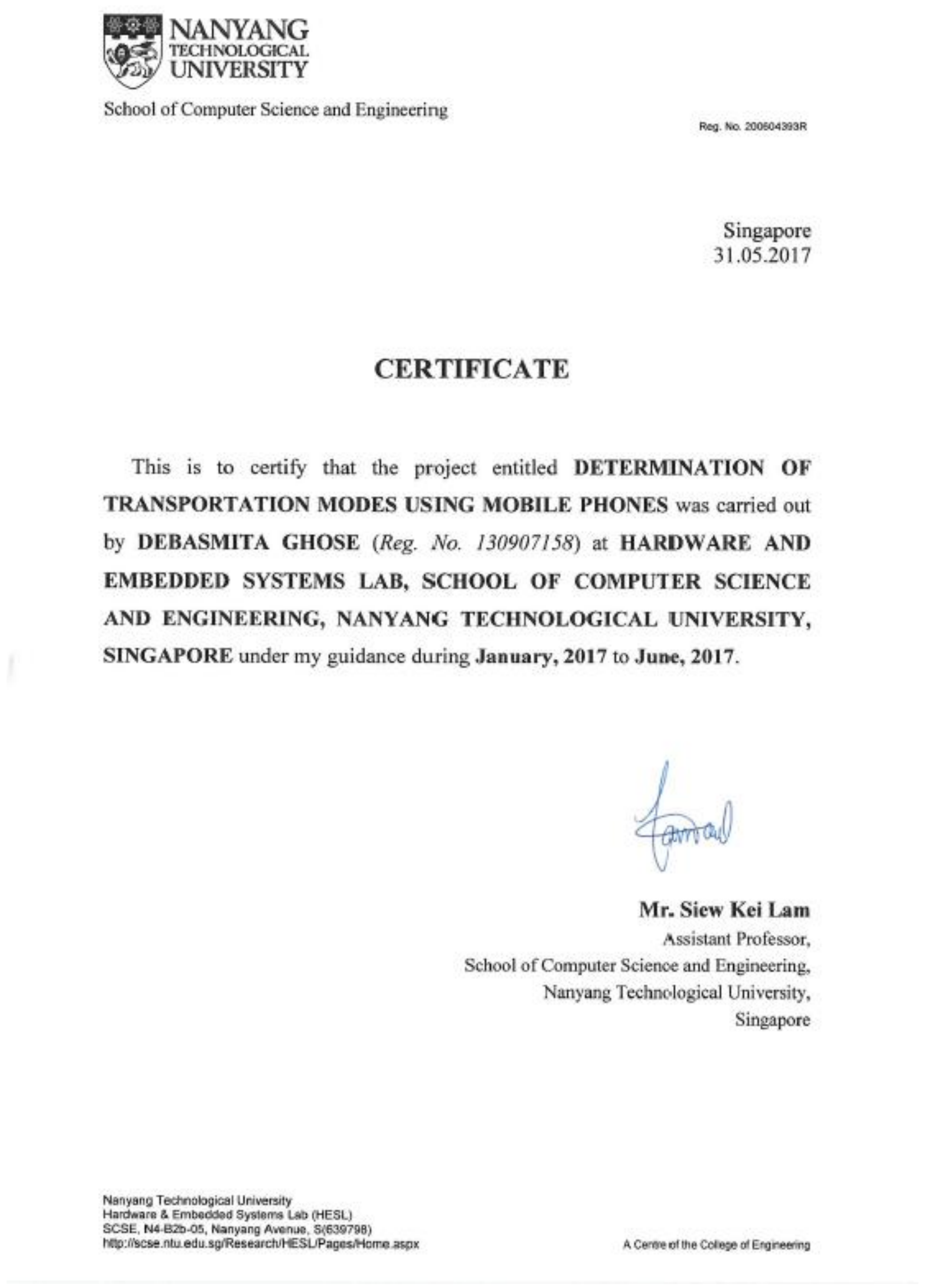
MANIPAL – 576 104 (KARNATAKA), INDIA

Manipal
05-07-2017

CERTIFICATE

This is to certify that the project titled **DETERMINATION OF TRANSPORTATION MODES USING MOBILE PHONES** is a record of the bonafide work done by **DEBASMITA GHOSE** (*Reg. No. 130907158*) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (BTech) in **ELECTRONICS AND COMMUNICATION ENGINEERING** of Manipal Institute of Technology Manipal, Karnataka, (A Constituent College of Manipal University), during the academic year 2016 - 2017.

Mrs. Aparna U.	**Prof. Dr. M. Sathish Kumar**
Project Guide	*HOD, E & C.*
	M.I.T, MANIPAL



NANYANG TECHNOLOGICAL UNIVERSITY

School of Computer Science and Engineering

Reg. No. 200604393R

Singapore
31.05.2017

CERTIFICATE

This is to certify that the project entitled **DETERMINATION OF TRANSPORTATION MODES USING MOBILE PHONES** was carried out by **DEBASMITA GHOSE** (*Reg. No. 130907158*) at **HARDWARE AND EMBEDDED SYSTEMS LAB, SCHOOL OF COMPUTER SCIENCE AND ENGINEERING, NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE** under my guidance during **January, 2017** to **June, 2017**.

[illegible]

Mr. Siew Kei Lam
Assistant Professor,
School of Computer Science and Engineering,
Nanyang Technological University,
Singapore

Nanyang Technological University
Hardware & Embedded Systems Lab (HESL)
SCSE, N4-B2b-05, Nanyang Avenue, S(639798)
http://scse.ntu.edu.sg/Research/HESL/Pages/Home.aspx

A Centre of the College of Engineering

ACKNOWLEDGMENTS

I take pleasure to express my sincere acknowledgements to all those people who have rendered the support and guidance in order to complete the present research project.

First of all, I am indebted to **Prof. B.V.R. Chowdari**, Senior Executive Director, President's Office, Nanyang Technological University (NTU), Singapore for selecting me for the NTU India Connect Research Internship Programme.

I express my deep sense of gratitude to **Prof. Thambipillai Srikanthan**, Professor, School of Computer Science and Engineering, Nanyang Technological University, Singapore and Prof. **Siew Kei Lam**, Associate Professor, School of Computer Science and Engineering, Nanyang Technological University, Singapore for providing me the opportunity to work on the project "**Determination of Transportation Modes using Mobile Phones**" at the Hardware and Embedded Systems Lab, School of Computer Science and Engineering, Nanyang Technological University, Singapore and providing the required infrastructural facilities to carry out the research work.

I would also like to thank **Mr. George Rosario Jagadeesh**, Research Associate, Hardware and Embedded Systems Lab, School of Computer Science and Engineering, Nanyang Techological University, Singapore for constantly guiding me through the entire project through his active involvement, and encouraging me to learn and try new things.

This project would not have been possible without the constant support and encouragement of **Dr. Gopalakrishna Prabhu**, Director, Manipal Institute of Technology, **Dr. Somashekara Bhat,** Associate Director (Development), Manipal Institute of Technology and **Dr. M. Sathish Kumar**, Head of the Department of Electronics and Communication Engineering, Manipal Institute of Technology.

I am thankful to **Mrs. Aparna U.**, Assistant Professor, Department of Electronics and Communication Engineering, Manipal Institute of Technology, who was my internal guide for this project for supporting me through all stages of this project.

I would also like to thank the lab in-charge of Hardware and Embedded Systems Lab, School of Computer Science and Engineering, Nanyang Technological University, Singapore, **Mr. Jeremia Chua** for always providing the necessary technical support.

Last but not the least, I would like to express my special thanks to my family members specially my mother Dr. Debjani Sarkar Ghose, my father Mr. Soumitra Kumar Ghose, my grand-mother Mrs. Snigdha Ghose, and my fellow interns at Nanyang Technological University, Singapore for their constant prayers, encouragement and cooperation for the completion of the project.

Debasmita Ghose
8th Semester,
Department of Electronics and Communication Engineering,
Manipal Institute of Technology, Manipal University,
Manipal - 576104

ABSTRACT

The public transportation system is a crucial part of the solution to the nation’s economic, energy and environmental challenges. This transportation system can be made more efficient, by studying the mobility of the users, from the sensor data collected by them. The current project have been developed to compare two methods of identifying the mode of transportation being used by a person using the location data collected from one’s cell phone.

The transportation modes, such as walking, driving, etc., that a user takes, can enrich the user’s mobility with informative knowledge and provide pervasive computing systems with more context information.

The raw data is time sequenced location data retrieved from the GPS in the cell phones of the users. This data was pre-processed in order to extract relevant features like velocity,, acceleration and heading change and to filter out datasets with huge gaps and redundancy. A change point segmentation algorithm based on the work by Zheng et. al. [2,3], where, the data was first broken down into segments using a loose bound of velocity and acceleration, segments smaller than a certain threshold were merged into the backward segment and consecutive segments of length smaller than another threshold were merged to form one segment was implemented using a Python Script. For every segment retrieved from the change point segmentation algorithm, the average velocity, the average acceleration and the average heading change has been calculated, and have been used as a feature for classification. The current work, uses selected trajectories of the GeoLife dataset [2,3,12], to train a decision tree based model, and uses GPS trajectory data collected from a user taking multimodal trips across Singapore, in order to test the model, using a Python script.

A comparative study of the two algorithms i.e. Decision Tree and Random Forest was performed on some trajectories of the GeoLife Dataset [2] and the Random Forest algorithm was found to be more accurate than the Decision Tree algorithm with an overall accuracy of 81.2%. A comparison of the work by Reddy et al. [1] and Zheng et al. [2,3] has also been performed, where the Decision Tree and the Random Forest classifier has been applied on a point to point basis instead of a segment to segment basis. It has been found that classification after change point segmentation, gives a higher overall accuracy, and the accuracy obtained for the current dataset using a Random Forest classifier is 96.7% in discerning between motorised and non-motorised modes of transportation (walk and bus), which is higher than what has been obtained in the previous work.

Therefore, the change point segmentation and classification algorithms are highly suited to build a solution to the problem of discerning between motorised and non-motorised modes of transport, using a series of time-stamped GPS locations. In the real world, this project can not only help the land transport authorities make the public transportation system more efficient, but can also help the police in law enforcement, if a system which collects real time GPS logs of people is put in place.

In order to achieve these results, Weka Machine Learning Toolkit, ArcGIS, MS Excel and Scikit Learn Library of Python is being used.

LIST OF TABLES

LIST OF FIGURES

Contents

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Mobile phones are ubiquitous devices. They have computational, sensing and communication capabilities and are carried by people throughout the day. Modern day mobile phones are capable of collecting time sequenced location data of the user using the GPS modules integrated with them. This data can be used to discern between the different transportation modes being used by people, thus endowing their mobility with more significance and providing pervasive computing systems with rich context information.

In the past, travel surveys have taken multiple formats, such as telephone interviews and questionnaires. These data collection strategies depend on manual labelling of the data after the trip, thus introducing inaccuracies. The use of GPS technologies in travel surveys have many advantages over the traditional methods. Some of them are:

1) It collects data passively, so it requires little time and effort from the respondent, thus reducing respondent burden and increasing data accuracy.
2) The GPS data shows the traveller's exact route choice, which is difficult and time consuming to collect via the traditional methods.
3) The GPS data shows the exact speed of the vehicle, which can be used to estimate the level of service on the transportation network.
4) This form of data collection causes very low respondent fatigue, so the length of the surveys can be increased from traditional single day to multiple day data collection.
5) Data is available at a finer resolution.

This project uses time sequenced location data collected using the GPS modules inbuilt in the users' cell phone, in order to discern between the two transportation modes 'Walking' and 'Bus'. The GPS traces were first broken down into segments, based on a four step algorithm, and sub traces of individual travel modes were separated. For each of the sub-traces, the average velocity and acceleration were calculated, and a decision tree based classifier was used to classify the traces into different modes. The results obtained by classification of this segmented dataset was compared with implementing a decision tree based classifier on a point by point basis on the raw GPS data. Further, the consecutive points or segments which were classified to belong to the same mode of transportation, were combined into a single mode.

This chapter is organised as follows. The first section of this chapter discusses the motivation behind this project. The second section talks about the objective of the project. The third section explains the target specifications. The fourth section discusses the project work schedule and the fifth section talks about the organisation of the project report.

1.2 MOTIVATION

The public transportation system is a crucial part of the solution to the nation’s economic, energy, and environmental challenges - helping to bring a better quality of life. This public transportation system can be made more efficient, by studying the mobility of the users, from the sensor data collected by them. With the knowledge of the traveller’s transportation mode, targeted and customised notifications regarding disruptions or other problems in the transport network, can be sent to specific user groups who are either using that public transportation service in that area or are frequent travellers or residents of that area.

Another motivation behind this project is to identify the demand of the public transportation in different areas in the city. The project will help the public transportation authorities to understand the demand of the public transportation in different areas at different times, thus helping them channelize their resources in a more efficient way. Further, it would benefit the residents to avoid traffic congestion and reduce waiting times for passengers.

The other application of travel mode detection is to collect crowd sourced real-time traffic information, in which traffic speeds are aggregated from mobile phones of users which act like probes. Transportation mode detection enables the aggregation system to filter out speed data submitted by people using non-motorised modes from speed data submitted by people using motorized modes.

Also, by understanding the transportation mode of each GPS track, users can absorb rich knowledge from each other people’s travel experience. They are able to know where others have been and also how everyone reaches their destination. Using this data, the system should recommend a bus line rather than a driving way, to promote the use of public transport.

1.3 OBJECTIVE

The following are the objectives of this project:

- To infer the mode of transportation (motorised or non-motorised) in compound trips, which contain more than one kind of transportation modes, using only GPS traces and independent of other information from maps and other sensors.
- To correctly detect the transition between the different transportation modes.
- To apply the model learned from the dataset of some users, on the GPS traces of other users to infer their mode of transportation.

1.4 PROJECT WORK SCHEDULE

January 2017	o Literature Review o Studied of the basic concepts in Machine Learning and other pre-requisites
February 2017	o Chose a suitable dataset o Preprocessed the data to extract suitable trips to be used for classification using MS Excel o Calculated features for classification o Chose a suitable classification model and features using Weka Machine Learning Toolkit and compared the classification accuracies of different models using different features.
March 2017	o Implemented a basic classifier to identify if the user is stationary or in motion using the Sci-kit Learn Toolkit of Python on a small sample dataset. o Exported the data from a CSV file into the Python environment for manipulation. o Chose a suitable segmentation algorithm to detect the change points of the transportation modes, from the existing work.
April 2017	o Implemented a four step change point segmentation algorithm using a Python script, in order to detect the change points of transportation modes. o Optimized the change point segmentation algorithm by changing the thresholds for the different features to suit the selected dataset. o Calculated the classification features on each of the segments. o Implemented a decision tree based classification algorithm to distinguish between the two modes of transportation “Walking” and “Bus” and calculated the classification accuracy.
May 2017	o Optimized the decision tree based classifier by changing the values of the different parameters. o Implemented a random forest classifier on the dataset and compared the classification accuracy with the one obtained using the decision tree classifier. o Implemented the decision tree and random forest classifier on un-segmented data and compare the classification accuracies with the one obtained using segmented data. o Used a data visualization technique to export the decision trees to a PDF file.
June 2017	o Documentation

1.5 ORGANIZATION OF THE PROJECT REPORT

Chapter 2 explains the background theory required to understand the various elements of the project. It also gives a brief literature review of the papers studied during the project.

Chapter 3 explains in detail the methodology used in building the project. It talks about the various software tools employed and the architecture of the project. It also explains the features used and the reason behind selecting the features.

Chapter 4 does a detailed analysis of the experiments conducted and the results obtained, the significance of the results obtained, the deviations from the expected results and its justification.

Chapter 5 summarises the project and explains the future scope of the work and the relevance of the work in the present scenario.

CHAPTER 2
BACKGROUND THEORY

2.1 INTRODUCTION

The project “Determination of Transportation Modes using Mobile Phones”, aims to employ the data collected by GPS over a period of time, in order to determine the mode of transportation used by a person.

Mobile phones have become an indispensable commodity in the modern day world, so much so, they are being used for numerous applications, that people had not imagined 20 years ago. This project uses mobile phones as probes to gather information about people’s mobility in order to detect the mode of transportation being used by them. This data is in the form of series of coordinates traversed by a person, collected at an interval of 1-5 seconds using the GPS module inbuilt in the user’s cell phone. This series of coordinates are used to calculate the various features used to finally determine the mode of transportation used by him.

This chapter is organised as follows. The second section describes briefly, the classification methodologies that can possibly be used for transport mode classification and explains the mathematical derivations. The third section gives a brief literature review of the research papers studied and does a comparative analysis of the different classification methodologies used. This chapter also does a comparative analysis of the number of modes classified by the different authors and the features used by them to classify the same. The fourth section gives a concludes the chapter by comparing the work done previously, with the work done in this project.

2.2 DESCRIPTION OF THE CLASSIFIERS

2.2.1 Naïve Bayes Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes’ theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of$P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

2.2.2 Decision Tree Classifier

Decision Trees are a non-parametric supervised learning method used for classification. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called over-fitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Given training vectors $x_i \in R^n$, i=1,..., l and a label vector $y \in R^l$, a decision tree recursively partitions the space such that the samples with the same labels are grouped together.
Let the data at node m be represented by Q. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ subsets

$$Q_{left}(\theta) = (x, y) | x_j <= t_m$$

$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta)$$

The impurity at m is computed using an impurity function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q, \theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_\theta G(Q, \theta)$$

Recurse for subsets $Q_{left}(\theta^*)$ and $Q_{right}(\theta^*)$ until the maximum allowable depth is reached, $N_m < \min_{samples}$ or $N_m = 1$.

If a target is a classification outcome taking on values 0,1,...,K-1, for node m, representing a region R_m with $N_m$observations, let

$$p_{mk} = 1/N_m \sum_{x_i \in R_m} I(y_i = k)$$

be the proportion of class k observations in node m

Common measures of impurity are Gini

$$H(X_m) = \sum_k p_{mk}(1 - p_{mk})$$

Cross-Entropy

$$H(X_m) = -\sum_k p_{mk} \log(p_{mk})$$

and Misclassification

$$H(X_m) = 1 - \max(p_{mk})$$

2.2.3 Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

2.2.4 Support Vector Machine Classifier

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of **margin** within SVM's theory. Therefore, the optimal separating hyper-plane *maximizes* the margin of the training data.

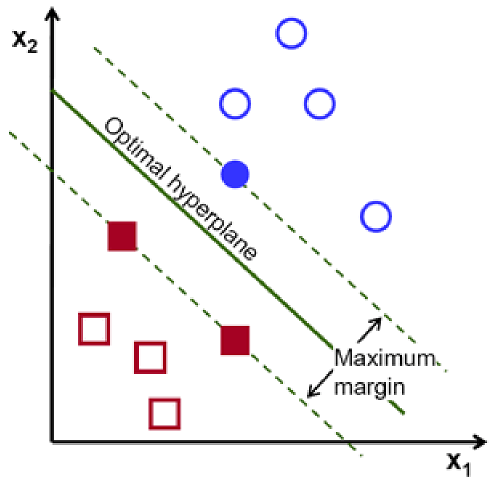


Figure 1: Support Vector Machine Example

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

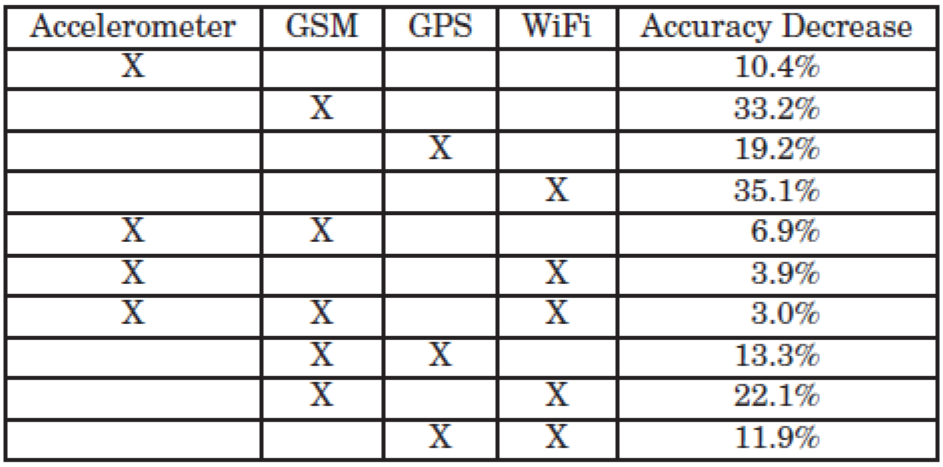
The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

2.3 LITERATURE REVIEW

The work by Reddy et al. [1] focuses on using mobile phones to determine the transportation mode of an individual when outside, whether the user is stationary, walking, running, biking or in motorized transport. They have performed a detailed analysis of the different sensors that can be used to collect data and have done a comparative analysis of the accuracy as shown in Table 2.

Table 2: Comparison of Classification Accuracy Decrease between Combinations of Different Sensing Devices [1]

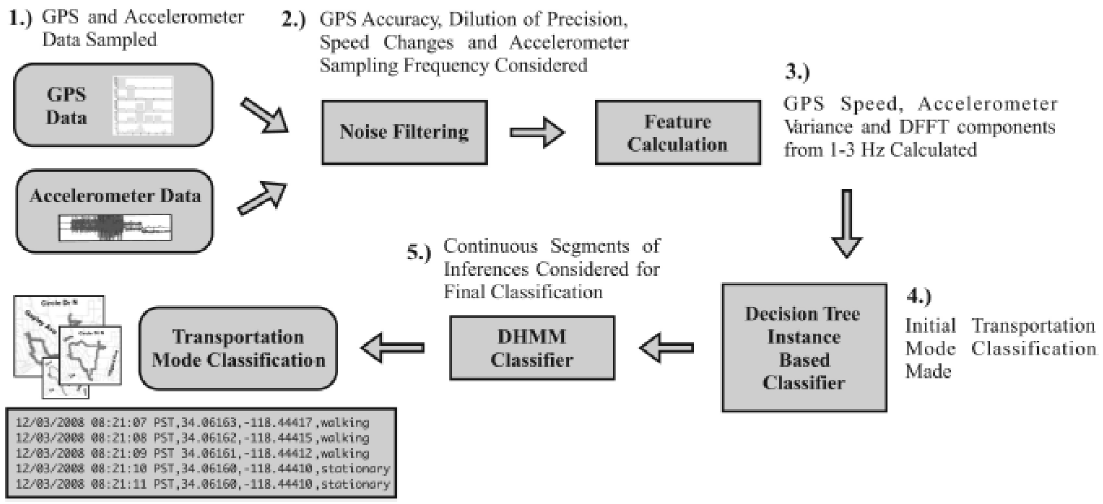


Accelerometer	GSM	GPS	WiFi	Accuracy Decrease
X				10.4%
	X			33.2%
		X		19.2%
			X	35.1%
X	X			6.9%
X			X	3.9%
X	X		X	3.0%
	X	X		13.3%
	X		X	22.1%
		X	X	11.9%

The two sensors that are sampled for data include the accelerometer and the GPS. In terms of the accelerometer, the magnitude of the force vector by combining the measurements from all three axis are used as a basis for the features. In terms of speed, the value obtained from the GPS receiver is directly used.

They used Hidden Markov Model and Decision Tree as the classifier to classify between the different modes. They implemented the system on a Nokia N95 device using the device's accelerometer and GPS module. The data used for training and testing the classifier was collected by 16 individuals over a period of time. Also, a study was performed where the phone was placed in different parts of the body and a comparison was performed between the results.

A 10 fold cross validation was performed on a decision tree classifier and an overall precision and recall accuracy was recorded to be 91.3%. The structure of the overall classifier is shown on Figure 2.


1.) GPS and Accelerometer Data Sampled
GPS Data
Accelerometer Data
2.) GPS Accuracy, Dilution of Precision, Speed Changes and Accelerometer Sampling Frequency Considered
Noise Filtering
Feature Calculation
3.) GPS Speed, Accelerometer Variance and DFFT components from 1-3 Hz Calculated
Decision Tree Instance Based Classifier
4.) Initial Transportation Mode Classification Made
DHMM Classifier
5.) Continuous Segments of Inferences Considered for Final Classification
Transportation Mode Classification

```
12/03/2008 08:21:07 PST,34.06163,-118.44417,walking
12/03/2008 08:21:08 PST,34.06162,-118.44415,walking
12/03/2008 08:21:09 PST 34.06161,-118.44412,walking
12/03/2008 08:21:10 PST,34.06160,-118.44410,stationary
12/03/2008 08:21:11 PST,34.06160,-118.44410,stationary
```

Figure 2: Structure of the Overall Classifier as proposed by S. Reddy et. al. [1]

The work by Zheng et al. [2,3,12] introduces the GeoLife dataset, which is a dataset of mobility data made available publically by Microsoft Research, Asia. It contains mobility data of 178 users in a period of over four years (from April 2007 to October 2011). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. This dataset contains 17,621 trajectories with a total distance of 1,251,654 kilometres and a total duration of 48,203 hours. These trajectories were recorded by different GPS loggers and GPS-phones, and have a variety of sampling rates. The work describes in detail about the architecture devised by them in inferring the transportation modes using time sequenced location data. The architecture proposed by them is shown in Figure 3.

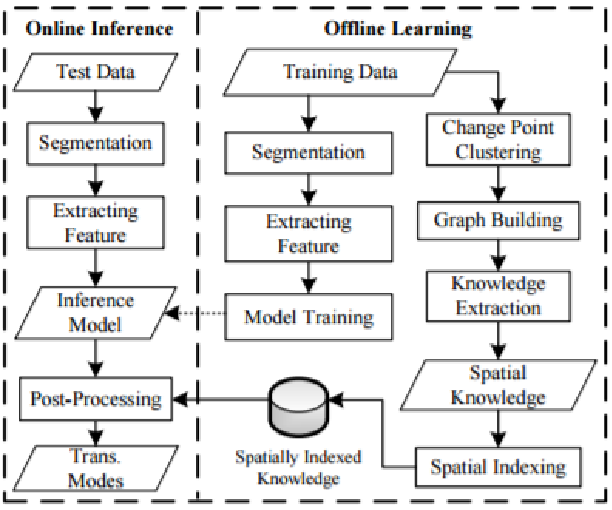
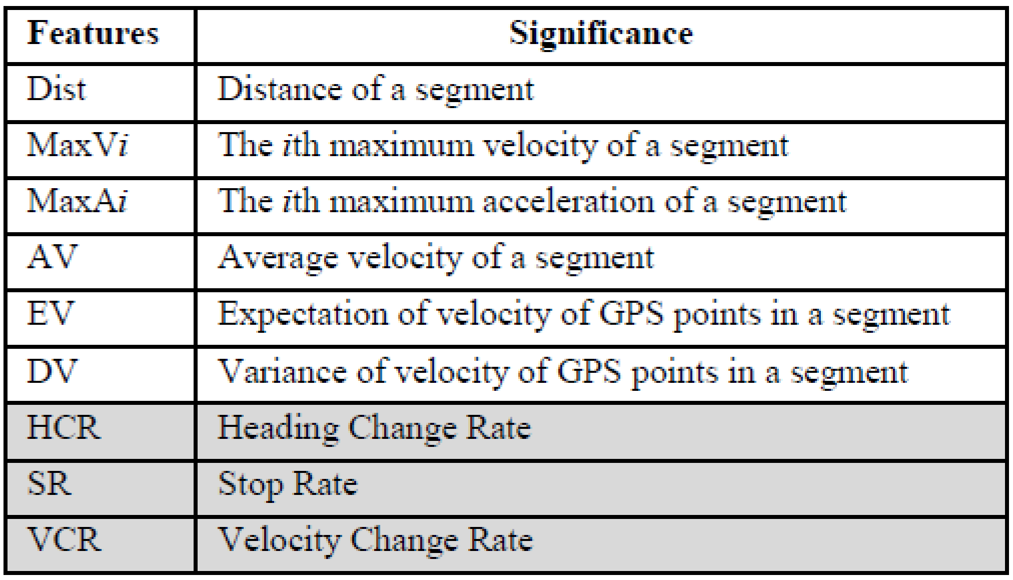


Figure 3: The proposed architecture proposed by Zheng, Yu. et. al. [2, 3, 12]

The test and the training data are extracted from the GeoLife dataset which contains labels for the different trajectories. They have proposed a segmentation algorithm, which first loosely classifies the points into Walk and Non-Walk points using velocity and acceleration. It then merges segments on the basis of the number of points in a particular segments.

They have also defined different features which can be used for classification between different modes. They have been listed in Table 2.

Table 1: List of Features used in the Experiment [2, 3, 12]



Features	Significance
Dist	Distance of a segment
MaxV*i*	The *i*th maximum velocity of a segment
MaxA*i*	The *i*th maximum acceleration of a segment
AV	Average velocity of a segment
EV	Expectation of velocity of GPS points in a segment
DV	Variance of velocity of GPS points in a segment
HCR	Heading Change Rate
SR	Stop Rate
VCR	Velocity Change Rate

They have found the decision tree classifier to be the most accurate for mode classification using the features defined above. They obtained a classification accuracy of 76.2% using the four step segmentation algorithm, Stop Rate, Heading Change Rate and Velocity Change Rate as features and a Decision Tree based inference model on the GeoLife dataset.

Jonsson [4] has proposed a theoretical model which can be used for mode classification. He has divided his classification algorithm into two parts namely segmentation and segment mapping as shown in Figure 4.

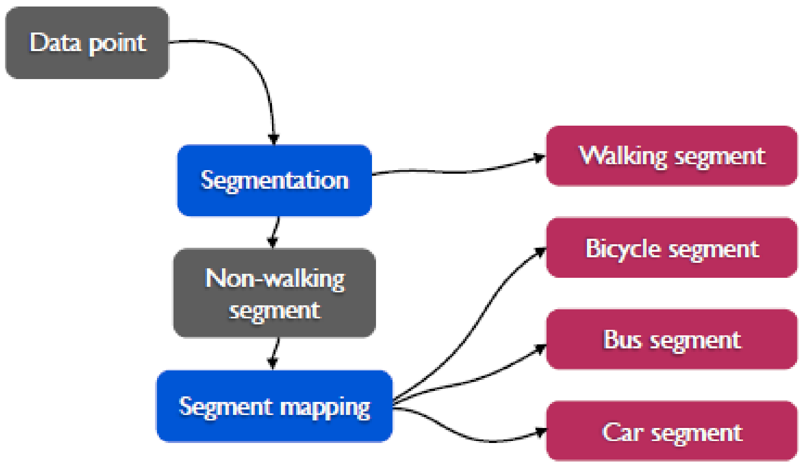
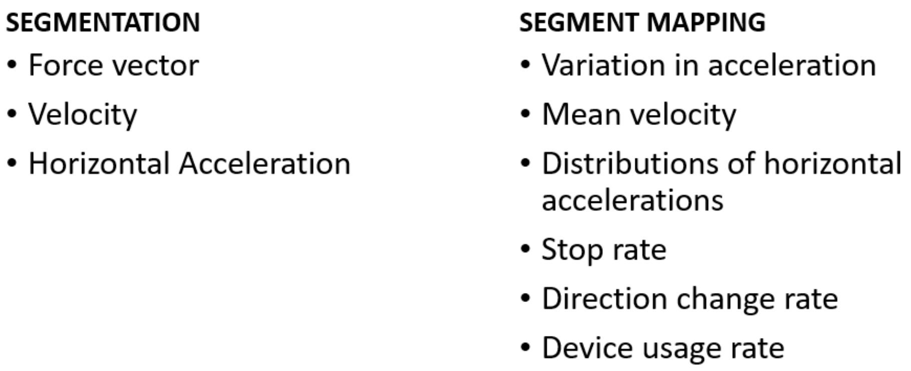


Figure 4: Descriptive model of the classification system as proposed by Jonnson, Fredrick [4]

The work describes different features for segmentation and segment mapping as listed in Table 3.

Table 2: List of features used for Segmentation and Segment Mapping [4]



SEGMENTATION	SEGMENT MAPPING
• Force vector	• Variation in acceleration
• Velocity	• Mean velocity
• Horizontal Acceleration	• Distributions of horizontal accelerations
	• Stop rate
	• Direction change rate
	• Device usage rate

Decision trees have been proposed to perform segmentation and segment mapping as shown in Figures 5 and 6.

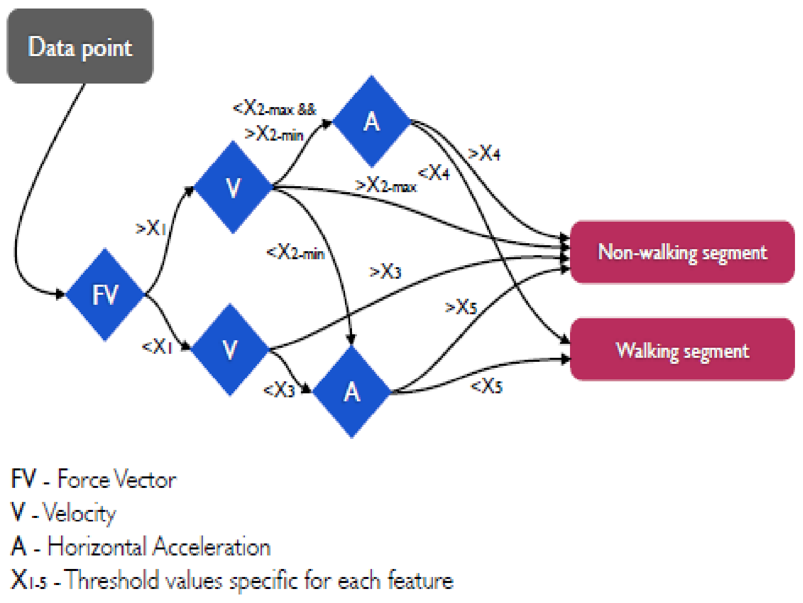


Figure 5: Proposed Decision Tree for Segmentation [4]

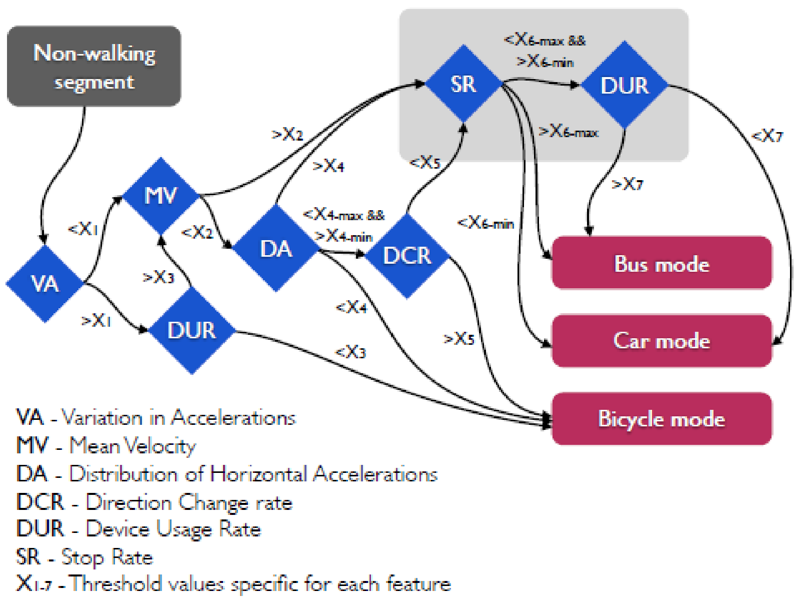


Figure 6: Proposed Decision Tree for Segment Mapping [4]

Stenneth et al. [5] use the features average accuracy of GPS coordinates, average speed, average heading change and average acceleration for mode classification in their work. They contradict the work by Zheng et al. [2,3,12], who describe heading change rate as a feature and have found average heading change to be a better feature to perform mode classification.
They compared the precision and recall accuracy of five distinct classification models namely Naïve Bayes, Bayesian Network, Decision Trees, Random Forest and Multilayer Perceptron using the WEKA Machine Learning Tool. Their results indicate that Random Forest is the best model, with an average precision accuracy of 93.70% and recall of 93.80% and they chose this as their final algorithm.

Table 4 gives a summary of the work that has taken place in this space using a similar approach as ours. This table gives the activity modes inferred, the test user base and the classification accuracy.

Table 3: Summarised Outcome of the Literature Review

Work	Classes	Users	Time	Accuracy
Zheng et al. [3]	Walk, Bike, Car, Bus	45	6 months	74%
Zheng et al. [2]	Walk, Bike, Motorised	65	10 months	76.2%
Gong et al. [6]	Walk, Car, Bus, Subway, Rail	53	5 days	82.6%
Liao et al.[20] Patterson et al. [21]	Walk, Motorised	1	60 days	84%
Liao et al. [8]	Still, Walk, Motorised	4	7 days	85%
Chen et al. [9]	Walk, Bike, Motorised	65	10 months	87.1%
Witayangkurn et al. [7]	Still, Walk, Bike, Car, Train	100	1 month	87.8%
Shalby et al. [23]	Walk, Bike, Car, Bus	1	1 month	92%
Stenneth et al. [5]	Still, Walk, Bike, Car, Bus, Train	6	3 weeks	92.9%
Reddy et al. [1]	Still, Walk, Run, Bike, Motorised	16	50 days	93.6%

2.4 CONCLUSION

Various authors have done significant works in the field of transportation mode detection. Several algorithms for segmentation of trips and mode classification have been proposed in the state of the art, and they have been tested in different cities all across the world. However, no significant models have been built, specifically for Singapore. Cottrill et al. [31] and Xiao et al. [32] have attempted to develop activity monitoring models for Singapore, by collecting mobility data across Singapore, but they have not tested a transportation mode detection system using this dataset.

The current work, uses selected trajectories of the GeoLife dataset [2 ,3, 12], to train a decision tree based model, and uses GPS trajectory data collected from a user taking multimodal trips across Singapore , in order to test the model. This gave overall accuracy of about 96.7% was obtained, using the decision tree classifier, on the data collected across Singapore, which is higher than what has been obtained in the state the art.

CHAPTER 3
METHODOLOGY

3.1 INTRODUCTION

This project aims to test different approaches in transportation mode detection using a series of time-stamped GPS locations. For this, the problem was broken down into two major parts i.e. segmentation of the trajectories and the classification of the travel modes. A four step segmentation algorithm proposed by Zheng et al. [2, 3, 12] was used, in order to segment a trajectory into sub-traces each belonging to one mode of transportation. For each of these sub-traces, different features were calculated, and suitable features were chosen for classification. Different classification algorithms were used to build models, in order to perform mode classification and calculating the performance accuracy, by comparing with the ground truth as annotated in the data set. Selected trajectories of the GeoLife dataset was used for training the decision tree based classifier to build a model, and the data collected by a user taking multimodal trips across Singapore was used for testing the model.

This chapter is organised as follows. The second section explains in detail the software, libraries and tools used in the project. The third section explains in detail the architecture of the proposed system, and every step in that architecture. The fourth section explains briefly the results expected out of the system.

3.2 SOFTWARE TOOLS

3.2.1 Weka Machine Learning Tool

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from a Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License. Weka's main user interface is the *Explorer*, but essentially the same functionality can be accessed through the component-based *Knowledge Flow* interface and from the command line. There is also the *Experimenter*, which allows the systematic comparison of the predictive performance of Weka's machine learning algorithms on a collection of datasets.

The *Explorer* interface features several panels providing access to the main components of the workbench:

- The *Preprocess* panel has facilities for importing data from a database, a comma-separated values (CSV) file, etc., and for pre-processing this data using a so-called *filtering* algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.
- The *Classify* panel enables applying classification and regression algorithms (indiscriminately called *classifiers* in Weka) to the resulting dataset, to estimate the accuracy of the resulting predictive model, and to visualize erroneous predictions, receiver operating characteristic (ROC) curves, etc., or the model itself (if the model is amenable to visualization like, e.g., a decision tree).
- The *Associate* panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The *Cluster* panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. There is also an implementation of the expectation maximization algorithm for learning a mixture of normal distributions.
- The *Select attributes* panel provides algorithms for identifying the most predictive attributes in a dataset.
- The *Visualize* panel shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analysed further using various selection operators.

3.2.2 Scikit Learn Library (PYTHON)

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The library is focused on modelling data. It is not focused on loading, manipulating and summarizing data. Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabelled data such as KMeans.
- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behaviour.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.
- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.
- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.

- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive Bayes, lazy methods, neural networks, support vector machines and decision trees.

3.2.3 ArcGIS

ArcGIS is a geographic information system (GIS) for working with maps and geographic information. It is used for: creating and using maps; compiling geographic data; analysing mapped information; sharing and discovering geographic information; using maps and geographic information in a range of applications; and managing geographic information in a database.

The system provides an infrastructure for making maps and geographic information available throughout an organization, across a community, and openly on the Web.

3.3 SYSTEM ARCHITECTURE

The architecture of the proposed system is briefly described in Figure 7.

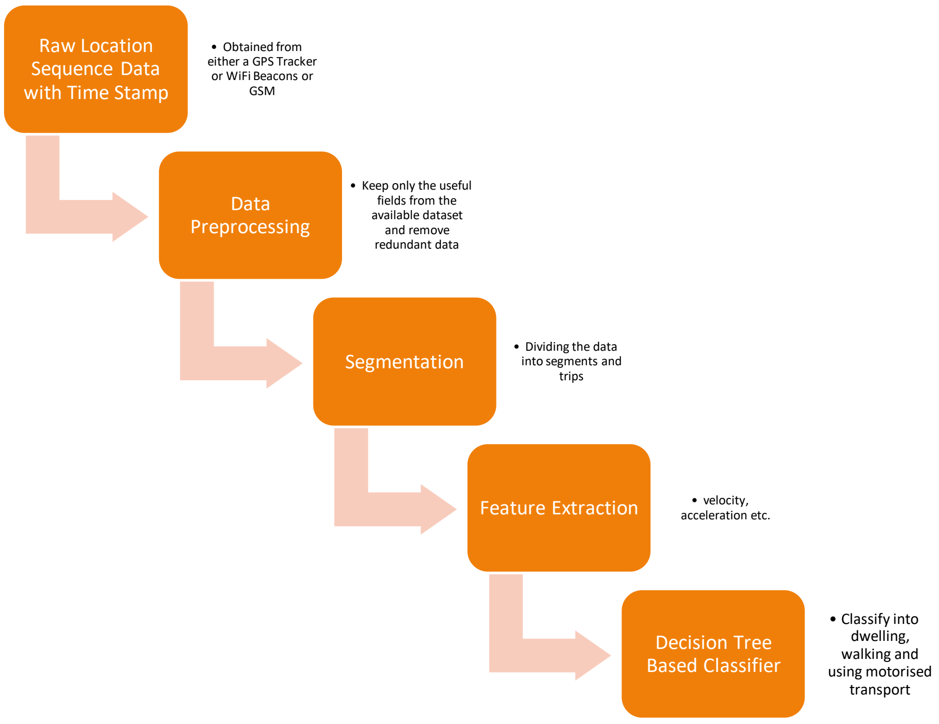

Raw Location Sequence Data with Time Stamp
• Obtained from either a GPS Tracker or WiFi Beacons or GSM
Data Preprocessing
• Keep only the useful fields from the available dataset and remove redundant data
Segmentation
• Dividing the data into segments and trips
Feature Extraction
• velocity, acceleration etc.
Decision Tree Based Classifier
• Classify into dwelling, walking and using motorised transport

Figure 7: Architecture of the Proposed System

3.3.1 Dataset

Some parts of the GeoLife dataset is being used, which is a dataset of mobility data made available publically by Microsoft Research, Asia. It contains mobility data of 178 users in a period of over four years (from April 2007 to October 2011). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. This dataset contains 17,621 trajectories with a total distance of 1,251,654 kilometres and a total duration of 48,203 hours. These trajectories were recorded by different GPS loggers and GPS-phones, and have a variety of sampling rates.

This dataset recoded a broad range of users' outdoor movements, including not only life routines like go home and go to work but also some entertainments and sports activities, such as shopping, sightseeing, dining, hiking, and cycling. Although this dataset is wildly distributed in over 30 cities of China and even in some cities located in the USA and Europe, the majority of the data was created in Beijing, China. 69 users have labelled their trajectories with **transportation mode**, such as driving, taking a bus, riding a bike and walking.

Every single folder of this dataset stores a user's GPS log files, which were converted to PLT format. Each PLT file contains a single trajectory and is named by its starting time. To avoid potential confusion of time zone, we use **GMT** in the date/time property of each point.
Field 1: Latitude in decimal degrees.
Field 2: Longitude in decimal degrees.
Field 3: All set to 0 for this dataset.
Field 4: Altitude in feet (-777 if not valid).
Field 5: Date - number of days (with fractional part) that have passed since 12/30/1899.
Field 6: Date as a string.
Field 7: Time as a string.

In addition to that, time sequenced location data was collected across Singapore using the GPS Logger Android Application in the following format.
Field 1: Latitude in decimal degrees.
Field 2: Longitude in decimal degrees.
Field 3: Bearing
Field 4: Altitude in feet (-777 if not valid).
Field 5: Date and time
Field 6: Velocity
Field 7: Number of Satellites

In order to filter out useful trajectories out of the GeoLife Dataset, an Excel sheet was used. Trajectories with gaps in data collection, which are less than 120 seconds, and containing at least 3 different mode changes between Bus and Walking were selected.

For further testing, GPS trajectory of 1 user was collected, which contained about 27,000 data points broken down into 4 independent trajectories, and the modes used were walking, waiting (stationary) and bus. This was used as a testing data set, for the classifier trained using the 20 trajectories extracted from the GeoLife dataset.

3.3.2 Segmentation

The segmentation method used is the one described in the work by Zheng et. al. [2,3,12]. The approach is derived from the following common sense of the real world:

1. People stop and then go while changing transportation modes
2. Walking should be a transition between different transportation modes

Therefore, the start point and end points of a *Walk Segment* could be a change point in very high probability.

Based on this, a four step Segmentation algorithm is demonstrated.
Step 1: Using a loose upper bound of velocity (V_t) and that of acceleration (a_t) to distinguish all possible *Walk Points* from *non-Walk Points*.
Step 2: If the length of a segment composed by consecutive *Walk Points* or *non-Walk Points* is less than a threshold, merge this one into its backward segment.
Step 3**:** If the length of a segment exceeds a certain threshold, the segment is regarded as a *Certain Segment*. Otherwise it is deemed as an *Uncertain Segment*. If the number of consecutive *Uncertain Segments* exceeds a certain threshold, these *Uncertain Segments* will be merged into one *non-Walk Segment.*
Step 4**:** The start point and end point of each *Walk Segment* are potential change points to partition a trip.

3.3.3 Features Used

A. Average Speed

The speed is computed from successive location changes since the direct GPS speed is not available for the GeoLife Dataset. The average speed will be computed for a series of GPS report.

Let {p_1, p_2, p_3, $p_4 \ldots p_n$} be a finite set of GPS sensor reports submitted from the traveller's mobile device within a time window.

Average accuracy = $\sum_{i=1\ to\ n} p_i^{acc}/n$

where p_i^{acc} is the estimated accuracy of the reported GPS position.

B. Average Heading Change

The heading is the direction from true north. For a set of GPS reports, the average heading change is computed.

Let $\{p_1, p_2, p_3, p_4 \ldots p_n\}$ represent a finite set of GPS reports submitted within a time window.

$$\textbf{Average heading change} = \sum_{i=1\ to\ n} \left| p_i^h - p_{i-1}^h \right| / n$$

$$\text{for all } 2 \leq \text{i} \leq \text{n}$$

where, p_i^h is the direction from true north included in the GPS sensor report.

C. Average Acceleration

Let $\{p_1, p_2, p_3, p_4 \ldots p_n\}$ be the finite set of GPS reports submitted within a time window.

$$p_i^{acceleration} = (p_i^v - p_{i-1}^v)/(p_i^t - p_{i-1}^t), \text{ for all } 2 \leq \text{i} \leq \text{n}$$

$$\textbf{Average acceleration} = \sum_{i=1\ to\ n} p_i^{acceleration} / n$$

where $p_i^{acceleration}$ is the acceleration of the mobile device.

3.3.4 Classification Algorithm

Decision trees and random forests will be used as the classification algorithms, as they have been found to provide the most accurate outputs according to the literature surveyed.

MS Excel was used to calculate the values of the features and convert the dataset into CSV files. This CSV file was converted into an ARFF file and this was fed to the WEKA Machine Learning Tool. The tool built a decision tree using the features specified and calculated the threshold values.

The issue faced while using the WEKA Machine Learning Toolkit is that it does not specified the classified instances, which is required in this case. So, Scikit Learn Library of Python, is being used to write a script that uses decision tree as a classifier, on the segmented dataset. A python script was written to segment the data according the algorithm by Zheng. et al [2], and a script was written to calculate the features of each segment and implement Decision Tree and Random forests to the segmented data.

Also, the work by Reddy, Sasank et al [1], proposes a point by point classification of data, instead of classification of segmented data. Therefore, performance of this method, was compared with the method proposed by Zheng et al. [2], by using Decision Tree and Random Forests algorithm on un-segmented data.

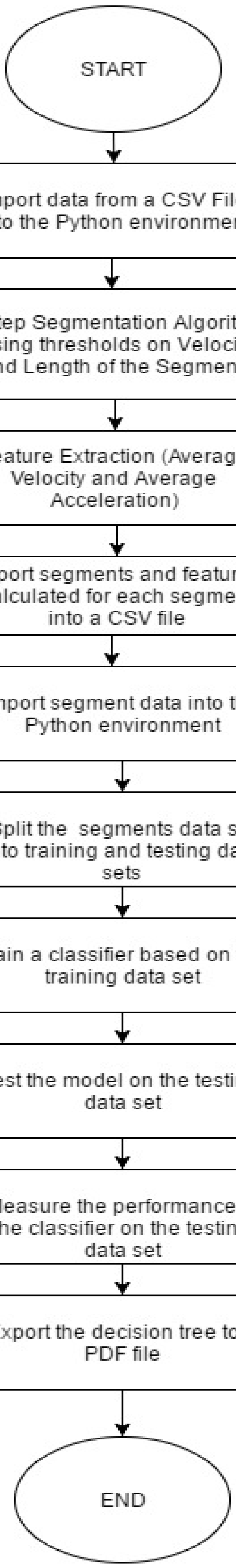

START
Import data from a CSV File into the Python environment
4 step Segmentation Algorithm using thresholds on Velocity and Length of the Segments
Feature Extraction (Average Velocity and Average Acceleration)
Export segments and features calculated for each segment into a CSV file
Import segment data into the Python environment
Split the segments data set into training and testing data sets
Train a classifier based on the training data set
Test the model on the testing data set
Measure the performance of the classifier on the testing data set
Export the decision tree to a PDF file
END

Figure 8: Flowchart of the Program used to implement the Project on Segmented Data

3.4 CONCLUSION

This chapter described in detail, the architecture of the transportation mode detection system and the tools used to test the various approaches, proposed in the state of the art, on the data collected in Singapore and selected trajectories of the GeoLife dataset.

A preliminary study was conducted using the Weka Machine Learning Toolkit, to select the most suitable features for classification and the classification algorithm. ArcGIS has been used to visualise the data on an OpenStreetMap, in order to identify any gaps or non-uniformity in the data stream. Then, a Python script was written in order to segment the data into sub-traces, with each sub-trace containing data belonging to one mode of transportation. Sci-kit learn library of Python, was further used to build a classification model, in order to perform mode classification. Finally, PyDot library of Python was used in order to visualise the data in a PDF file [Figure 8].

This method is an amalgamation of various methods suggested by different researchers working on transportation mode classification. Different parts of this method has been tested on different transportation datasets, and high classification accuracy has been obtained for each of these methods. Therefore, this method was chosen in order to find a solution to the problem of transportation mode classification.

CHAPTER 4
RESULT ANALYSIS

4.1 INTRODUCTION

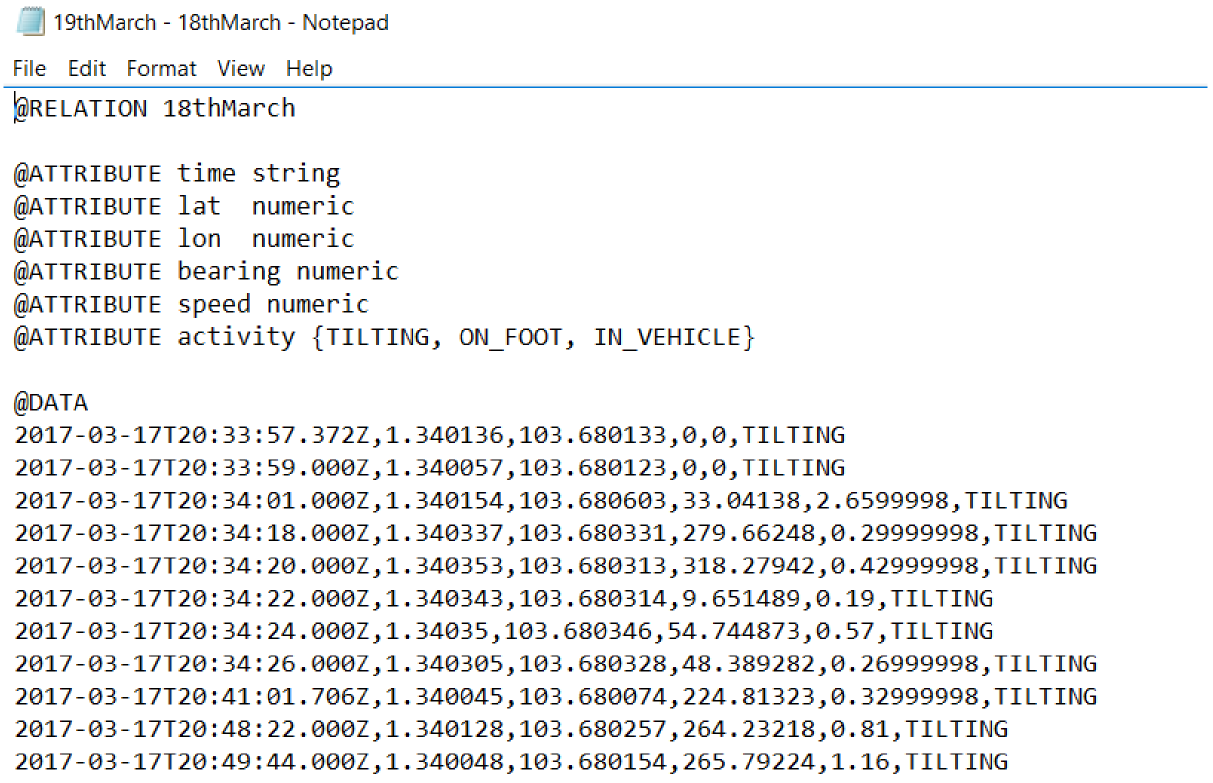
This project aims to test different approaches proposed by various researchers, to perform transportation mode classification in the context of Singapore. In order to test different approaches, a series of inter-related experiments were performed, the results obtained are explained in this chapter. .

In the second section, a detailed analysis of all the results obtained has been done. The third section talks about the significance of the results obtained. The fourth section discusses any deviation from the expected results and the fifth section gives the concluding statements about the results obtained.

4.2 RESULT ANALYSIS

4.2.1 Experiment 1: Weka Machine Learning Tool

Figure 9 shows the data converted to Attribute Relation File Format (ARFF). This file was used as an input to the Decision Tree classifier in the Weka Machine Learning Tool.



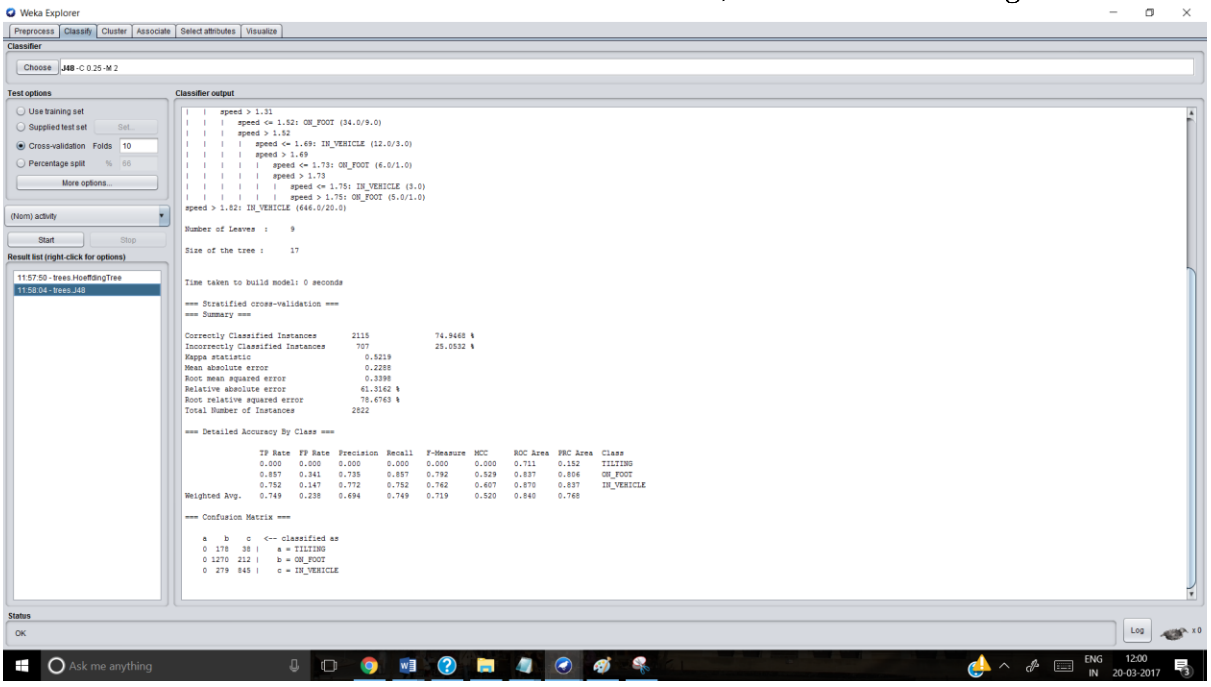
```
@RELATION 18thMarch

@ATTRIBUTE time string
@ATTRIBUTE lat  numeric
@ATTRIBUTE lon  numeric
@ATTRIBUTE bearing numeric
@ATTRIBUTE speed numeric
@ATTRIBUTE activity {TILTING, ON_FOOT, IN_VEHICLE}

@DATA
2017-03-17T20:33:57.372Z,1.340136,103.680133,0,0,TILTING
2017-03-17T20:33:59.000Z,1.340057,103.680123,0,0,TILTING
2017-03-17T20:34:01.000Z,1.340154,103.680603,33.04138,2.6599998,TILTING
2017-03-17T20:34:18.000Z,1.340337,103.680331,279.66248,0.29999998,TILTING
2017-03-17T20:34:20.000Z,1.340353,103.680313,318.27942,0.42999998,TILTING
2017-03-17T20:34:22.000Z,1.340343,103.680314,9.651489,0.19,TILTING
2017-03-17T20:34:24.000Z,1.34035,103.680346,54.744873,0.57,TILTING
2017-03-17T20:34:26.000Z,1.340305,103.680328,48.389282,0.26999998,TILTING
2017-03-17T20:41:01.706Z,1.340045,103.680074,224.81323,0.32999998,TILTING
2017-03-17T20:48:22.000Z,1.340128,103.680257,264.23218,0.81,TILTING
2017-03-17T20:49:44.000Z,1.340048,103.680154,265.79224,1.16,TILTING
```

Figure 9: Data in ARFF Format

Figure 10 shows the output of the Decision Tree Classifier in the Weka Machine Learning Tool, using 10 fold cross validation. The classifier accuracy is 74.94%. The tool found Speed to be the most efficient feature and created a 9 level tree, which is shown in Figure 11.



J48 -C 0.25 -M 2

Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66

(Nom) activity

Result list (right-click for options)
11:57:50 - trees.HoeffdingTree
11:58:04 - trees.J48

```
|   |   speed > 1.31
|   |   |   speed <= 1.52: ON_FOOT (34.0/9.0)
|   |   |   speed > 1.52
|   |   |   |   speed <= 1.69: IN_VEHICLE (12.0/3.0)
|   |   |   |   speed > 1.69
|   |   |   |   |   speed <= 1.73: ON_FOOT (6.0/1.0)
|   |   |   |   |   speed > 1.73
|   |   |   |   |   |   speed <= 1.75: IN_VEHICLE (3.0)
|   |   |   |   |   |   speed > 1.75: ON_FOOT (5.0/1.0)
speed > 1.82: IN_VEHICLE (646.0/20.0)

Number of Leaves  :     9

Size of the tree :      17


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2115               74.9468 %
Incorrectly Classified Instances       707               25.0532 %
Kappa statistic                          0.5219
Mean absolute error                      0.2288
Root mean squared error                  0.3398
Relative absolute error                 61.3162 %
Root relative squared error             78.6763 %
Total Number of Instances             2822

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    0.000      0.000    0.000      0.000    0.711     0.152     TILTING
                 0.857    0.341    0.735      0.857    0.792      0.529    0.837     0.806     ON_FOOT
                 0.752    0.147    0.772      0.752    0.762      0.607    0.870     0.837     IN_VEHICLE
Weighted Avg.    0.749    0.238    0.694      0.749    0.719      0.520    0.840     0.768

=== Confusion Matrix ===

    a    b    c   <-- classified as
    0  178   38 |    a = TILTING
    0 1270  212 |    b = ON_FOOT
    0  279  845 |    c = IN_VEHICLE
```

Status
OK

ENG IN 12:00 20-03-2017

Figure 10: Output of the Decision Tree Classifier

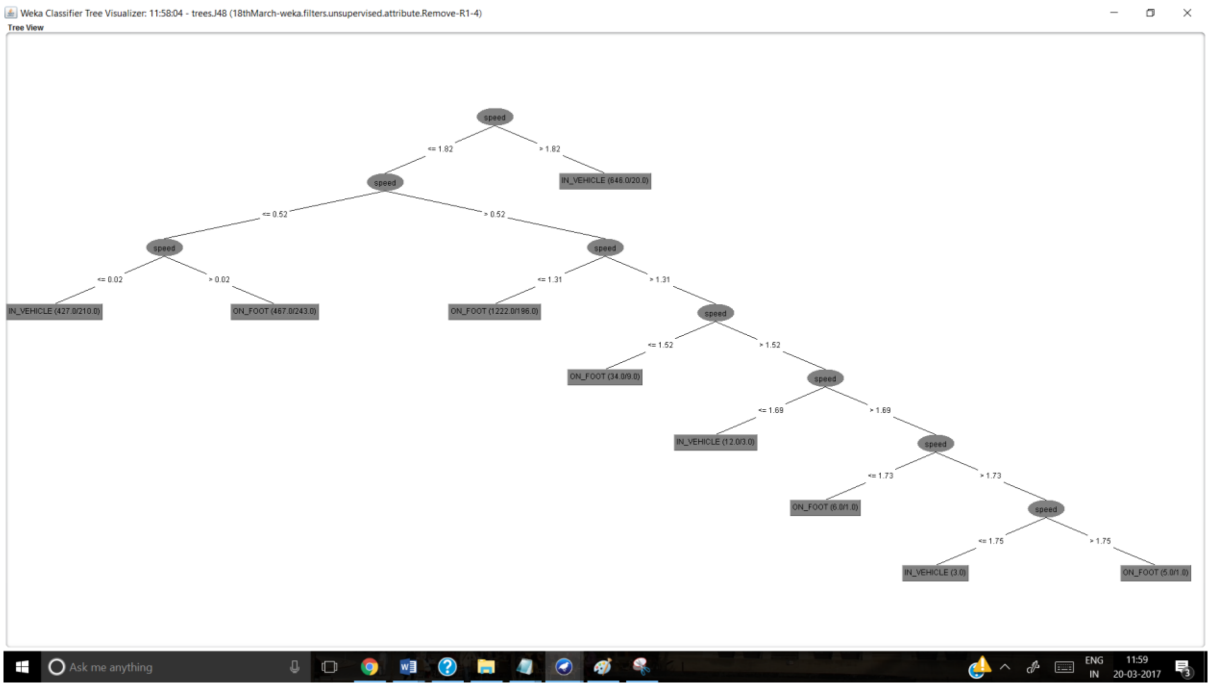


Figure 11: Decision Tree built by Weka Machine Learning Tool

Figure 12 shows the data collected plotted on a raster version of OpenStreetMap of Singapore on ArcGIS to show the distribution of data on the map.

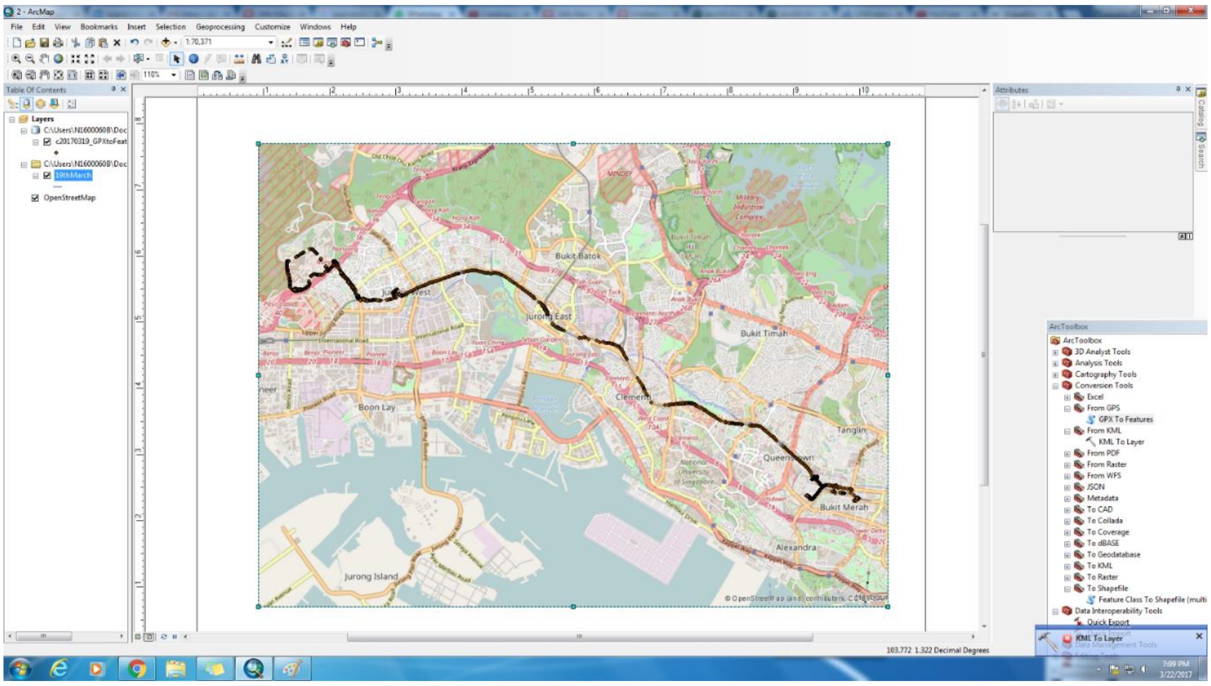


Figure 12: Data points on an OpenStreetMap

4.2.2 Experiment 2: Using Python Script on selected trajectories of the GeoLife Dataset (for both training and testing)

20 trajectories were selected from the GeoLife dataset, using an MS Excel sheet. The selected trajectories satisfy the following criteria:

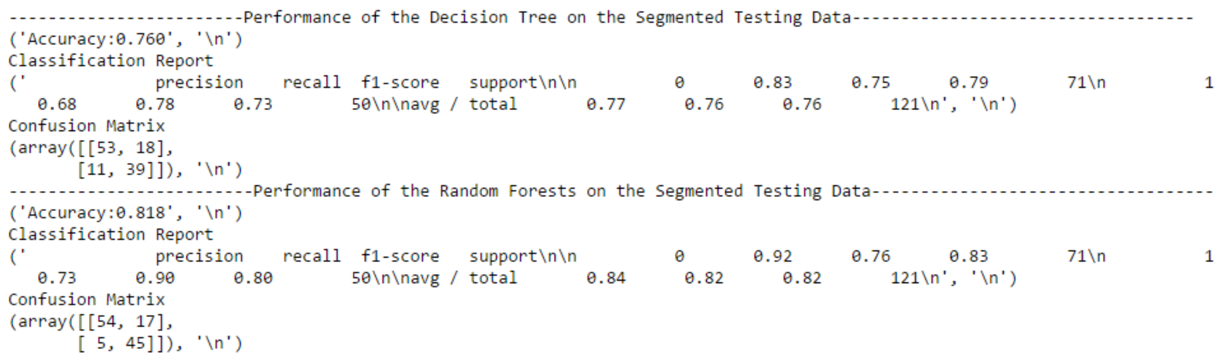
I. At no point in a trajectory, the gap between two points is more than 120 seconds.
II. The modes used in those trajectories is either Walk or Bus
III. There are at least 3 change-points in one trajectory.

These trajectories were then segmented using the four step segmentation algorithm proposed by Zheng et al. [2], which was implemented using a Python Script [Appendix 3]. Features of average velocity and average acceleration was computed for each of the computed trajectories. These segments were exported to a CSV file, which has been used as in input to the classification algorithms.

A test-train split was performed on these segments, in order to separate the segments into training and testing data. Then, the decision tree and random forest algorithms were implemented on both the training data and the testing data.

Those 20 trajectories containing 27,542 data points, were segmented into 1,206 trajectories using the four step segmentation algorithm. Each segment contains data belonging to any one particular mode of transportation.

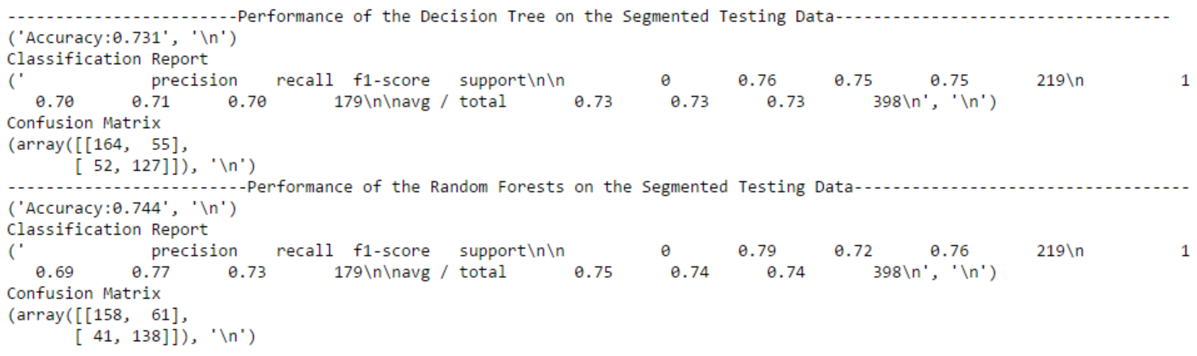
Using a test-train split of 0.1, which means 90% of the data was used to train the classifier and 10% of data was used to test the classifier, the overall accuracy of the decision tree classifier was 76% and the overall accuracy of the random forest classifier was 81.8% as shown by Figure 12.



```
-----------------------Performance of the Decision Tree on the Segmented Testing Data----------------------------------
('Accuracy:0.760', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.83      0.75      0.79        71\n          1
   0.68      0.78      0.73        50\n\navg / total       0.77      0.76      0.76       121\n', '\n')
Confusion Matrix
(array([[53, 18],
       [11, 39]]), '\n')
------------------------Performance of the Random Forests on the Segmented Testing Data----------------------------------
('Accuracy:0.818', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.92      0.76      0.83        71\n          1
   0.73      0.90      0.80        50\n\navg / total       0.84      0.82      0.82       121\n', '\n')
Confusion Matrix
(array([[54, 17],
       [ 5, 45]]), '\n')
```

Figure 13: Result of Classification of Segmented Data with test-train split = 0.1

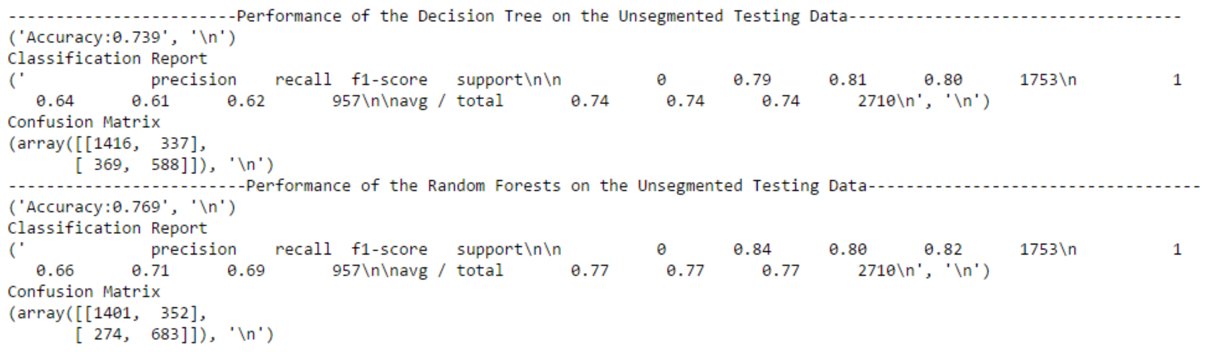
Even if the test-train split is increased to 0.33, random forest classifier continues be more accurate than the decision tree classifier, for the testing data, as shown by Figure 13.



```
-----------------------Performance of the Decision Tree on the Segmented Testing Data----------------------------------
('Accuracy:0.731', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.76      0.75      0.75       219\n          1
   0.70      0.71      0.70       179\n\navg / total       0.73      0.73      0.73       398\n', '\n')
Confusion Matrix
(array([[164,  55],
       [ 52, 127]]), '\n')
------------------------Performance of the Random Forests on the Segmented Testing Data----------------------------------
('Accuracy:0.744', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.79      0.72      0.76       219\n          1
   0.69      0.77      0.73       179\n\navg / total       0.75      0.74      0.74       398\n', '\n')
Confusion Matrix
(array([[158,  61],
       [ 41, 138]]), '\n')
```

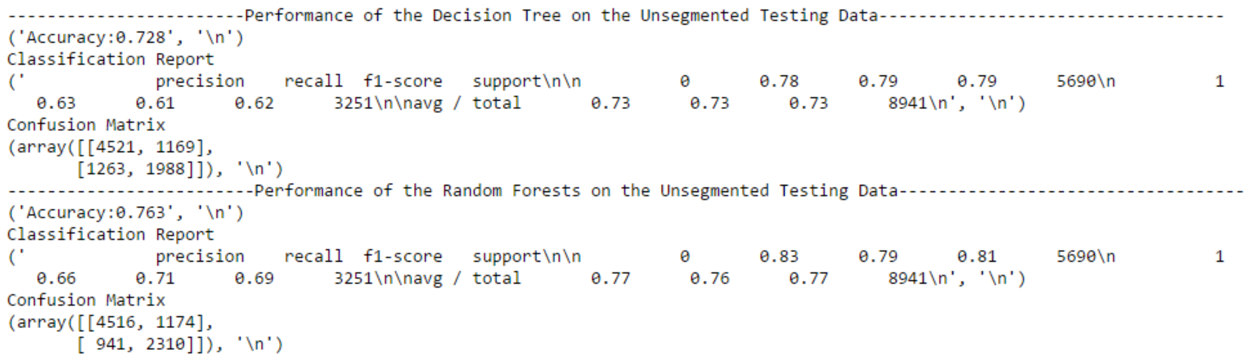
Figure 14: Result of Classification of Segmented Data with test-train split = 0.33

The system proposed by Reddy, Sasank et al. [1], does not use any explicit segmentation algorithm, to segment the data, instead use point by point classification, and combine the points belonging to the same mode of transportation in the post-processing. This point by point classification of modes was also implemented using the same Python Script, and the results are shown in Figure 14 and Figure 15.



```
-----------------------Performance of the Decision Tree on the Unsegmented Testing Data----------------------------------
('Accuracy:0.739', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.79      0.81      0.80      1753\n          1
   0.64      0.61      0.62       957\n\navg / total       0.74      0.74      0.74      2710\n', '\n')
Confusion Matrix
(array([[1416,  337],
       [ 369,  588]]), '\n')
------------------------Performance of the Random Forests on the Unsegmented Testing Data----------------------------------
('Accuracy:0.769', '\n')
Classification Report
('              precision    recall  f1-score   support\n\n          0       0.84      0.80      0.82      1753\n          1
   0.66      0.71      0.69       957\n\navg / total       0.77      0.77      0.77      2710\n', '\n')
Confusion Matrix
(array([[1401,  352],
       [ 274,  683]]), '\n')
```

Figure 15: Result of Classification of Un-segmented Data with test-train split = 0.1



```
-----------------------Performance of the Decision Tree on the Unsegmented Testing Data----------------------------------
('Accuracy:0.728', '\n')
Classification Report
('             precision    recall  f1-score   support\n\n          0       0.78      0.79      0.79      5690\n          1
   0.63      0.61      0.62      3251\n\navg / total       0.73      0.73      0.73      8941\n', '\n')
Confusion Matrix
(array([[4521, 1169],
       [1263, 1988]]), '\n')
------------------------Performance of the Random Forests on the Unsegmented Testing Data----------------------------------
('Accuracy:0.763', '\n')
Classification Report
('             precision    recall  f1-score   support\n\n          0       0.83      0.79      0.81      5690\n          1
   0.66      0.71      0.69      3251\n\navg / total       0.77      0.76      0.77      8941\n', '\n')
Confusion Matrix
(array([[4516, 1174],
       [ 941, 2310]]), '\n')
```

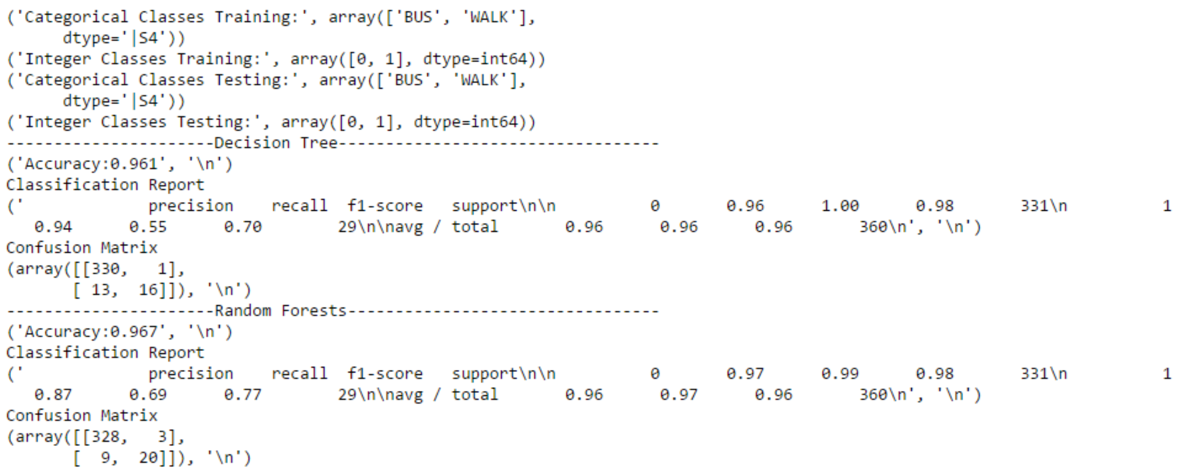
Figure 16: Result of Classification of Un-segmented Data with test-train split = 0.33

The results show that even for un-segmented data, random forest algorithm has a slightly higher accuracy than decision tree algorithm.

After classification, the decision tree and the individual trees created by the random forest algorithm were visualised using the pydot library, and these were imported onto a PDF file.

4.2.3 Experiment 3: Using Python script on trajectory data collected in Singapore for testing and the classifier trained in Experiment 2 (Section 4.2.2)

The 4 trajectories collected by a user travelling in Singapore, was broken down into segments using the four step segmentation algorithm proposed by Zheng et al. [2].
The 20 trajectories, in segmented form, used in Experiment 2 (Section 4.2.2), was used to train a decision tree and a random forest classifier.
The segments extracted from the 4 trajectories were used as testing data for the decision tree and random forest classifiers and the results are shown in Figure 16.



```
('Categorical Classes Training:', array(['BUS', 'WALK'],
      dtype='|S4'))
('Integer Classes Training:', array([0, 1], dtype=int64))
('Categorical Classes Testing:', array(['BUS', 'WALK'],
      dtype='|S4'))
('Integer Classes Testing:', array([0, 1], dtype=int64))
----------------------Decision Tree--------------------------------
('Accuracy:0.961', '\n')
Classification Report
('             precision    recall  f1-score   support\n\n          0       0.96      1.00      0.98       331\n          1
   0.94      0.55      0.70        29\n\navg / total       0.96      0.96      0.96       360\n', '\n')
Confusion Matrix
(array([[330,   1],
       [ 13,  16]]), '\n')
----------------------Random Forests-------------------------------
('Accuracy:0.967', '\n')
Classification Report
('             precision    recall  f1-score   support\n\n          0       0.97      0.99      0.98       331\n          1
   0.87      0.69      0.77        29\n\navg / total       0.96      0.97      0.96       360\n', '\n')
Confusion Matrix
(array([[328,   3],
       [  9,  20]]), '\n')
```

Figure 17: Result of Classification using the data collected in Singapore as the testing dataset

It is clearly seen the that the accuracy of the decision tree classifier is 96.1% and the random forest classifier is 96.7%, which is comparable to the accuracy of the classifiers used in solving similar problems in the state of the art. It is also evident that the random forest classifier has a slight advantage over decision tree classifier, so it agrees with the result obtained earlier.

4.3 SIGNIFICANCE OF THE RESULT OBTAINED

The project "Determination of Transportation Modes using Mobile Phones", tested various approaches of discerning between motorised and non-motorised mode of transportation, as proposed by various researchers. Through a series of experiments on various datasets, the most accurate classification method (i.e. random forest classifier) and the best features for classification (average velocity and average acceleration) have been found. Using this, an overall classification accuracy of 96.7% was obtained, for trajectory data collected in Singapore, using the classifier trained on selected trajectories of the GeoLife dataset. Therefore, there is a significant increase in accuracy from what has been obtained in the state-of-the-art so far (Table 4).

This significant increase in classification accuracy between motorised and non-motorised form of transport, can pave the way for further research in this field using this model for training a classifier, on different testing datasets collected across Singapore. This study can be extended by developing or testing existing algorithms in to distinguish further between different forms of motorised transport.

In the real world, this project can not only help the land transport authorities make the public transportation system more efficient, but can also help the police in law enforcement, if this system is made real time, i.e. a stream of GPS coordinates is transmitted on a continuous basis to a server, where the mode classification is performed.

4.4 DEVIAIONS FROM THE EXPECTED RESULTS AND ITS JUSTIFICATION

The 4 trajectories collected, by a volunteer taking multimodal trips across Singapore, contained the modes walk, waiting (stationary) and bus, whereas, the segmentation algorithm implemented is capable of segmenting only modes walk and bus. Therefore, the mode "Waiting" was included in the segment for "Walking", which might have caused some error in the results.
It is not possible to use the un-segmented form of the data collected in Singapore for testing purposes, since the training dataset does not have instances with mode as "Waiting".

4.5 CONCLUSION

The primary objective of the project ‘Determination of Transportation Modes using Mobile Phones’ was to verify different approaches proposed by various researchers to differentiate between the different modes of transportation (Motorised and Non-Motorised) being used by a person. This is being done through a series of experiments, which use various software tools on different datasets. Using Weka Machine Learning Tool, velocity and acceleration were found to be the best features for classification and the decision tree was found to be the most effective classifier, with an accuracy of 74.94% on some transportation data of a user collected across Singapore.

The drawback of the Weka Machine Learning Tool is that the classified instances are not accessible to the user, which is needed in this project for post-processing. Therefore, the Sci-kit learn toolkit of Python was used to write a script to perform classification using the decision tree and random forest algorithm. Selected trajectories of the GeoLife dataset was segmented using the change point segmentation algorithm proposed by Zheng et al. [2,3,12], and then was classified using the decision tree and random forest algorithm. The result of this was compared with the work of Reddy et al. [1], where the classification between motorised and non-motorised modes of transportation was done on a point to point basis, instead of a segment basis. It was found that random forest algorithm, performed slightly better than decision tree algorithm on both segmented and un-segmented data. The overall accuracy of the random forest algorithm on segmented data was 81.8%, whereas the overall accuracy of the random forest algorithm on un-segmented data was 76.9%. Therefore, it is clearly evident that segmentation of the dataset improved the accuracy of classification.

This work was taken forward by using the trajectories selected from the GeoLife dataset for training the decision tree and random forest classifiers, and testing the classifier on the data collected by a user taking multimodal trips across Singapore. The decision tree and random forest classifiers on the segmented dataset gave an accuracy of 96.1% and 96.7%, respectively, which is higher than what has been obtained in the state of the art so far.

Therefore, the change point segmentation and classification algorithms are highly suited to build a solution to the problem of discerning between motorised and non-motorised modes of transport, using a series of time-stamped GPS locations.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE OF WORK

5.1 SUMMARY

5.1.1 Objectives

The following are the objectives of this project:

- ➢ To infer the mode of transportation (motorised or non-motorised) in compound trips, which contain more than one kind of transportation modes, using only GPS traces and independent of other information from maps and other sensors.
- ➢ To correctly detect the transition between the different transportation modes.
- ➢ To apply the model learned from the dataset of some users, on the GPS traces of other users to infer their mode of transportation.

5.1.2 Work Methodology

A detailed study of the state of the art methods for performing transportation mode classification was performed and based on that, the proposed system was designed.

The following are the steps followed in this project:

1) Data collection / Dataset selection
2) Feature selection
3) Segmentation
4) Classification

The raw data is time sequenced location data retrieved from the GPS in the cell phones of the users. This data was pre-processed in order to extract relevant features like velocity, acceleration and heading change. The dataset also contained the ground truth of the mode of transportation being used by them for the time period they reported the data for. The dataset was pre-processed using MS Excel, in order to remove redundant data and filter out sections of the trajectory which were discontinuous.

A change point segmentation algorithm based on the work by Zheng et. al. [2,3], where, the data was first broken down into segments using a loose bound of velocity and acceleration, segments smaller than a certain threshold were merged into the backward segment and consecutive segments of length smaller than another threshold were merged to form one segment was implemented using a Python Script.

For every segment retrieved from the change point segmentation algorithm, the average velocity, the average acceleration and the average heading change has been calculated, and have been used as a feature for classification.

The Decision Tree classifier has been used as a classification algorithm, as it was found to be the most accurate algorithm for mode classification according to the state of the art [1]. Decision Trees are a non-parametric supervised learning method used for classification. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

5.1.3 Result Summary

Table 5 gives a summary of the results obtained in this project, on different datasets.

Table 4: Summary of Results

	Segmented	**Un-segmented**
GeoLife Dataset (selected trajectories)		
	test-train-split = 0.33	
Decision Tree	73.1%	72.8%
Random Forest	74.4%	76.3%
	test-train-split = 0.1	
Decision Tree	76%	73.9%
Random Forest	81.8%	76.9%
Transportation Data Collected in Singapore		
Decision Tree	96.1%	-
Random Forest	96.8%	-

5.2 CONCLUSION

The primary objective of the project 'Determination of Transportation Modes using Mobile Phones' was to verify different approaches proposed by various researchers to differentiate between the different modes of transportation (Motorised and Non-Motorised) being used by a person.

Transportation mode detection is an emerging field of study. Several algorithms for segmentation of trips and mode classification have been proposed in this state of the art, and they have been tested in different cities all across the world. However, no significant models have been built, specifically for Singapore which can detect and analyse mode of transportation properly. Cottrill et al. [31] and Xiao et al. [32] have attempted to develop activity monitoring models for Singapore, by collecting mobility data across Singapore, but they have not tested a transportation mode detection system using this dataset.

The current work, uses selected trajectories of the GeoLife dataset [2 ,3, 12], to train a decision tree based model, and uses GPS trajectory data collected from a user taking multimodal trips across Singapore, in order to test the model. These improved methodology gave overall accuracy of about 96.7% using the random forest classifier, on the data collected across Singapore, which is higher than what has been obtained in the state the art.

A detailed study has been done on the architecture of the transportation mode detection system. Various tools were used to test different approaches, proposed in the state of the art, on the data collected in Singapore and selected trajectories of the GeoLife dataset.

Series of experiments have been done in the current work which use various software tools on different datasets. Using Weka Machine Learning Tool, velocity and acceleration were found to be the best features for classification and the decision tree was found to be the most effective classifier, with an accuracy of 74.94% on some transportation data of a user collected across Singapore.

The drawback of the Weka Machine Learning Tool is that the classified instances are not accessible to the user, which is needed in this project for post-processing. Therefore, the Sci-kit learn toolkit of Python was used to write a script to perform classification using the decision tree and random forest algorithm. Selected trajectories of the GeoLife dataset was segmented using the change point segmentation algorithm proposed by Zheng et al. [2,3,12], and then was classified using the decision tree and random forest algorithm. The result of this was compared with the work of Reddy et al. [1], where the classification between motorised and non-motorised modes of transportation was done on a point to point basis, instead of a segment basis. It was found that random forest algorithm, performed slightly better than decision tree algorithm on both segmented and un-segmented data. The overall accuracy of the random forest algorithm on segmented data was 81.8%, whereas the overall accuracy of the random forest algorithm on un-segmented data was 76.9%. Therefore, it is clearly evident that segmentation of the dataset improved the accuracy of classification.

This work was taken forward by using the trajectories selected from the GeoLife dataset for training the decision tree and random forest classifiers, and testing the classifier on the data collected by a user taking multimodal trips across Singapore. The decision tree and random forest classifiers on the segmented dataset gave an accuracy of 96.1% and 96.7%, respectively, which is higher than what has been obtained in the state of the art so far.

Therefore, the change point segmentation and classification algorithms are highly suited to build a solution to the problem of discerning between motorised and non-motorised modes of transport, using a series of time-stamped GPS locations.

5.3 FUTURE SCOPE OF THE WORK

The following work is in the ongoing stage:

- Improving the segmentation algorithm to accommodate more than two transportation modes.
- Improving the accuracy of the Decision Tree classifier by improving the change point segmentation algorithm.
- GPS trajectories of more volunteers taking multimodal, multi-leg, public transport trips in Singapore are being collected for further experiments
- Incorporating high level information like the location of the bus stops and traffic signals across Singapore, in order to ascertain the modes being indicated by the classification algorithm.
- Making the system capable of handling real-time stream of data.

This work can be further extended by determining the exact route taken by the person, by comparing the trajectories with the map of the various bus routes.

REFERENCES

[1]. Reddy, Sasank, et al. "Using mobile phones to determine transportation modes." *ACM Transactions on Sensor Networks (TOSN)* 6.2 (2010): 13.

[2]. Zheng, Yu, et al. "Understanding mobility based on GPS data." *Proceedings of the 10th international conference on Ubiquitous computing*. ACM, 2008.

[3]. Zheng, Yu, et al. "Learning transportation mode from raw GPS data for geographic applications on the web." *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008.

[4]. Jonsson, Fredrik. "Determining transportation mode in mobile phones using human agent movements." *USCCS 2010* (2010): 73.

[5]. Stenneth, Leon, et al. "Transportation mode detection using mobile phones and GIS information." *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2011.

[6]. Gong, Hongmian, et al. "A GPS/GIS method for travel mode detection in New York City." *Computers, Environment and Urban Systems* 36.2 (2012): 131-139.

[7]. Witayangkurn, Apichon, et al. "Trip reconstruction and transportation mode extraction on low data rate GPS data from mobile phone." *Proceedings of the international conference on computers in urban planning and urban management (CUPUM 2013)*. 2013.

[8]. Liao, Lin, Dieter Fox, and Henry Kautz. "Extracting places and activities from gps traces using hierarchical conditional random fields." *The International Journal of Robotics Research* 26.1 (2007): 119-134.

[9]. Chen, Zhiyuan, Bahran Sanjabi, and Dino Isa. "A location-based user movement prediction approach for Geolife project." *Int. J. Comput. Eng. Res* 2.7 (2012): 16-19.

[10]. Mun, M., et al. "Parsimonious mobility classification using GSM and WiFi traces." *Proceedings of the Fifth Workshop on Embedded Networked Sensors (HotEmNets)*. 2008.

[11]. Biljecki, Filip, Hugo Ledoux, and Peter Van Oosterom. "Transportation mode-based segmentation and classification of movement trajectories." *International Journal of Geographical Information Science* 27.2 (2013): 385-407.

[12]. Zheng, Yu, et al. "Mining interesting locations and travel sequences from GPS trajectories." *Proceedings of the 18th international conference on World Wide Web*. ACM, 2009.

[13]. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[14]. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten (2009). The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1.

[15]. H. Zhang (2004). “The optimality of Naive Bayes”. Proc. FLAIRS

[16]. L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.

[17]. J.R. Quinlan. C4. 5: programs for machine learning. Morgan Kaufmann, 1993.

[18]. T. Hastie, R. Tibshirani and J. Friedman. Elements of Statistical Learning, Springer, 2009

[19]. Anderson, Ian, and Henk Muller. "Practical Activity Recognition using GSM Data∗." (2006).

[20]. Liao, Lin, et al. "Learning and inferring transportation routines." *Artificial Intelligence* 171.5-6 (2007): 311-331.

[21]. Patterson, Donald J., et al. "Inferring high-level behavior from low-level sensors." *International Conference on Ubiquitous Computing*. Springer Berlin Heidelberg, 2003.

[22]. Zhang, Lijuan, et al. "Multi-stage approach to travel-mode segmentation and classification of gps traces." *Proceedings of the ISPRS Guilin 2011 Workshop on International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Guilin, China*. Vol. 2021. 2011.

[23]. Chung, Eui-Hwan, and Amer Shalaby. "A trip reconstruction tool for GPS-based personal travel surveys." *Transportation Planning and Technology* 28.5 (2005): 381-401.

[24]. Bohte, Wendy, 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 (2009): 285-297.

[25]. Bohte, Wendy, Kees Maat, and Wilko Quak. "A method for deriving trip destinations and modes for GPS-based travel surveys." *Research in Urbanism Series* 1.1 (2008): 127-143.

[26]. Stopher, Peter, et al. "Deducing mode and purpose from GPS data." *Institute of Transport and Logistics Studies* (2008): 1-13.

[27]. Papliatseyeu, Andrei, and Oscar Mayora. "Mobile habits: Inferring and predicting user activities with a location-aware smartphone." *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*. Springer Berlin/Heidelberg, 2009.

[28]. Draijer, Geert, Nelly Kalfs, and Jan Perdok. "GPS as a Data Collection Method for Travel Research: The use of GPS for data collection for all modes of travel." *Transportation Research Board 79th Annual Meeting*. 2000.

[29]. Schuessler, Nadine, and Kay Axhausen. "Processing raw data from global positioning systems without additional information." *Transportation Research Record: Journal of the Transportation Research Board* 2105 (2009): 28-36.

[30]. Biljecki, Filip, et al. *Automatic segmentation and classification of movement trajectories for transportation modes*. Association of Geographic Information Laboratories Europe, 2012.

[31]. Cottrill, Caitlin, Francisco Pereira, Fang Zhao, Ines Dias, Hock Lim, Moshe Ben-Akiva, and P. Zegras. “Future Mobility Survey.” Transportation Research Record: Journal of the Transportation Research Board 2354 (December 2013): 59–67.

[32]. Xiao, Yu, et al. "Transportation activity analysis using smartphones." *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. IEEE, 2012.

[33]. Mohan, Prashanth, Venkata N. Padmanabhan, and Ramachandran Ramjee. "Nericell: rich monitoring of road and traffic conditions using mobile smartphones." *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008.

[34]. Frendberg, Maria. *Determining transportation mode through cellphone sensor fusion*. Diss. MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2011.

[35]. Hemminki, Samuli, Petteri Nurmi, and Sasu Tarkoma. "Accelerometer-based transportation mode detection on smartphones." *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2013.

[36]. Thiagarajan, Arvind et al. “VTrack: accurate, energy-aware road traffic delay estimation using mobile phones.” Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. Berkeley, California: ACM, 2009. 85-98.

[37]. Lin, Miao, Wen-Jing Hsu, and Zhuo Qi Lee. "Detecting modes of transport from unlabelled positioning sensor data." *Journal of Location Based Services* 7.4 (2013): 272-290.

ANNEXURE 1: User Guide of GeoLife Dataset

User Guide

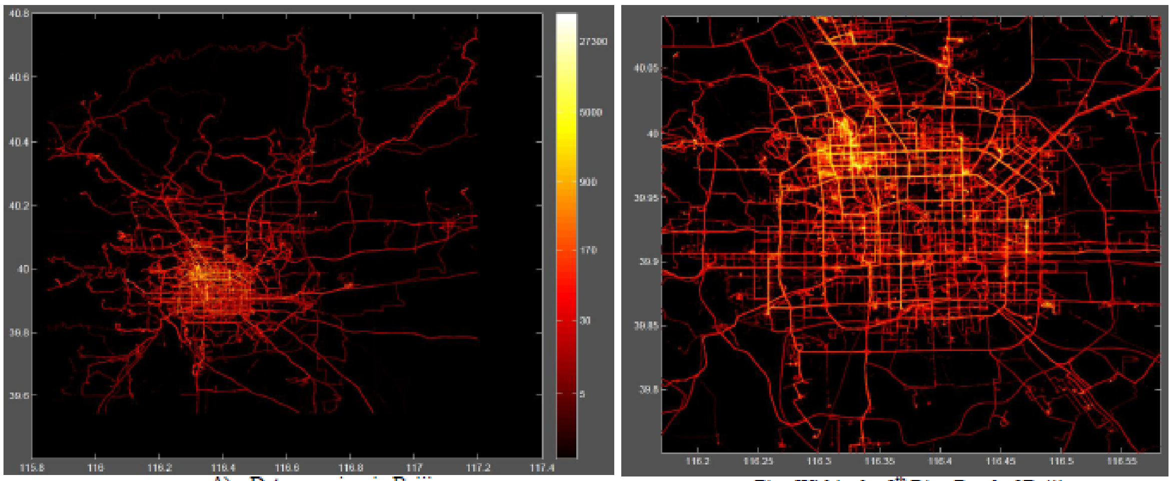
Version 1.2 (2011/10/31)

1. Data Description

This GPS trajectory dataset was collected in (Microsoft Research Asia) Geolife project by 178 users in a period of over four years (from April 2007 to October 2011). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. This dataset contains 17,621 trajectories with a total distance of 1,251,654 kilometers and a total duration of 48,203 hours. These trajectories were recorded by different GPS loggers and GPS-phones, and have a variety of sampling rates. 91 percent of the trajectories are logged in a dense representation, e.g. every 1~5 seconds or every 5~10 meters per point.

This dataset recoded a broad range of users' outdoor movements, including not only life routines like go home and go to work but also some entertainments and sports activities, such as shopping, sightseeing, dining, hiking, and cycling. This trajectory dataset can be used in many research fields, such as mobility pattern mining, user activity recognition, location-based social networks, location privacy, and location recommendation.

Although this dataset is wildly distributed in over 30 cities of China and even in some cities located in the USA and Europe, the majority of the data was created in Beijing, China. Figure 1 plots the distribution (heat map) of this dataset in Beijing. The figures standing on the right side of the heat bar denote the number of points generated in a location.

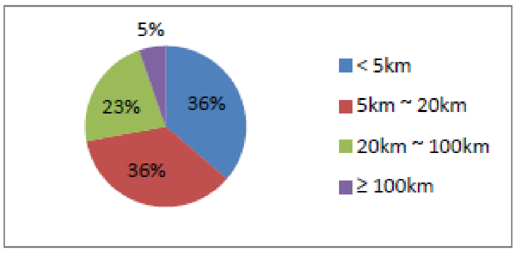


A) Data overview in Beijing B) Within the 5th Ring Road of Beijing

Figure 1 Distribution of the dataset in Beijing city

The distributions of distance and duration of the trajectories are presented in Figure 2 and Figure 3.

In the data collection program, a portion of users have carried a GPS logger for years, while some of the others only have a trajectory dataset of a few weeks. This distribution is presented in Figure 4, and the distribution of the number of trajectories collected by each user is shown in Figure 5.



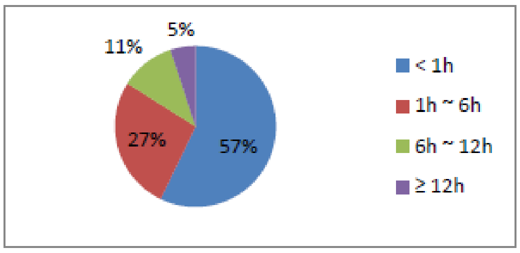


Figure 2 Distribution of trajectories by distance **Figure 3 Distribution of trajectories by effective duration**

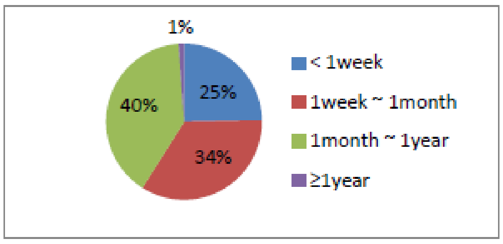


Figure 4 Distribution of users by data collection period

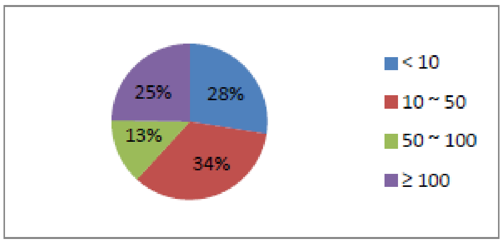


Figure 5 Distribution of users by trajectories

2. What's new?

2.1. Transportation mode labels

69 users have labeled their trajectories with **transportation mode**, such as driving, taking a bus, riding a bike and walking. There is a label file storing the transportation mode labels in each user's folder. See section 4.2 for the format of labels.

The total distance and duration of transportation modes are listed in Figure 6. Though this only covers a part of the dataset used in the following papers, the scale of this released dataset can still support transportation mode learning.

Transportation mode	Distance (km)	Duration (hour)
Walk	10,092	5,436
Bike	6,244	2,352
Bus	20,230	1,492
Car & taxi	32,848	2,383
Train	36,253	745
Airplane	24,789	40
Other	9,493	404
Total	139,953	12,856

Figure 6 Total distance and duration of transportation modes

2.2. Changes on the scale of dataset

Changes on the scale of dataset between version 1.1 and version 1.2 are listed in Figure 7. Effective days refer to the total number of days where there's a record in the dataset.

	Version 1.1	Version 1.2	Change
Time span of the collection	04/2007 – 12/2010	04/2007 – 10/2011	+10 months
Number of users	169	178	+9
Number of trajectories	16,550	17,621	+1,071
Number of points	21,363,084	23,667,828	+2,304,744
Total distance	1,212,451 km	1,251,654km	+39,203 km
Total duration	46,065 hour	48,203hour	+2,138 hour
Effective days	9,694	10,413	+719

Figure 7 Changes on the scale of dataset

3. Paper Citation

Please cite the following papers when using this GPS dataset.

[1] Yu Zheng, Lizhu Zhang, Xing Xie, Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In Proceedings of International conference on World Wild Web (WWW 2009), Madrid Spain. ACM Press: 791-800.

[2] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, Wei-Ying Ma. Understanding Mobility Based on GPS Data. In Proceedings of ACM conference on Ubiquitous Computing (UbiComp 2008), Seoul, Korea. ACM Press: 312-321.

[3] Yu Zheng, Xing Xie, Wei-Ying Ma, GeoLife: A Collaborative Social Networking Service among User, location and trajectory. Invited paper, in IEEE Data Engineering Bulletin. 33, 2, 2010, pp. 32-40.

4. Data Format

4.1. Trajectory file

Every single folder of this dataset stores a user's GPS log files, which were converted to PLT format. Each PLT file contains a single trajectory and is named by its starting time. To avoid potential confusion of time zone, we use **GMT** in the date/time property of each point, which is different from our previous release.

PLT format:

Line 1…6 are useless in this dataset, and can be ignored. Points are described in following lines, one for each line.

Field 1: Latitude in decimal degrees.

Field 2: Longitude in decimal degrees.

Field 3: All set to 0 for this dataset.

Field 4: Altitude in feet (-777 if not valid).

Field 5: Date - number of days (with fractional part) that have passed since 12/30/1899.

Field 6: Date as a string.

Field 7: Time as a string.

Note that field 5 and field 6&7 represent the same date/time in this dataset. You may use either of them.

Example:

```
39.906631,116.385564,0,492,40097.5864583333,2009-10-11,14:04:30
39.906554,116.385625,0,492,40097.5865162037,2009-10-11,14:04:35
```

4.2. Transportation mode labels

Possible transportation modes are: *walk*, *bike*, *bus*, *car*, subway, *train*, *airplane*, *boat*, *run* and *motorcycle*. Again, we have converted the date/time of all labels to **GMT**, even though most of them were created in China.

Example:

Start Time	End Time	Transportation Mode
2008/04/02 11:24:21	2008/04/02 11:50:45	bus
2008/04/03 01:07:03	2008/04/03 11:31:55	train
2008/04/03 11:32:24	2008/04/03 11:46:14	walk
2008/04/03 11:47:14	2008/04/03 11:55:07	car

First, you can regard the label of both *taxi* and *car* as *driving* although we set them with different labels for future usage. Second, a user could label the transportation mode of a *light rail* as *train* while others may use *subway* as the label. Actually, no trajectory can be recorded in an underground subway system since a GPS logger cannot receive any signal there. In Beijing, the light rails and subway systems are seamlessly connected, e.g., line 13 (a light rail) is connected with line 10 and line 2, which are subway systems. Sometimes, a line (like line 5) is comprised of partial subways and partial light rails. So, users may have a variety of understanding in their transportation modes. You can differentiate the real train trajectories (connecting two cities) from the light rail trajectory (generating in a city) according to their distances. Or, just treat them the same.

5. Contact

If you have any questions about this dataset, please contact Dr. Yu Zheng from Microsoft Research Asia.

Yu Zheng

Tel: 86-10-59173038 Email: yuzheng@microsoft.com

Homepage: http://research.microsoft.com/en-us/people/yuzheng/default.aspx

ANNEXURE 2: Implementing Decision Tree and Random Forests using Sci-kit Learn Library of Python

2.1 sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(*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_split=1e-07*, *class_weight=None*, *presort=False*)

Parameters:

criterion : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : string, optional (default="best")

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_features : int, float, string or None, optional (default=None)

The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.
- If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features).
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_featuresfeatures.

max_depth : int or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int, float, optional (default=2)

The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Changed in version 0.18: Added float values for percentages.

min_samples_leaf : int, float, optional (default=1)

The minimum number of samples required to be at a leaf node:

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for percentages.

min_weight_fraction_leaf : float, optional (default=0.)

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_leaf_nodes : int or None, optional (default=None)

Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

class_weight : dict, list of dicts, "balanced" or None, optional (default=None)

Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

random_state : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

min_impurity_split : float, optional (default=1e-7)

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

New in version 0.18.

presort : bool, optional (default=False)

Whether to presort the data to speed up the finding of best splits in fitting. For the default settings of a decision tree on large datasets, setting this to true may slow down the training process. When using either a smaller dataset or a restricted depth, this may speed up the training.

2.2 sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.RandomForestClassifier(*n_estimators=10*, *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_split=1e-07*, *bootstrap=True*, *oob_score=False*, *n_jobs=1*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*)

Parameters:

n_estimators : integer, optional (default=10)

The number of trees in the forest.

criterion : string, optional (default=”gini”)

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this parameter is tree-specific.

max_features : int, float, string or None, optional (default=”auto”)

The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.
- If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
- If “auto”, then max_features=sqrt(n_features).
- If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).
- If “log2”, then max_features=log2(n_features).
- If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_featuresfeatures.

max_depth : integer or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int, float, optional (default=2)

The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Changed in version 0.18: Added float values for percentages.

min_samples_leaf : int, float, optional (default=1)

The minimum number of samples required to be at a leaf node:

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for percentages.

min_weight_fraction_leaf : float, optional (default=0.)

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_leaf_nodes : int or None, optional (default=None)

Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_split : float, optional (default=1e-7)

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

New in version 0.18.

bootstrap : boolean, optional (default=True)

Whether bootstrap samples are used when building trees.

oob_score : bool (default=False)

Whether to use out-of-bag samples to estimate the generalization accuracy.

n_jobs : integer, optional (default=1)

The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.

random_state : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

verbose : int, optional (default=0)

Controls the verbosity of the tree building process.

warm_start : bool, optional (default=False)

When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

class_weight : dict, list of dicts, “balanced”,

“balanced_subsample” or None, optional (default=None) Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data
as n_samples / (n_classes * np.bincount(y))

The “balanced_subsample” mode is the same as “balanced” except that weights are computed based on the bootstrap sample for every tree grown.

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

ANNEXURE 3: Python Implementation of the Problem using test-train-split on Selected Trajectories of the Geo-Life Dataset

```
import math
import numpy as np
import scipy as sc
import pandas as pd
import xlwt
import csv

####################################################################################################
################################# Change Point Segmentation #########################################
####################################################################################################

############################################### Step 1 ##############################################

book = xlwt.Workbook(encoding="utf-8")
sheet1 = book.add_sheet("Sheet 1")
sheet1.write(0,0,"Latitude")
sheet1.write(0,1,"Longitude")
sheet1.write(0,2,"Average Velocity")
sheet1.write(0,3,"Average Acceleration")
sheet1.write(0,4,"Mode")
df = pd.read_csv('C:\\Users\\N1600060B\\Documents\\Transport Mode Detection\\Final\\test.csv',sep=',',header=None)
#ff=open('dumpfile.csv','w+')
data = df.values
#print data
# data = datat.tolist()
#print len(data[:,6])
cValues = np.where((data[:,6])>1.6)[0]
#print len(cValues)
segment = data[3:cValues[0]+1,:]
#print segment
segments = [segment]
#print segments
start = 4
end = -1
for i in range(3,len(cValues)):
    if cValues[i]!=cValues[i-1]+1:
        end = cValues[i-1]+1
        seg1 = data[start:end,:]
        start2 = cValues[i-1]+1
        end2 = cValues[i]
        seg2 = data[start2:end2,:]
        if len(seg1)!=0:
            #print len(seg1)
            segments.append(seg1)
        if len(seg2)!=0:
            segments.append(seg2)
        #print len(seg2)
      start = cValues[i]
#print segments
#print len(segments)

############################################## Step 2 ##############################################

newseglist = []
newseglist.append(segments[0])
mergeIndex=0
for ii in range(1,len(segments)):
   if len(segments[ii])<=3:
      newseglist[mergeIndex] = np.concatenate((newseglist[mergeIndex],segments[ii]),axis=0)
   else:
      newseglist.append(segments[ii])
      mergeIndex = mergeIndex+1
#-------------------------------debugging-----------------------------------#
# print len(newseglist)
# for jj in range(0,len(newseglist)):
#   print len(newseglist[jj])
# print newseglist
#-----------------------------end of debugging------------------------------#

############################################## Step 3 ##############################################

seglistReborn = []
mergeMode = 0
mergeIndex = 0
for j in range(0,len(newseglist)):
   if len(newseglist[j])>10:
      seglistReborn.append(newseglist[j])
      mergeMode=0
      seglistBorn = [seglistReborn]
   elif len(newseglist[j])<=10 and mergeMode==0:
      seglistReborn.append(newseglist[j])
      mergeIndex = len(seglistReborn)-1
      mergeMode=1
      seglistBorn = [seglistReborn]
   elif len(newseglist[j])<=10 and mergeMode==1:
      seglistReborn[mergeIndex] = np.concatenate((seglistReborn[mergeIndex],newseglist[j]),axis=0)
      seglistBorn = [seglistReborn]
   #print seglistBorn[0]
del newseglist
#-------------------------------debugging-----------------------------------#
#print len(seglistReborn)
#for jj in range(0,len(seglistReborn)):
#     print len(seglistReborn[jj])
#print seglistReborn[1][0][0]
#-----------------------------end of debugging------------------------------#

##################################### Step 4 / Calculating Features #######################################
array_segment=[]
meanData = np.sum(seglistReborn[0][:,[6,7]],axis=0)/len(seglistReborn[0])
#print meanData
```

```
for i in range(1,(len(seglistBorn[0]))):
    #print seglistBorn[0][i]
    if len(seglistBorn[0][i]) is not 0:
        meanVelocity = sum(seglistBorn[0][i][:,[6]])/len(seglistBorn[0][i])
        meanAcceleration = sum(seglistBorn[0][i][:,[7]])/len(seglistBorn[0][i])
    #print "Length of Segment = "
    #print len(seglistReborn[i])
    #print "Start Lattitude = "
    #print seglistReborn[i][0][0]
    #print "Start Longitude = "
    #print seglistReborn[i][0][1]
    #print "End Lattitude = "
    #print seglistReborn[i+1][0][0]
    #print "End Longitude = "
    #print seglistReborn[i+1][0][1]
    #print "Mode = "
    #print seglistReborn[i][0][8]
    #print " Features = "
    #print np.sum(seglistReborn[i][:,[6,7]],axis=0)/len(seglistReborn[i])
    #print "--------------------------------------"
        #print len(seglistBorn[0][i])
        sheet1.write(i,0,seglistReborn[i][0][0])
        sheet1.write(i,1,seglistReborn[i][0][1])
        sheet1.write(i,2,meanVelocity[0])
        sheet1.write(i,3,meanAcceleration[0])
        sheet1.write(i,4,seglistReborn[i][0][8])
book.save("segments_train_1.csv")

########################################################################################################################
#################################### Classification of Segmented Data #############################################
#########################################################################################################################

print "############################ Classification of Segmented Data #################################################"

with open('C:\\Users\\N1600060B\\segments_data.csv','r') as csvfile:
    mode_reader = csv.reader(csvfile, delimiter=',', quotechar='"')

############################################### Header contains feature names ##########################################

    row=next(mode_reader)
    feature_names=np.array(row)
    #print feature_names

################################################### Load dataset and target classes #######################################

    mode_X,mode_y=[],[]
    for row in mode_reader:
        mode_X.append(row)
        mode_y.append(row[4])
    mode_X = np.array(mode_X)
    mode_y = np.array(mode_y)
    #print mode_X
    #print mode_y
```

```
######################################## Retain required columns ####################################

    mode_X = mode_X[:, [2,3]]
    feature_names = feature_names[[2,3]]
    #print feature_names
    #print mode_X

########################################### Encode mode ###########################################

    from sklearn.preprocessing import LabelEncoder
    enc = LabelEncoder()
    label_encoder = enc.fit(mode_y[:])
    print("Categorical Classes:", label_encoder.classes_)
    integer_classes = label_encoder.transform(label_encoder.classes_)
    print("Integer Classes:", integer_classes)
    t = label_encoder.transform(mode_y[:])
    mode_y[:] = t
    #print (mode_X[4], mode_y[4])

################################### Separate training and test sets ####################################

    from sklearn.cross_validation import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(mode_X,
                                        mode_y,
                                        test_size=0.1,
                                        random_state=30,
                                        stratify=None)
    #print X_test

##################################### Decision tree #################################################

    from sklearn import tree
    clf = tree.DecisionTreeClassifier(criterion='entropy',
                                splitter='best',
                                max_depth=30,
                                min_impurity_split=1e-09,
                                presort = True,
                                min_samples_leaf=3,
                                min_samples_split=20)
    clf = clf.fit(X_train,y_train)
    #print clf

###################################### Random Forests ##############################################

    from sklearn.ensemble import RandomForestClassifier
    clf_rf_train = RandomForestClassifier(n_jobs=3,
                                criterion='entropy',
                                max_depth=30,
                                min_samples_split=20,
                                min_samples_leaf=5,
                                verbose=0,
                                warm_start=False)
    clf_rf_train = clf_rf_train.fit(X_train,y_train)
```

```
############################### Measure the accuracy of the decision tree ####################################

  from sklearn import metrics

  def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True):
      y_pred=clf.predict(X)
      if show_accuracy:
          print("Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n")
      if show_classification_report:
          print("Classification Report")
          print(metrics.classification_report(y,y_pred),"\n")
      if show_confusion_matrix:
          print("Confusion Matrix")
          print(metrics.confusion_matrix(y,y_pred),"\n")

  print "--------------------Performance of the Decision Tree on the Segmented Training Data---------------------------------"
  measure_performance(X_train,y_train,clf, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True)

  print "--------------------Performance of the Random Forest on the Segmented Training Data----------------------------------"
  measure_performance(X_train,y_train,clf_rf_train, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True)

################## Evaluating the performance of the decision tree on the testing data ############################

  clf_dt = tree.DecisionTreeClassifier(criterion='entropy',
                                splitter='best',
                                min_impurity_split=1e-09,
                                max_depth=30,
                                presort = True,
                                min_samples_leaf=3,
                                min_samples_split=20)
  clf_dt.fit(X_train, y_train)
  x = clf_dt.predict(X_test)
  #for i in range(1,(len(X_test))):
  #    print x[i]
  #    print y_test[i]

  print "------------------------Performance of the Decision Tree on the Segmented Testing Data-----------------------------------"
  measure_performance(X_test,y_test,clf_dt)

################# Evaluating the performance of the random forest on the testing data ###########################

  clf_rf_test = RandomForestClassifier(n_jobs=3,
                                criterion='entropy',
                                max_depth=30,
                                min_samples_split=20,
                                min_samples_leaf=5,
                                verbose=0,
                                warm_start=False)
  clf_rf_test = clf_rf_test.fit(X_train,y_train)
  y = clf_rf_test.predict(X_test)
```

```
    print "------------------------Performance of the Random Forests on the Segmented Testing Data--------------------------------
--"
    measure_performance(X_test,y_test,clf_rf_test)


################################ Exporting results to a CSV File ########################################

#print len(X_test)
book_result = xlwt.Workbook(encoding="utf-8")
sheet2 = book.add_sheet("Sheet 2")
sheet2.write(0,0,"Average Velocity")
sheet2.write(0,1,"Average Acceleration")
sheet2.write(0,2,"Mode - Ground Truth")
sheet2.write(0,3,"Mode - Predicted")
for i in range(1,(len(X_test))):
    sheet2.write(i,0,X_test[i][0])
    sheet2.write(i,1,X_test[i][1])
    sheet2.write(i,2,y_test[i])
    sheet2.write(i,3,x[i])
#book_result.save("Test_Results.csv")

##################################### Data visualisation ############################################

import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
from sklearn.externals.six import StringIO
import pydot
dot_data = StringIO()
tree.export_graphviz(clf,
                    out_file='tree_dt.dot')

#tree.export_graphviz(clf_rf_train,
#                    out_file='tree_rf.dot')

i_tree = 0
for tree_in_forest in clf_rf_train.estimators_:
    with open('tree_' + str(i_tree) + '.dot', 'w') as my_file:
        my_file = tree.export_graphviz(tree_in_forest, out_file = my_file)
        #(graph_rf,) = pydot.graph_from_dot_file('tree_' + str(i_tree) + '.dot')
        #graph_rf.write_pdf('randomForests.pdf')
    i_tree = i_tree + 1

#graph = pydot.graph_from_dot_data(dot_data.getvalue())
#graph.write_pdf("mode.pdf")
#from subprocess import check_call
#check_call(['dot','-Tpng','tree.dot','-o','OutputFile.png'])

(graph_dt,) = pydot.graph_from_dot_file('tree_dt.dot')
graph_dt.write_pdf('dectree.pdf')

(graph_rf_1,) = pydot.graph_from_dot_file('tree_1.dot')
graph_rf_1.write_pdf('randomForests_1.pdf')

(graph_rf_2,) = pydot.graph_from_dot_file('tree_2.dot')
```

```
graph_rf_2.write_pdf('randomForests_2.pdf')

(graph_rf_3,) = pydot.graph_from_dot_file('tree_3.dot')
graph_rf_3.write_pdf('randomForests_3.pdf')

####################################################################################################
################################## Classification of Unsegmented Data ####################################
####################################################################################################

print "########################## Classification of Unsegmented Data ######################################"

with open('C:\\Users\\N1600060B\\Documents\\Transport Mode Detection\\Final\\test.csv','r') as csvfile_ns:
  mode_reader_ns = csv.reader(csvfile_ns, delimiter=',', quotechar='"')

################################## Header contains feature names #######################################

  row_ns=next(mode_reader_ns)
  feature_names_ns=np.array(row_ns)
  #print feature_names_ns

#################################### Load dataset and target classes ######################################

  mode_X_ns,mode_y_ns=[],[]
  for row_ns in mode_reader_ns:
    mode_X_ns.append(row_ns)
    mode_y_ns.append(row_ns[8])
  mode_X_ns = np.array(mode_X_ns)
  mode_y_ns = np.array(mode_y_ns)
  #print mode_X_ns
  #print mode_y_ns

################################### Retain required columns ##########################################

  mode_X_ns = mode_X_ns[:, [6,7]]
  feature_names_ns = feature_names_ns[[6,7]]
  #print feature_names_ns
  #print mode_X

############################################## Encode mode ##########################################

  from sklearn.preprocessing import LabelEncoder
  enc_ns = LabelEncoder()
  label_encoder_ns = enc_ns.fit(mode_y_ns[:])
  print("Categorical Classes:", label_encoder_ns.classes_)
  integer_classes_ns = label_encoder_ns.transform(label_encoder_ns.classes_)
  print("Integer Classes:", integer_classes_ns)
  t_ns = label_encoder_ns.transform(mode_y_ns[:])
  mode_y_ns[:] = t_ns
  #print (mode_X[4], mode_y[4])

##################################### Separate training and test sets ####################################

  from sklearn.cross_validation import train_test_split
  X_train_ns, X_test_ns, y_train_ns, y_test_ns = train_test_split(mode_X_ns,
                                                    mode_y_ns,
                                                    test_size=0.1,
                                                    random_state=30,
                                                    stratify=None)
  #print X_test

########################################## Decision tree ##############################################

  from sklearn import tree
  clf_ns = tree.DecisionTreeClassifier(criterion='entropy',
                                  splitter='best',
                                  max_depth=30,
                                  min_impurity_split=1e-09,
                                  presort = True,
                                  min_samples_leaf=3,
                                  min_samples_split=20)
  clf_ns = clf_ns.fit(X_train_ns,y_train_ns)
  #print clf

########################################### Random Forests ###########################################

  from sklearn.ensemble import RandomForestClassifier
  clf_rf_train_ns = RandomForestClassifier(n_jobs=3,
                                    criterion='entropy',
                                    max_depth=30,
                                    min_samples_split=20,
                                    min_samples_leaf=5,
                                    verbose=0,
                                    warm_start=False)
  clf_rf_train_ns = clf_rf_train_ns.fit(X_train_ns,y_train_ns)

############################### Measure the accuracy of the decision tree ##################################

  from sklearn import metrics

  def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True):
      y_pred=clf.predict(X)
      if show_accuracy:
          print("Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n")
      if show_classification_report:
          print("Classification Report")
          print(metrics.classification_report(y,y_pred),"\n")
      if show_confusion_matrix:
          print("Confusion Matrix")
          print(metrics.confusion_matrix(y,y_pred),"\n")

  print "-------------------Performance of the Decision Tree on the Unsegmented Training Data--------------------------------"
  measure_performance(X_train_ns,y_train_ns,clf_ns, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True)

  print "-------------------Performance of the Random Forest on the Unsegmented Training Data--------------------------------
"
```

```
    measure_performance(X_train_ns,y_train_ns,clf_rf_train_ns, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True)

################# Evaluating the performance of the decision tree on the testing data ############################

    clf_dt_ns = tree.DecisionTreeClassifier(criterion='entropy',
                                    splitter='best',
                                    min_impurity_split=1e-09,
                                    max_depth=30,
                                    presort = True,
                                    min_samples_leaf=3,
                                    min_samples_split=20)
    clf_dt_ns.fit(X_train_ns, y_train_ns)
    x_ns = clf_dt_ns.predict(X_test_ns)
    #for i in range(1,(len(X_test))):
    #     print x[i]
    #     print y_test[i]

    print "-----------------------Performance of the Decision Tree on the Unsegmented Testing Data--------------------------------
--"
    measure_performance(X_test_ns,y_test_ns,clf_dt_ns)

################# Evaluating the performance of the random forest on the testing data ############################

    clf_rf_test_ns = RandomForestClassifier(n_jobs=3,
                                   criterion='entropy',
                                   max_depth=30,
                                   min_samples_split=20,
                                   min_samples_leaf=5,
                                   verbose=0,
                                   warm_start=False)
    clf_rf_test_ns = clf_rf_test_ns.fit(X_train_ns,y_train_ns)
    y_ns = clf_rf_test_ns.predict(X_test_ns)

    print "------------------------Performance of the Random Forests on the Unsegmented Testing Data-----------------------------
-----"
    measure_performance(X_test_ns,y_test_ns,clf_rf_test_ns)


##################################### Exporting results to a CSV File ###################################

#print len(X_test)
#book_result_ns = xlwt.Workbook(encoding="utf-8")
#sheet3 = book.add_sheet("Sheet 3")
#sheet3.write(0,0,"Average Velocity")
#sheet3.write(0,1,"Average Acceleration")
#sheet3.write(0,2,"Mode - Ground Truth")
#sheet3.write(0,3,"Mode - Predicted")
#for i in range(1,(len(X_test_ns))):
#     sheet3.write(i,0,X_test[i][0])
#     sheet3.write(i,1,X_test[i][1])
#     sheet3.write(i,2,y_test[i])
#     sheet3.write(i,3,x[i])
#book_result.save("Test_Results.csv")
```

```
########################################## Data visualisation ##########################################

import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
from sklearn.externals.six import StringIO
import pydot
dot_data = StringIO()
tree.export_graphviz(clf_ns,
                out_file='tree_dt_ns.dot')

#tree.export_graphviz(clf_rf_train,
#                 out_file='tree_rf.dot')

#i_tree_ns = 0
#for tree_in_forest in clf_rf_train.estimators_:
#    with open('tree_ns_' + str(i_tree) + '.dot', 'w') as my_file_ns:
#        my_file_ns = tree.export_graphviz(tree_in_forest, out_file = my_file)
        #(graph_rf,) = pydot.graph_from_dot_file('tree_' + str(i_tree) + '.dot')
        #graph_rf.write_pdf('randomForests.pdf')
#    i_tree_ns = i_tree_ns + 1

#graph = pydot.graph_from_dot_data(dot_data.getvalue())
#graph.write_pdf("mode.pdf")
#from subprocess import check_call
#check_call(['dot','-Tpng','tree.dot','-o','OutputFile.png'])

(graph_dt_ns,) = pydot.graph_from_dot_file('tree_dt_ns.dot')
graph_dt_ns.write_pdf('dectree_ns.pdf')

#(graph_rf_ns_1,) = pydot.graph_from_dot_file('tree_ns_1.dot')
#graph_rf_ns_1.write_pdf('randomForests_ns_1.pdf')

#(graph_rf_ns_2,) = pydot.graph_from_dot_file('tree_ns_2.dot')
#graph_rf_ns_2.write_pdf('randomForests_ns_2.pdf')

#(graph_rf_ns_3,) = pydot.graph_from_dot_file('tree_ns_3.dot')
#graph_rf_ns_3.write_pdf('randomForests_ns_3.pdf')
```

ANNEXURE 4: Python Implementation of the Problem using selected trajectories of the GeoLife Dataset for training the classifier and data collected in Singapore for testing the classifier

```
import math
import numpy as np
import scipy as sc
import pandas as pd
import xlwt
import csv

####################################################################################################
######################################## Change Point Segmentation ###################################
####################################################################################################

############################################## Step 1 ##################################################

book = xlwt.Workbook(encoding="utf-8")
sheet1 = book.add_sheet("Sheet 1")
sheet1.write(0,0,"Latitude")
sheet1.write(0,1,"Longitude")
sheet1.write(0,2,"Average Velocity")
sheet1.write(0,3,"Average Acceleration")
sheet1.write(0,4,"Mode")
df = pd.read_csv('C:\\Users\\N1600060B\\Documents\\Transport Mode Detection\\DataSet - Singapore\\Trip
4.csv',sep=',',header=None)
#ff=open('dumpfile.csv','w+')
data = df.values
#print data
# data = datat.tolist()
#print len(data[:,6])
cValues = np.where((data[:,6])>1.6)[0]
#print len(cValues)
segment = data[3:cValues[0]+1,:]
#print segment
segments = [segment]
#print segments
start = 4
end = -1
for i in range(3,len(cValues)):
    if cValues[i]!=cValues[i-1]+1:
        end = cValues[i-1]+1
        seg1 = data[start:end,:]
        start2 = cValues[i-1]+1
        end2 = cValues[i]
        seg2 = data[start2:end2,:]
        if len(seg1)!=0:
            #print len(seg1)
            segments.append(seg1)
        if len(seg2)!=0:
            segments.append(seg2)
            #print len(seg2)
        start = cValues[i]
#print segments
#print len(segments)
################################################## Step 2 ##########################################

newseglist = []
newseglist.append(segments[0])
mergeIndex=0
for ii in range(1,len(segments)):
    if len(segments[ii])<=3:
        newseglist[mergeIndex] = np.concatenate((newseglist[mergeIndex],segments[ii]),axis=0)
    else:
        newseglist.append(segments[ii])
        mergeIndex = mergeIndex+1
#------------------------------debugging----------------------------------#
# print len(newseglist)
# for jj in range(0,len(newseglist)):
#   print len(newseglist[jj])
# print newseglist
#----------------------------end of debugging-----------------------------#

########################################## Step 3 ##############################################

seglistReborn = []
mergeMode = 0
mergeIndex = 0
for j in range(0,len(newseglist)):
    if len(newseglist[j])>10:
        seglistReborn.append(newseglist[j])
        mergeMode=0
        seglistBorn = [seglistReborn]
    elif len(newseglist[j])<=10 and mergeMode==0:
        seglistReborn.append(newseglist[j])
        mergeIndex = len(seglistReborn)-1
        mergeMode=1
        seglistBorn = [seglistReborn]
    elif len(newseglist[j])<=10 and mergeMode==1:
        seglistReborn[mergeIndex] = np.concatenate((seglistReborn[mergeIndex],newseglist[j]),axis=0)
        seglistBorn = [seglistReborn]
    #print seglistBorn[0]
del newseglist
#------------------------------debugging----------------------------------#
#print len(seglistReborn)
#for jj in range(0,len(seglistReborn)):
#     print len(seglistReborn[jj])
#print seglistReborn[1][0][0]
#----------------------------end of debugging-----------------------------#

################################### Step 4 / Calculating Features #######################################

array_segment=[]
#meanData = np.sum(seglistReborn[0][:,[6,7]],axis=0)/len(seglistReborn[0])
#print meanData
```

```
for i in range(1,(len(seglistBorn[0]))):
    #print seglistBorn[0][i]
    if len(seglistBorn[0][i]) is not 0:
        meanVelocity = sum(seglistBorn[0][i][:,[6]])/len(seglistBorn[0][i])
        meanAcceleration = sum(seglistBorn[0][i][:,[7]])/len(seglistBorn[0][i])
    #print "Length of Segment = "
    #print len(seglistReborn[i])
    #print "Start Lattitude = "
    #print seglistReborn[i][0][0]
    #print "Start Longitude = "
    #print seglistReborn[i][0][1]
    #print "End Lattitude = "
    #print seglistReborn[i+1][0][0]
    #print "End Longitude = "
    #print seglistReborn[i+1][0][1]
    #print "Mode = "
    #print seglistReborn[i][0][8]
    #print " Features = "
    #print np.sum(seglistReborn[i][:,[6,7]],axis=0)/len(seglistReborn[i])
    #print "--------------------------------------"
        #print len(seglistBorn[0][i])
        sheet1.write(i,0,seglistReborn[i][0][0])
        sheet1.write(i,1,seglistReborn[i][0][1])
        sheet1.write(i,2,meanVelocity[0])
        sheet1.write(i,3,meanAcceleration[0])
        sheet1.write(i,4,seglistReborn[i][0][8])
book.save("trip_4_segments.csv")

######################################################################################################
################################### Classification of Segmented Data ########################################
######################################################################################################

import csv
import numpy as np

f1 = file('C:\\Users\\N1600060B\\segments_train_data_2.csv','r')
f2 = file('C:\\Users\\N1600060B\\trip_segments.csv','r')

c1=csv.reader(f1)
c2=csv.reader(f2)

#################################### Header contains feature names #######################################

#training data
row_train=next(c1)
feature_names_train=np.array(row_train)
#print feature_names_train

#testing data
row_test=next(c2)
feature_names_test=np.array(row_test)
#print feature_names_test

################################## Load dataset and target classes ##########################################
```

```
mode_X_train,mode_y_train=[],[]
mode_X_test,mode_y_test=[],[]

#training data
for row_train in c1:
    mode_X_train.append(row_train)
    mode_y_train.append(row_train[4])
mode_X_train = np.array(mode_X_train)
mode_y_train = np.array(mode_y_train)
#print mode_X_train
#print mode_y_train[1]

#testing data
for row_test in c2:
    mode_X_test.append(row_test)
    mode_y_test.append(row_test[4])
mode_X_test = np.array(mode_X_test)
mode_y_test = np.array(mode_y_test)
#print mode_X_test
#print mode_y_test[1]

####################################2 Retain required columns ########################################

#training data
mode_X_train = mode_X_train[:, [2,3]]
feature_names_train = feature_names_train[[2,3]]
#print feature_names_train
#print mode_X_train

#testing data
mode_X_test = mode_X_test[:, [2,3]]
feature_names_test = feature_names_test[[2,3]]
#print feature_names_test
#print mode_X_test

########################################## Encode mode ############################################

from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()

#training data
label_encoder_train = enc.fit(mode_y_train[:])
print("Categorical Classes Training:", label_encoder_train.classes_)
integer_classes_train = label_encoder_train.transform(label_encoder_train.classes_)
print("Integer Classes Training:", integer_classes_train)
t_1 = label_encoder_train.transform(mode_y_train[:])
mode_y_train[:] = t_1
#print (mode_X_train[4], mode_y_train[4])

#testing data
label_encoder_test = enc.fit(mode_y_test[:])
print("Categorical Classes Testing:", label_encoder_test.classes_)
integer_classes_test = label_encoder_test.transform(label_encoder_test.classes_)
```

```
print("Integer Classes Testing:", integer_classes_test)
t_2 = label_encoder_test.transform(mode_y_test[:])
mode_y_test[:] = t_2
#print (mode_X_test[4], mode_y_test[4])

####################################### Classification ################################################

#decision tree
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=30, min_samples_leaf=3,
presort=True,min_samples_split=20)
clf = clf.fit(mode_X_train,mode_y_train)
#print clf.predict(mode_X_test)

#random forests
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(n_jobs=3, criterion='entropy', max_depth=30,min_samples_split=20,min_samples_leaf=5)
clf_rf = clf_rf.fit(mode_X_train,mode_y_train)

############################# Function to calculate classification accuracy ##################################

from sklearn import metrics

def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=True, show_confusion_matrix=True):
    y_pred=clf.predict(X)
    if show_accuracy:
        print("Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n")
    if show_classification_report:
        print("Classification Report")
        print(metrics.classification_report(y,y_pred),"\n")
    if show_confusion_matrix:
        print("Confusion Matrix")
        print(metrics.confusion_matrix(y,y_pred),"\n")

#measure_performance(mode_X_train,mode_y_train,clf, show_accuracy=True, show_classification_report=True,
show_confusion_matrix=True)

############# Evaluating the performance of the classifiers on the testing data ####################################

clf_dt = tree.DecisionTreeClassifier(criterion='entropy', max_depth=30, min_samples_leaf=3,
presort=True,min_samples_split=20)
clf_dt.fit(mode_X_test, mode_y_test)
clf_rf_test = RandomForestClassifier(n_jobs=3, criterion='entropy',
max_depth=30,min_samples_split=20,min_samples_leaf=5)
clf_rf_test = clf_rf_test.fit(mode_X_test,mode_y_test)
print "----------------------Decision Tree----------------------------------"
measure_performance(mode_X_test, mode_y_test,clf_dt)
print "----------------------Random Forests---------------------------------"
measure_performance(mode_X_test, mode_y_test,clf_rf_test)
```

PROJECT DETAILS

Student Details			
Student Name	**DEBASMITA GHOSE**		
Register Number	130907158	Section / Roll No	A / 18
Email Address	debasmita2pat@gmail.com	Phone No (M)	+65-93983171
Project Details			
Project Title	**DETERMINATION OF TRANSPORTATION MODES USING MOBILE PHONES**		
Project Duration	5 months	Date of reporting	3rd January, 2017
Expected date of completion of project	3rd June, 2017		
Organization Details			
Organization Name	**NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE**		
Full postal address with pin code	Hardware and Embedded Systems Lab, School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Ave, Singapore 639798		
Website address	scse.ntu.edu.sg/Research/HESL		
Supervisor Details			
Supervisor Name	**Mr. Siew Kei Lam**		
Designation	Assistant Professor		
Full contact address with pin code	Hardware and Embedded Systems Lab, School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Ave, Singapore 639798		
Email address	siewkei_lam@pmail.ntu.edu.sg	Phone No (M)	+65-6908 3336
Internal Guide Details			
Faculty Name	**Mrs. Aparna U.**		
Full contact address with pin code	Dept. of E&C Engg., Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	aparna.u@manipal.edu		