Knowledge inference from smartphone GPS data

Mohsen Rezaie

A Thesis

in

The Department

of

Geography, Planning and Environment

Presented in Partial Fulfillment of the Requirements

for the Degree of

MSc (Geography, Urban and Environmental Studies) at

Concordia University

Montréal, Québec, Canada

April 2018

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohsen Rezaie**

Entitled: **Knowledge inference from smartphone GPS data**

and submitted in partial fulfillment of the requirements for the degree of

MSc (Geography, Urban and Environmental Studies)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Prof. Alan Nash

_____ External Examiner
Prof. Bilal Farooq

_____ Examiner
Prof. Yann-Gaël Guéhéneuc

_____ Supervisor
Prof. Zachary Patterson

_____ Co-supervisor
Prof. Jia Yuan Yu

Approved by _____
Pascal Biron, Chair
Department of Geography, Planning and Environment

_____ 2018

_____
André Roy, Dean
Faculty of Arts and Science

Abstract

Knowledge inference from smartphone GPS data

Mohsen Rezaie

With the advent of the incorporation of GPS receivers and then GPS-enabled smartphones in transportation data collection, many studies have looked at how to infer meaningful information from this data. Research in this field has concentrated on the use of heuristics and supervised machine learning methods to detect trip ends, trip itineraries, travel mode and trip purpose. Until now approaches to inference have relied on the use of fully-validated data. However, respondent burden associated with validation lowers participation rates and reduces the amount of precisely validated data because some people do not validate their trips or misreport them.

This thesis consists of two studies. In the first study I propose the application of a semi-supervised method to mode detection from smartphone travel survey data. Semi-supervised methods let researchers and planners use both validated and un-validated data. I compare the accuracy of three popular supervised methods (Decision Tree, Random Forest and Logistic Regression) with a simple semi-supervised method (Label Propagation with KNN kernel). Simple features such as speed, duration and length of trip and closeness of start and end points to transit network are used for model estimation. The results show that the semi-supervised method outperforms the supervised methods in the presence of high proportions of un-validated data and better predicts the observations in the test set. Furthermore, the run-time of the best model among the supervised methods was on average almost 16 times longer than the average run-times of the semi-supervised method.

In the second study, I develop a method to infer transit itineraries from smartphone travel survey data. Since the application of semi-supervised algorithms in travel surveys and transit itinerary detection are both in the early stages of development, a supervised approach is taken to tackle the problem of transit itinerary detection. To this end, trip features were extracted from smartphone

collected data and transit network information available in the General Transit Feed Specification (GTFS) format. Based on these features, a Random Forest model was trained. Using the model, transit routes for 62% of trip segments was correctly detected.

Acknowledgments

Thanks, to each and everyone who supported, small or big, verbally, action or just simply, a nod.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Collecting daily travel data is one of the most important challenges in urban transportation planning and transportation infrastructure management. Transportation planners use the information from current daily travel to study people's travel behavior and plan the future transportation system.

Since the 1950s, methods used for household trip data collection have been evolving. The initial household travel surveys developed in the 1950s used face-to-face interviews that suffered from high costs and being time consuming (Shafique and Hato, 2015). To overcome these disadvantages, researchers started using computer-assisted telephone interviews, which were less costly and time consuming compared to face-to-face interviews. However, and while they remain the workhorse of urban transportation data collection, these also have some disadvantages, such as inaccuracy in recording of starting and ending times of trips, trip under-reporting and non-response (Bricka et al., 2009; Shafique and Hato, 2015; Stopher and Greaves, 2007). These problems are attributed partly to the large number of questions asked to respondents that are based on memory (Nitsche et al., 2014), and partly to respondents forgetting short trips or reporting wrong data unintentionally.

GPS technology, during late 1990s, introduced a new era of trip data collection by making it possible to record the traces of travelers via the installation of GPS devices in respondent vehicles. GPS devices carried by travelers during their daily trips were also used. Using GPS devices for collecting trip data has some disadvantages, such as the relatively high cost of such devices and that it can take time for GPS to get accurate fixes on locations. The latter can result in inaccurate GPS points collected when a device starts working. This is known as a "cold start" and occurs when

a device has no almanac data. Similarly, "warm starts" occur when a device has valid almanac data, but lacks ephemeris data (Yoldi and Lebena, 2002). Also, it is quite common for travelers to forget their GPS devices when traveling (Nitsche et al., 2014). Finally, these devices are typically installed in traveler vehicles so that only people using vehicles are included in data collection. These disadvantages have led to GPS devices being primarily as supplements to traditional data collection methods.

In the last decade, smartphones equipped with GPS chips have become increasingly popular as a tools to collect travel behavior data. Since smartphones have become one of the essential devices of modern life and are usually carried by their owners wherever they go, they have a clear advantage over GPS devices in transportation surveys. Also smartphones do better with warm starts compared to dedicated GPS devices (Bierlaire et al., 2013). Considering the potential of smartphones for collecting travel data, researchers have increasingly focused on developing mobile applications to collect travel data (Patterson and Fitzsimmons, 2016; Safi et al., 2015).

Despite the clear advantages of smartphones and GPS receivers, and while they can easily record time and location information, the use of smartphone collected data requires methods for inferring travel information, methods that are in the process of being developed. Essentially, smartphone-collected data is a sequence of time-stamped GPS locations. As such, it must be processed and turned into travel information such as travel mode and route to be used for transportation planning purposes. Generally, information inference is divided into the following sequential steps: trip/segment identification, map-matching, mode detection and purpose detection (Shen and Stopher, 2014). Among these, mode detection has received the most attention in the scientific literature. Some smartphone and other wearable accessories even provide an estimation of mode of transport by default, but these estimates are not always accurate and, because they are intended to detect the type of activity of users for health purposes, they do not provide the details needed in urban planning. Also some accurate and detailed information detected by applications such as Google Maps are not available publicly.

While many studies have tried to automatically detect mode of transport and achieve increasingly better results (i.e. higher accuracies), they have mostly used supervised learning methods. All data used for calibrating supervised algorithms must be validated by participants and un-validated

data cannot be used in the model estimation. When all of the observations in a dataset are validated, supervised algorithms are believed to estimate better models compared to unsupervised and semi-supervised algorithms. Therefore, supervised algorithms are usually used to estimate models on fully validated data. However, when there are unvalidated observations in a dataset, use of a semi-supervised method can help us take advantage of valuable unlabeled (un-validated) data. Even when data is collected by surveyors rather than regular respondents, since resources are limited, there is always a trade-off between survey size and precision of data collection in the survey (more specifically data validation). Use of a semi-supervised algorithm that only needs a small portion of data to be validated has the potential of using data from large-scale surveys without requiring participants to learn the validation process and taking their time for validating the collected data. As such, by further reducing participant burden, such an algorithm might make it possible to make important steps towards substituting traditional travel surveys with GPS travel surveys. As such, the use of semi-supervised methods in mode detection from smartphone travel survey data was the goal behind the first study of this thesis.

While a large body of literature has examined the methods to infer travel information from GPS records, few have considered inferring information about transit itineraries. As such, the purpose of the second study of this thesis is to develop algorithms to detect transit itineraries. Because both transit itinerary inference and the use of semi-supervised algorithms in travel information inference from smartphone travel surveys are new fields, the second study considers the novel problem of transit itinerary inference, while using supervised algorithms whose characteristics are better understood.

In Chapter 2, first a literature review of GPS surveys is provided. Then we provide a summary of methods used in previous mode detection and transit itinerary inference studies. Among machine learning algorithms used in the literature, I give a quick description of those that are widely used. I discuss some of their advantages and disadvantages and explain the reasons leading to the selection of the algorithms used in this thesis. Datasets used in this study are explained in Chapter 3. Chapter 4 describes the semi-supervised vs. supervised mode detection study. First, the steps taken to find optimal meta-parameters for the models are explained. Then, models are estimated and a few comparisons between the accuracy of the models in different scenarios are provided and discussed.

Chapter 5 documents the work related to the detection of transit route itinerary using DataMobile[1] and the General Transit Feed Specification (GTFS) data provided by the Société de transport de Montréal (STM). Chapter 6 includes a conclusion of both studies, limitations of the two studies and suggestions for future works.

[1] https://www.itinerum.ca/about.html

Chapter 2

Literature Review

This section concentrates primarily on methods used to to detect travel mode from GPS- and smartphone-collected data. Given there small amount of literature on transit itinerary inference, methods used to detect transit itinerary are left to a short discussion after that on mode.

Early studies in mode detection were based on heuristically derived rules. With this approach, rules were manually set to detect mode of transport (Gilani, 2005; Bohte and Maat, 2009). Later, researchers became interested in the automatic inference of information from travel data. Researchers started using machine learning algorithms (Miller et al., 1992; Wolf et al., 2003) to save time (rather than spending time looking for the rules manually) and got better results. This literature has focused on supervised learning methods, of which there are many. These methods use fully validated (labeled) data to train a model. Recently, there has been research into the use of un-supervised learning methods using unvalidated data to develop travel information inference models based on GPS data (Patterson et al., 2003; Liao et al., 2007). However, these models were user-specific, aiming at individual daily behavior. Therefore, the models trained in these studies were suitable to estimate the behavior of the same person rather than being able to predict the behavior of other individuals. Also the data were collected by the authors was small in size.

The rest of this consists of two subsections. The first part discusses the evolution of travel surveys, different steps of GPS data processing and the first efforts and studies in this area, based rule-based approaches of processing and information inference. The second part explains machine

learning methods, discusses studies that have used learning algorithms and also describes the algorithms chosen for this current study.

2.1 Traditional Travel Survey Methods

Travel surveys are a common way to collect data needed in transportation planning. Starting in the early 1950s, face-to-face methods were used to collect data from households. Surveyors visited households in person and asked questions about general information on the households and their travel habits. This information was collected by paper and pencil. These surveys included questions on the socio-demographic information of household, including number of people living in the household, household income, number of vehicles and also about daily trips made by household members like their origin and destination, mode of transport, trip purpose and start time.

However because of confidentiality of data and cost of these surveys, in the 1960s, mail-out/mail-back methods were introduced where questionnaires were sent to households by mail and after completing the answers, were mailed back to the surveyors (Wolf, 2000). These methods are still being used in some cases (e.g., (Stopher et al., 2007; Handy et al., 2005)).

In the 1980s, computer-assisted methods were introduced to replace the use of paper and pencil to facilitate data entry. The three different types of this method are computer-assisted telephone interviews (CATI), computer-assisted personal interviews, and computer-assisted self-interviews (CASI) (Stopher et al., 2008). In computer assisted personal interviews, surveyors meet respondents in person and ask them to answer questionnaires on a computer. In CATI, surveyors call households on the phone and respondents answer questions by pressing dial numbers of the telephone or by responding to the interviewer who inputs respondent answers. In CASI surveys, such as web-based surveys, respondents have access to questionnaires through the web and can answer questions remotely. Even though these methods have been used for decades in transportation studies and are still in use, they suffer from at least two issues. First is non-response (Tourangeau et al., 1997). A considerable portion of people ignore the questionnaires and do not respond since the answering procedure is time consuming. Also some of them may not mail back surveys because they must return them by post. Even though on-line options are less burdensome, these surveys also suffer

from non-response. Second is misreporting, when when respondents intentionally or unintentionally mis- or under-report information about their daily travel. Misreporting can be caused by failure to correctly recollect, or from carelessness (Wolf, 2000).

In the late 1990s GPS was proposed as a potential tool to be used in travel surveys (Wagner, 1997). It was seen to have potential to overcome some misreporting issues (Bricka et al., 2009). Studies report rates of 20-30% of trip under-reporting in traditional surveys when compared to GPS surveys (Ogle et al., 2002; Wolf et al., 2003; Bricka and Bhat, 2006). Also, heavy travelers, those who make more non-motorized trips, workers who travel as part of their jobs and middle-aged individuals are more likely to under-report their trips. As a results, GPS collection methods have a greater impact on the records of trips belonging to these demographics (Bricka et al., 2012).

To collect other attributes of trips, such as mode and purpose of the trip, personal data assistants (PDAs) were used in early studies of GPS survey (Bachu et al., 2001). Respondents had to record this information at the beginning of each trip, which made the survey procedure more complicated and costly. To decrease burden on participants, prompted recall (PR) surveys were introduced. With prompted recall, participants are provided information on their trips using a map showing GPS log data and are asked to add other attributes of their trips. They may even be able to correct and augment GPS log data that may have been missing due to lack of GPS signal, or when not carrying the device. In a study in 2001, a face-to-face survey was conducted along with a PR survey (Bachu et al., 2001). Participants had to report trip purpose and vehicle occupancy as well. The study suggests that PR surveys make data collection easier for respondents since it only takes 15-20% of the time required for answering face-to-face questions. In addition, with PR, respondents were found to be able to recall information for up to four days. Web-based surveys are another form of PR where respondents receive a map of their daily trip through a website and they can edit it, validate the information, and add other attributes of trips (Giaimo et al., 2010; Greaves et al., 2010).

2.2 The Potential of GPS Data

Since smartphones are increasing in popularity and because almost all of them include an internal GPS chip, they are an appealing choice for GPS surveys (Gilani, 2005). As smartphone

penetration is increasing, the use of these devices is becoming even more promising. Also most smartphones have both GPS and accelerometer sensors, so both of these types of information can be collected (Shen and Stopher, 2014). As stated before, smartphones also do better in warm start compared to GPS-dedicated devices (Bierlaire et al., 2013).

There is a lot of new research using smartphones to collect GPS data for trips (Reddy et al., 2010; Hudson et al., 2012; Xiao et al., 2012; Bierlaire et al., 2013; Patterson and Fitzsimmons, 2016). Smartphones have also been used to study freight transportation (Joubert and Axhausen, 2011) and public transportation networks (Saddier et al., 2016).

The City of Montreal has started a collaboration with Waze^{TM}. Waze is an application collects spatial information of the users. This information can help the municipal to have detailed and real time information about the road and transportation network (CBC-News, 2016). Patterson (Patterson, 2017) has also developed the travel survey application MTL Trajet for the City of Montreal to collect information on the travel behavior of Montrealers. The app was used in the fall of 2016 and 2017.

2.2.1 Steps in GPS Data Processing

GPS devices and smartphones can record time and position information. For the most part, they do not indicate trip mode, trip end and purpose of the trip except for in-vehicle devices that record engine off as trip end (Shen and Stopher 2014). Although in prompted recall surveys, participants provide some of this information, doing so places additional burden on respondents. Thus, not all of them fill out all the fields carefully in a real-world surveys. This has encouraged researchers to substitute prompted recall surveys with information inferred from GPS (and other sensor) data. Also like any other device, GPS devices are not absolutely accurate. Therefore the data provided by these devices does not necessarily match with road networks and does not necessarily indicate the route taken by the user. The data must be processed before being used in transportation planning. In addition to the fact that GPS devices may run out of battery, or that respondents might leave devices somewhere else while making trips, there are two main sources of error; signal loss and signal noise (Biljecki, 2010). With cold or warm starts while looking for reliable signals, a device may record some points that are inaccurate or miss a part of trips. Urban canyons are another source of signal

problems where high-rise buildings that disrupt GPS signals. Signals can also be lost in tunnels and underground rail networks. These signal problems result in spurious trips, missing trips or part of trips (Shen and Stopher, 2014).

The first step toward processing data is Trip Identification (TI) or more specifically Segment Identification (SI). Trips are defined as a commute from one point at which an activity has taken place to another point where another activity takes place. Sections of a trip done with a single mode of transport are trip segments.

Mode detection and map matching (route detection) are the next steps in data processing. Since location data recorded by GPS devices are not always precise, they must be matched with transportation networks. Because of the close relationship between route and mode of transport, their detection is usually done simultaneously (Shen and Stopher, 2014).

Another aspect of trip information to be inferred is trip purpose. Despite the fact that this is probably the most difficult aspect of trip information to infer, some researches have tried to detect it (Wolf et al., 2001; Stopher et al., 2008; Bohte and Maat, 2009; Yazdizadeh et al., 2018).

2.2.2 Methods Used in GPS Data Processing

Trip and segment identification processing are typically rule-based. In manually-set rule-based methods, a set of hierarchical rules and criteria is used by the analyst processing the data (Bohte and Maat, 2009; Wolf, 2000; Schüssler and Axhausen, 2009).

As mentioned before, there are two important problems in TI: signal loss and signal noise. Wolf suggested at least a couple of minutes delay between two trips since the traffic light cycles are mostly less than this time period (Wolf, 2000). Regarding the ”120 second rule”, Tsui studied signal loss and suggested that when the duration of signal loss is between 120 seconds and 600 seconds and change in location is less than 50 meters, there should be a short duration activity, and when the traveled distance is more than 500 meters, there may be an underground trip even if the duration for signal loss is more than 120 seconds (Tsui, 2005; Gonzalez et al., 2010). Also some activities may take less time such as pick up/drop off that can even be done in less than a minute. In this regard Shen and Stopher suggested 60 seconds as dwell time (Shen and Stopher, 2013). In another study, a two-step model is used which in the first step movement between two meaningful

points is considered as a journey and in the second step trips are found with 12 seconds as a dwell time threshold (Biljecki, 2010). However, the choice of this threshold is not justified by the authors. Also this relatively low threshold for the time interval between the trips results in too many trips. Therefore, in another step after mode detection, excessive trips are found and merged.

There is also a large body of literature in route and mode detection using rule-based approaches. Most rule-based models used in mode detection are based on travel speed, acceleration/deceleration and GIS information (Gilani, 2005; Bohte and Maat, 2009). Stopher et al. (Stopher et al., 2008) suggest that walking trips can be distinguished with speeds below 6 km/h and vehicle trips can be distinguished with speeds above 40 km/h. Also acceleration/deceleration can be helpful in distinguishing walk and bicycle trips or automobile and public transport trips. Even household information such as the location of place of residences can be useful. Also public transport time tables, routes and stops can help to distinguish public transport trips. Using this method Stopher et al. report that 95% of trip modes are detected correctly. Data from accelerometer sensors is another source of data to use in detecting mode but it does not detect all modes and further information is still needed to distinguish modes with similar acceleration patterns (Stenneth et al., 2011). Accelerometer data is also more sensitive to how the device is carried (e.g., fastened to arms or legs of a cyclist). Automatic Vehicle Location (AVL) is another source of information which has recently been used in mode and route detection. It is used by Carrel et al. to detect transit routes and using a set of rules they correctly detected almost 90% of the trips out of 103 validated trips (Carrel et al., 2015).

Fuzzy rule-based algorithms are another method used in GPS data processing. This method penalizes modes where some characteristics of trip are not close to usual characteristics of the modes, such as speed and acceleration. Biljecki used a fuzzy method to detect modes (Biljecki, 2010). He used speed, acceleration, proximity to a network and proximity to water surface among other variables as characteristics of trips. Schussler and Axhausen proposed a fuzzy-logic approach in which some fuzzy rules are made based on median of speed distribution and 95th percentiles of speed and acceleration distributions for which a score is assigned (Schüssler and Axhausen, 2009). The final decision as to what mode is being used is made based on the minimum score for each mode among the set of characteristic scores.

In a 2000 study, a number of simple rules were used to match the points with the underlying transportation network (White et al., 2000). In a recent study, a probabilistic method was used to match GPS records with routes on the map (Bierlaire et al., 2013). It generated a set of candidate paths and calculated the likelihood of each of them being the true-path by assuming independently normal distributed errors of location in longitudinal and latitudinal directions resulting in a Rayleigh distribution of overall error. The method was applied on a sample of 19 trips recorded in approximately 2 hours. Parameters of the model made on the sample were all statistically significant.

While there is a large body of literature in GPS data processing using manually-set rule-based models, there is a growing number of studies that have considered the use of machine learning methods, particularly in the inference of travel mode.

2.2.3 Machine Learning

Generally, and in contrast to manually-set rule-based algorithms, Machine Learning algorithms are algorithms that enable computers to build models without being explicitly programmed. Compared to traditional statistics, in machine learning emphasis is on algorithmic considerations such as efficiency of algorithms. Also while asymptotic behaviour is often praised and sought in statistics, machine learning pays more attention to finite sample behavior. It is not just about convergence of estimates when the sample sizes grow to infinity but is more focused on the prediction accuracy of models. Also machine learning tends to work with a more "distribution-free" paradigm with as few assumptions as possible and letting the algorithm choose the best model (Shai and Shai, 2014).

To find the best model based on the given observations, sample data is usually divided into three parts: a training set, a validation set to validate the model and variables used, and a test set to report prediction error. Models are made based on the training set by choosing the best fitting model to the training set. Algorithms are steps looking for the best values for parameters given a hypothesis about the structure of the model (e.g., degree of freedom). The best hypothesis is chosen based on the results of the model on validation tests (cross-validation) which is disjoint of the training set. Cross-validation captures over-fitting of model to training set. Lastly the error of results using the best model on the test set is reported as model's prediction error.

Supervised, Unsupervised and Semi-supervised Algorithms

Based on the presence of known output in the sample data, learning algorithms are divided into supervised, unsupervised and semi-supervised. An algorithm is called supervised only if it takes advantage of known labels or values for the output variable in the sample dataset. With these algorithms, the model is prepared in the training process and is corrected when its predictions for the training set do not match with the observations. Regression and classification problems are of this type and algorithms such as Logistic Regression, Artificial neural Network and classification Trees are usually considered supervised (Bishop, 2006). However, there are some extensions of these algorithms that are unsupervised. In unsupervised learning, training data consists of input values without any corresponding target values. Unsupervised algorithms estimate a model by deducing structures present in observations. It is usually done by organizing data by similarity. Common problems solved by unsupervised learning are clustering and association rule learning. One of the most popular and simple unsupervised algorithms is K-Means (Bishop, 2006). Also, when a sample is mostly unlabeled but there are few labelled observations, semi-supervised algorithms are used which are usually extensions of other algorithms by making assumptions about how to model unlabelled observations.

Learning Methods

Based on function, there are many types of algorithms used in machine learning that are different in structure and computational demand (such as required memory). In this section a review on some of the most popular learning methods is provided.

A simple but useful class of algorithms are instance-based learners. Usually there are not many steps in their learning procedures. These methods store sample data and compare new data with it to make a prediction based on similar observations. The focus is on the representation of data and finding a suitable measure of similarity. One of the most popular instance-based learners is K-Nearest Neighbours (KNN) with which data are projected in a multi-dimensional space where each axis represents a feature of the instances. Given a new instance, K observations with the least Euclidean distance to this new instance are selected within the sample data. The prediction for new

a instance is the mode of labels for those K observations (Bishop, 2006).

A very popular and simple method in learning is the classification tree. This method is often fast and accurate. Decision Trees can be represented as a directed graph consisting of a finite number of nodes and arrows of which each node is a direct successor (child) of one and only one node, except for the root which does not have any predecessor (parent). Each arrow (branch) represents a rule on a feature. It generates a Decision Tree based on a sample set of observations by making a set of hierarchical rules on features of instances. It is a rule-based modelling where rules are generated and the best one is chosen automatically by the algorithm rather than adjusting thresholds and comparing rules manually. As a result, it is more likely to find the best model. Decision Trees need to be pruned to avoid over-fitting. Models can be stored easily without consuming a lot of memory and can be applied on new observations very quickly for prediction (Gelfand et al., 1989).

Artificial Neural Network is a method inspired by the functioning of brains that process complex tasks through the interaction of many simple processing units. Each processing unit takes some unprocessed inputs or outputs of other processing units as input and returns an output. An ANN model can be represented as a directed graph with hierarchical groups (layers) of nodes, to which nodes within a layer are not connected, and all the nodes are both successors and predecessors of other nodes, except for the nodes in the first and last layer. Nodes in the first layer represent the input features, and output is returned in the last layer. ANN is good in modelling complex behaviours such as medical image and signal processing (Miller et al., 1992), finance (Wong and Selvi, 1998), power electronics and motion control (Bose, 1994) and gotechnical engineerig (Shahin et al., 2001). It also produces probabilistic outputs. The main disadvantage of ANNs is that they usually needs a lot of training (Bishop, 2006).

Regression models are statistical models used in machine learning. They are usually used when the desired output is a continuous variable but there are some regression models that can be used to predict discrete variables. A regression model is in the form of a function that provides a predicted value for a desired output (Bishop, 2006).

Bayesian algorithms are learning algorithms that apply Bayes' theorem explicitly to classification or regression problems. Assuming a prior distribution for parameters of the model, the data is applied to get a posterior distribution. Sometimes, features are assumed to be independent to ease

computation (Naive Bayes) but some algorithms such as Bayesian Networks can consider dependencies as well (McCallum et al., 1998).

The Markov chain is another probabilistic model that consists of a set of random variables having the property that given present, future of system is independent of the past:

$P(Z_i|Z_1, Z_2, \cdots, Z_{(i-1)}) = P(Z_i|Z_{(i-1)})$

where Z_j is jth variable in the sequence of variables. This means that if present state is known, knowledge on prior states does not improve prediction of future (Bishop, 2006).

Support Vector Machines (SVM) are based on projecting sample observations in a multidimensional space and splitting the data with hyper-planes to have sections including points with the same label (hard-margin) or least number of observations with other labels (soft-margin). In a simple SVM algorithm, a hyper-plane with maximum margin is chosen. SVMs initially find a linear relation of features to classify binary observations. However, it is possible to use SVMs in nonlinear classification by projecting instances in a space where the proper classifier has a linear form (Kernel trick). Also for multi-class problems it is possible to break down the problem into a finite number of binary-class problems (Bishop, 2006).

Rather than a single hypothesis, ensemble methods use a set of hypotheses to explain the data. They take votes over the set of hypotheses to predict the target values. Experimental evidence shows that ensemble methods (e.g., Random Forest) are often more accurate than any single classifier (e.g., Decision Tree) (Dietterich, 2002).

Clustering algorithms are typically used in unsupervised learning. They split sample data into subsets (clusters) based on a similarity measurement (such as vicinity to a centroid or density) to which observations within a cluster are similar, but from which observations of different clusters are dissimilar (Bishop, 2006).

Regularization algorithms are extensions to other methods, usually regression algorithms that penalize complex models in favour of having simpler models. The algorithms may result in the reduction in the number of features affecting the output or lowering their effects and providing the opportunity to decide about which one should be removed from the model. This also imposes increase in error of models (Schölkopf and Smola, 2002).

2.3 Machine Learning Methods and GPS Data

This subsection describes Machine Learning methods used in information inference from GPS and smartphone data. It considers primariliy travel mode detection and finishes with transit itinerary inference.

In 2013, Feng and Timmermans used a Bayesian Belief Network (BNN) (Feng and Timmermans, 2013). They used data collected by dedicated GPS devices. The data included 80,670 observations (GPS points) collected every second, almost 23 hours of data. They investigated the use of accelerometer and GPS data and detected modes correctly in 92% of cases. They used filtered data and excluded records for train journeys because of GPS signal loss inside trains and poor data.

Liao et al. used 60 days of data from one person to apply a hidden Markov model for mode detection (Liao et al., 2007). They used the first 30 days as the training set and the second 30 days as the test set. This model focused on one persons travel behavior and inferred detailed information about trips and even returned errors and mistakes made by the participant in validation, such as missing the destination stop and getting off the bus at the next stop. Although the model is built using an unsupervised procedure and works well in detail for travel patterns, it is trained for one person instead of a sample of people and therefore a customized model needs to be made for each person.

Neural Network modeling has been used in a number of studies. Tsui used a fuzzy-logic NN algorithm to detect modes of transportation (Tsui and Shalaby, 2006). While his process relied more on a fuzzy system, an NN was used to find the values for the fuzzy membership functions. His algorithm detected trip modes correctly in 94% of cases, although they were compared with trip diaries provided by participants that may have some of the typical errors in diaries such as under-reporting of short trips. Also in 2010, Gonzalez et al. used an NN in mode detection (Gonzalez et al., 2010). As input for their model, they used six attributes: two attributes representing accuracy of points, speed, acceleration, dwell time and standard deviation of distances between stop points. They trained the model with two types of inputs: firstly they fed their models with all the GPS coordinates recorded, and for the rest of the models they input "critical points" (i.e., points representing a change in direction of movement). Surprisingly, accuracies for the models trained with critical

points are higher than accuracies of models that took benefit of whole the data. This may have been caused by overfitting the models to the training data and noise included in the records. Although the method they used require only a small portion of data, only three modes were included (i.e., walk, bus and car).

In a 2012 study (Bolbol et al., 2012), an SVM was used to detect mode of transportation for daily trips of 81 persons over 2 weeks. To detect mode for the segments of the trips, a moving window algorithm was used to consider attributes of adjacent segments and the best window size was found to be three segments. In the network of London with different modes of transportation (i.e., walk, bike, car, bus, LRT and metro) the algorithm correctly detected modes for 88% of the trips (Bolbol et al., 2012). The authors did not use other sources of information like locations of bus stops and metro stations. From four attributes including length of segment, speed, acceleration and change in heading, they chose speed and acceleration to be used in their model.

Witayangkurn et al. also used an SVM on data collected in Japan by 100 participants during 1 month (Witayangkurn et al., 2013). To give a more realistic taste to the experiment and to try a survey with low cost and frequency of data collection, GPS points were collected every 5 minutes. After using a handful of rules to segment the trips and prepare the data, they reached a precision of 87% (portion of correct detection) and 85% recall (portion of correct detected modes to all observation of same mode).

STOP

In 2008 Zheng et al. used Conditional Random Field (CRF) to include the probability of transition from one mode to another one (Zheng et al., 2008). In this method, which is a specific case of Markov random field, after assigning a score (probability) for a mode of transportation for each segment of a trip, the probability of using that mode considering the modes used in the adjacent segments of the trip multiplies with it and the result is the total probability of that mode being used on that segment of the trip. In addition to CRF, three other algorithms including Decision Tree, Bayesian Net and SVM were used in this study. Finally the results of the four algorithms were compared and DT outperformed the other three algorithms in this comparison.

In another study, Reddy et al. compared 6 classifiers and finally used a combination of Decision Tree and hidden Markov model to detect transportation modes (Reddy et al., 2010). The method

was able to correctly detect the mode in 93.6% of cases. Also, this study includes the effect of sensor position while being carried by the participants and the results do not show major differences between the carrying positions such as in-hand, attached to chest or inside a bag. Although this study investigated interesting aspects of data collection and analysis. The "modes" considered in this study are still, walk, run, bike and motorized vehicle which does not discriminate public transportation and private vehicle, let alone different transit modes and routes of transit. Moreover this study does not consider GIS and personal information, which can useful in prediction.

Stenneth et al. compared 5 classification models including naive Bayes, Bayesian network, Decision Tree, Random Forest and Multilayer Perceptron (Stenneth et al., 2011). They found that the random forest dominates the others in precision. They also reported a noticeable improvement in mode detection by considering transportation network features, especially for the motorized and public modes of transportation.

Typically, researchers use speed and acceleration (and realted attributes such as average speed, maximum speed, standard deviation of speed and acceleration, 95^{th} percentile speed) as well as GIS information of trips in their studies (Shen and Stopher, 2014). Stenneth et al. have considered closeness to transit infrastructure for mode detection and defined several variables such as "average bus location closeness", "candidate bus location closeness", "average rail line trajectory closeness", "bus stop closeness rate" (Stenneth et al., 2011).

2.4 Learning Algorithm in This Study

Looking into the literature on information inference from GPS surveys, it can be found that most of the studies that have used learning algorithms have used them for mode detection (Wu et al., 2016). This is likely because of the importance of mode detection in processing GPS data and generally in transportation planning.

Bierlaire et al. used a probabilistic map matching method rather than rule-based algorithms (Bierlaire et al., 2013). The method used in their study is theoretically strong and the equations used there might be generalizable to higher dimensions and thus be applicable to other problems such as transit route detection. However, the inclusion of integrals in computations makes it more

likely to be time consuming, especially in higher dimensions. This method is not yet applied on large datasets; even in 2-dimensional space. It was not selected to be used in this thesis because the focus of the current study is on the application of learning algorithms that are used in a large number of studies on GPS data processing and have performed well on large datasets rather than comparison of every method used in the literature.

Looking for a suitable learning algorithm, there is no single right learning method for a problem. Different algorithms have been used in the literature to infer travel information and most of them have resulted in fairly high accuracies. A couple of studies have compared accuracy of some algorithms, but it does not necessarily mean that the dominant algorithms will get better accuracies for this problem as well. This is because a set of assumptions suitable for a problem in one domain may work poorly in another. Thus, there is no universally best model (Wolpert, 1996). However, cross-validation can be used to empirically choose the best model for a particular problem (Murphy, 2012).

Usually if all, or a part, of a dataset is labeled, the use of unsupervised methods such as clustering algorithms is unnecessary because neglecting the validated part results in a loss of information. Since validating few observations in GPS surveys is typically attainable in a research project, there is no need to use unsupervised algorithms.

SVM is a robust method that can be extended to nonlinear classification. But usually it is not a good choice when the number of observations is high. The algorithm is highly complex and requires a lot of memory (Horvath, 2003). It is also very slow in testing (Burges et al., 1996; Osuna and Girosi, 1998). ANN is also considered to give good results, but it is not preferred for large sample sizes because it large computational burden (Tu, 1996). Because in large surveys such as regional surveys there are potentially millions of trips and GPS records, these methods were not considered to be practical.

Bayesian algorithms can fit a model for this study but finding a good prior is an issue. It requires a good knowledge about the data and the relationship between the input and output variables. Logistic regression is another candidate for this study, but it is more used in decision behaviour studies.

Decision Tree is a simple and fast method. It does not require a lot of memory and can deal with

huge datasets. Also, the fact that many studies have used manually-set rule-based models gives the impression that a rule-based learning algorithm such as classification tree should be suitable.

Also an ensemble of Decision Tree classifiers (e.g., Random Forest) may improve the performance and make the model robust with respect to noise. This method is used to overcome generalization biases (Ho, 1995).It has been used in different studies (Witayangkurn et al., 2013; Montini et al., 2014) and has resulted in higher accuracy (Stenneth et al., 2011).

So far we have discussed Machine Learning methods with an emphasis on their use in mode detection. Transit itinerary inference has not been mentioned simply as a result of a paucity of research on the topic in general, and the use of Machine Learning methods in this context in particular. At the same time, conclusions about the above methods in the context of mode are also relevant in the context of transit itinerary inference. However, since so little is known about transit itinerary inference, a Machine Learning method whose behavior is well-known and understood is considered wisest in this context.

Chapter 3

Data

A number of data sources were used in this research. Travel mode data was collected as a part of a travel survey using a smartphone application, DataMobile, now known as Itinerum^{TM}. It was developed, and is maintained, in the Transportation Research for Integrated Planning (TRIP) Lab at Concordia University in Montreal, Canada. In October and November of 2014, 44,000 members of the Concordia community (students, faculty and staff) were invited by e-mail to download and install the application on their smartphones (Patterson and Fitzsimmons, 2016). The application included a short survey on participant socio-demographics and some information on frequent trips (e.g., transportation mode between home and the university). The application then ran in the background to collect GPS points automatically. The interval between records was set to 25m, although somtimes intervals were longer in the case of signal loss, e.g. in the underground Metro system. The collected data was then transferred via Internet to TRIP Lab servers.Transit itinerary data (described in Section 3) was collected by surveyors in the summer of 2016.

Each observation collected by Itinerum^{TM} is a vector $u = (i, p^*, \tau^*, m)$ representing a GPS point where the elements in each record include:

- i *(user-ID)*, a string including 32 characters (letters and numbers). It is unique for each participant and randomly generated to ensure the confidentiality of the data.
- $p^* = (\phi^*, \lambda^*)$ *(longitude and latitude)* are two numbers, each one with 6 digits after decimal point. They provide geographical location for a collected point in degrees.

Table 3.1: Example Itinerum^{TM} observations

Record	User-ID	Longitude	Latitude	Time-stamp	Mode
u^1	6A755F	-73.5607	45.5150	02/11/2014 8:22:10 AM	Bike
u^2	127BD2	-73.5416	45.5365	20/10/2016 4:37:22 AM	Transit

- τ^* *(timestamp)* includes date and local time at which a point is collected.
- m *(mode)* is a string representing the mode that the participant was using when the point was collected. Details on this are provided in Section 3 below.

Other information such as phone device estimates of speed, acceleration, altitude and as well as horizontal- and vertical-accuracies were also stored, but were not used in the analysis. During data collection for the transit itinerary detection study, surveyors were also asked to record transit route used during their trips. Therefore, each record in the corresponding the transit itinerary dataset has two more elements:

- q *(route)* is an integer number that identifies the transit route that was used when the point was collected.
- w *(ipere)* is an integer number that identifies the origin and destination of the trip based on the defined trips in the regional household Origin-Destination (OD) travel survey of Montreal in 2013 (see 3 below).

Table 3.1 presents a few examples of observations collected with Itinerum^{TM}.

A trace of GPS points on a map with two candidate transit routes for the trip are presented in Figure 3.1. The red dots on the map show the points collected with DataMobile. It shows a trip where the respondent has taken a bus to reach a metro station. The rest of GPS trace is lost until the respondent exited the metro station at Concordia University.

The second source of information used is the General Transit Feed Specification (GTFS) data that included information about the public transit network. This data was provided by Société de Transport de Montréal (STM) and is publicly available on the Internet [1]. It includes location of bus stops and metro stations and their daily service schedules. Daily schedules might not be same for

[1] https://transitfeeds.com/p/societe-de-transport-de-montreal/39

Figure 3.1: A public transit trip from the 2014 DataMobile study including candidate bus routes

different days of week. GTFS data usually consists of many comma-separated values (CSV) files (or tables), each including a type of information. There are three main files used in this study:

(1) **Trips**: This file includes a list of transit trips with their unique trip-IDs. Each record in this file is a vector ρ representing a unique transit trip. Many of the ρs might belong to a single transit route. The main elements of a vector ρ are:

- σ *(route-ID)* is a foreign key to identify transit routes.
- ψ *(trip-ID)* is the primary key for this file. Each scheduled service from beginning to the end of a transit route has its unique trip-ID. Two services are considered different and assigned different *trip-ID*s only if they pass different stops or arrive/depart the same stops at different times of the day.

Example records from the *trips* table are provided in Table 3.2.

Table 3.2: Example records in GTFS *trips* table

Record	**Route-ID**	**Trip-ID**	**Trip head-sign**
ρ^1	165	166f 165 b 166 a	Cote Des Neiges
ρ^2	435	166a 435 n 165 b	Express Cote Des Neiges

Table 3.3: Example records from the GTFS *stops* table

Record	**Stop-ID**	**Stop lon.**	**Stop lat.**	**Stop name**
ν^1	627	-73.4451	45.3256	Cote Des Neiges - Van Horne
ν^2	10014	-73.5364	45.5466	Sainte-Catherine/d'Orléans
ν^3	7	-73.5978	45.5312	Station Rosemont

(2) **Stops**: This file contains the location of metro stations and bus stops. Each record is a vector $\nu = (\kappa, p)$ where $p = (\phi, \lambda)$ and:

- κ *(stop-ID)* is an integer that is the primary key in the file to identify stations and stops.
- ϕ *(longitude) and* λ *(latitude)* are as explained before in Table 3.1.

Table 3.3 shows example records from the *stops* table.

(3) **Stop-times**: Contains the scheduled arrival/departure times of transit vehicles to/from stops for all transit trips. Each record is a vector μ representing an arrival and departure of a transit vehicle for a single transit trip at a station or stop. The main elements in μ are:

- ψ *(trip-ID)* is a foreign key in this file.
- κ *(stop-ID)* is a foreign key in this file.
- τ *(arrival/departure time)* shows the time of day that a transit vehicle performing a given transit trip is supposed to arrive at/depart from a specific stop. This is a schedule, and the duration of time spent at stops is relatively short; both the arrival and departure times are the same.

Example records in *stop-times* are shown in Table 3.4.

Table 3.4: Example records in the GTFS *stop_times* table

Record	**Trip-ID**	**Departure time**	**Stop-ID**
μ^1	166f 165 b 166 a	2016/06/15 13:45:52	627

The last dataset used is a map of city zones. It was used to limit the queries on the trips and transit routes in the same zone and decrease run-time for data processing. The main elements in this dataset are:

- *zone-ID* is the primary key in this file.
- *boundary* is a closed line-string that represents boundary for each zone

Mode detection data

For the mode detection, only points associated to trips with validated transportation mode were used since the supervised methods require fully labeled training data.

Participants declared their main and alternative mode of transportation in their regular daily trips. To validate the mode used in the trips, only participants that declared a single transportation mode were chosen and their corresponding trips were used in the study. Based on this criterion, 788 trips with validated transportation mode provided by the participants were selected. (While this is a relatively small dataset, the intention is to test inference methods that could be more easily scaled, that is semi-supervised methods.) Transportation modes included walk (212 trips), bike (14 trips), car (49 trips), transit (453 trips), car & transit (28 trips) and shuttle (32 trips).

Trips were defined by their start and end points, but were not broken into segments (i.e., sections of a trip, with their own mode of transport). The dominant transportation mode (by distance traveled) was assigned to the entire trip. This was a constraint related to how the data were labeled when validated by respondents,i.e. they were asked to assign a mode for their entire trip.

Transit Itinerary detection data

For transit itinerary detection, student surveyors were recruited for data collection. These surveyors were asked to carry iPhones with DataMobile installed while undertaking predefined sequences of trips. The first trip of each sequence was assigned randomly and the next ones were chosen in a way that the distance between the end of a trip and the beginning of the next one did not to exceed 750m. If there was more than one candidate trip starting within this range, one was chosen randomly (Zahabi et al., 2017).Trips were chosen from the regional household Origin-Destination

(OD) survey for Montreal in 2013 (Agence métropolitaine de transport, 2013). To collect validated GPS data, but at the same time have a realistic simulation of those trips, decisions about which transit mode or route to take in each trip segment was made by the surveyor. For validation, surveyors recorded their boarding and alighting times and also transit line numbers (Zahabi et al., 2017) and associated them to the collected point *ex post* using GIS software. The dataset includes 390 trips.

Chapter 4

Semi-supervised vs. Supervised Mode Detection

4.1 Methodology

4.1.1 Data Preprocessing and Feature Selection

The data collected by smartphones consisted of observations, each of which represented a single GPS point. The first step was to aggregate GPS points into trips and to derive trip features. The result was a randomly ordered array of $(u^1, x^1, y^1), ..., (u^{702}, x^{702}, y^{702})$, each representing a trip $s \in \{1, ..., 702\}$, $u^s = (u^s_1, ..., u^s_{r(s)})$ is the time-ordered sequence of time stamped points of the trip and $r(s)$ is a function of trip s, equal to the number of GPS points collected in the trip. The i^{th} time stamped point of the trip s is $u^s_i = (u^s_i, \tau^s_i)$ where τ^s_i is the time stamp associated with the GPS point, u^s_i. Also x^s is the feature vector of the trip s and y^s is its corresponding label that is the mode of transportation used in the trip s.

Since the main purpose of this thesis was proposing the application of a semi-supervised learning method rather than improving the current algorithms, only a few simple features were used. Let P be the set of points representing all bus stops and metro stations in the network and $D(a, b)$ be the distance between the points a and b. For a trip s with $u^s = (u^s_1, ..., u^s_{r(s)})$, a feature vector x^s was derived with the following features as its components:

- t, total trip travel time in minutes:

$$t^s = \tau^s_{r(s)} - \tau^s_1 \tag{1}$$

- v, average trip speed in $^{km}/_{h}$:

$$v^s = \frac{D(u^s_1, u^s_{r(s)})}{(\tau^s_{r(s)} - \tau^s_1)} \tag{2}$$

- l, trip length in meters:

$$l^s = \sum_{i=2}^{r(s)} D(u^s_i, u^s_{i-1}) \tag{3}$$

- d_O, The minimum distance between the origin of a trip and the closest transit stop in meters:

$$d^s_O = \min_{p \in P} D(u^s_1, p) \tag{4}$$

- d_D, The minimum distance between the destination of a trip and the closest transit stop in meters:

$$d^s_D = \min_{p \in P} D(u^s_{r(s)}, p) \tag{5}$$

Features were calculated in PostgreSQL [1], which is a relational database management software. It has an add-in named PostGIS [2] designed specifically to manage GIS operations.

For the aim of feature selection, a correlation matrix was used to remove attributes with high correlations. Among the proposed features, only origin-destination distance and length of trip were highly correlated (> 0.6). Therefore, direct distance was excluded in the model training step. Table 4.2 shows examples of the values for the feature vectors, x^s, and their associated labels, y^s.

[1] https://www.postgresql.org
[2] https://postgis.net

Table 4.1: Correlation matrix between features in the dataset

Attributes	v	t	l_a	l_b	d_O	d_D
v	1	-0.08	0.39	0.39	-0.09	0.20
t	-0.08	1	0.23	0.24	0.03	0.10
l_a	0.39	0.23	1	0.97	-0.06	0.57
l_b	0.39	0.24	0.97	1	-0.06	0.56
d_O	-0.09	0.03	-0.06	-0.06	1	0.02
d_D	0.20	0.10	0.57	0.56	0.02	1

Table 4.2: Example of the dataset used to train the models

x_1 (v)	x_2 (t)	x_3 (l_b)	x_4 (d_D)	x_5 (d_O)	y (mode)
4.4	1247	5487	43	104	transit
1.7	824	1410	400	32	car

4.1.2 Modelling

In this step, trip features were used to train the mode detection models. To do the comparison, six simple and widely used algorithms were chosen: Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), Naïve Bayes (NB) and Multi-Layer Perceptron (MLP) as supervised methods, and Label Propagation (LP) as a semi-supervised method. However, NB and MLP resulted in low accuracies and in some cases they did not even converge. Therefore, they are not present in some figures and comparisons and the four other algorithms were used as the main algorithms in the comparisons.

For each method, in order to estimate the best model and to see the sensitivity of the model with respect to its meta-parameters, different models were estimated by changing the meta-parameters. This is explained in detail in the next section where the results of the sensitivity analyses are provided. The data was randomly split into a training set (80%) and a test set (20% of observations) and then the models were trained and tested. For a given set of values for meta-parameters, this process was repeated 100 times using different random splits in order to capture the effect of training sample on the model. This allowed the calculation of the average accuracy together with the variation of accuracy due to changes in training and test sets.

The novelty of this study is testing the use of a semi-supervised method for mode inference from GPS surveys and how they compare with traditional supervised methods using different proportions of validated data. Therefore, a very important issue is to compare the semi-supervised method with

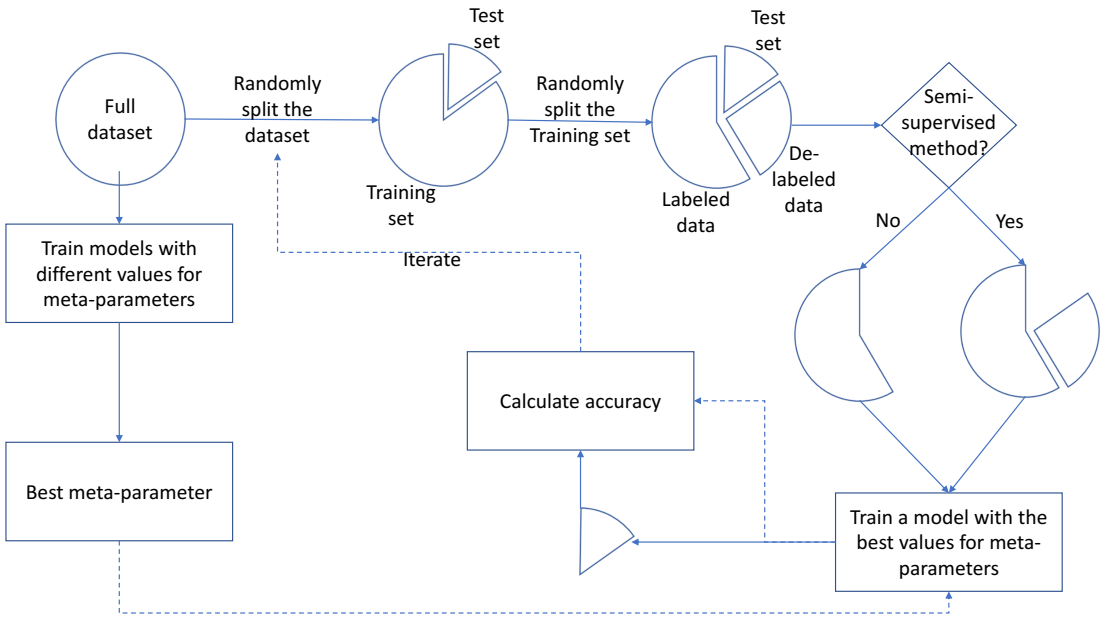

Figure 4.1: Dataset split into training and test set and de-labeling a portion of training set

other methods considering different scenarios for the portion of unlabeled data. Thus, even for the supervised methods, after finding the best settings using the entire dataset, these settings were used to estimate models on different portions of de-labeled data. After splitting the data into training set and test sets, labels for a portion of the training set were removed. This is visualized in Figure 4.1. Obviously, the supervised algorithms could only use the portion of training set that still had labels, but the semi-supervised algorithm could use all the observations in the training set to make a model.

Decision Tree for mode detection

The algorithm used for growing trees in this study is recursive partitioning. In this method, the training set of observations is split into subsets by putting thresholds on the features to have subsets with mostly homogeneous labels. Also to avoid over-splitting the training set into too many subsets, the Decision Tree needed to be pruned. The minimum number of observations in the nodes and cost-complexity was considered for pruning in this study. Details of recursive partitioning and pruning can be found in Therneau et al. (Therneau et al., 1997).

Random Forest for mode detection

The second method used in this study is an ensemble algorithm that generates several trees based on random samples (with replacement) from the training set. While generating each tree, this algorithm picks the best split from a subset of candidate features rather than the best split among all features (Breiman, 2001). Selection of the best split from a subset of features rather than all of them adds a slight bias in the predictions of the classifiers, but in return slightly decreases the variance. The main parameter of this algorithm is the number of generated trees.

The only difference in the algorithm used in scikit-learn package in Python used in this study with the one described by Breiman is that the algorithm embedded in scikit-learn uses average of probabilistic predictions of classifiers rather than taking votes. The developers of this package have not explained the reason behind their decision to use a modified version of Breiman's algorithm. One possible explanation is that only Breiman's algorithm assigns the frequency of estimates among classifiers as a probability of each particular estimate. On the other hand, the modified version gives more information such as average, variance and distribution of probability for each particular estimate using the probability values provided by each classifier.

Logistic Regression for mode detection

In the terminology of statistics this model is called regression, however it is usually used for classification purposes. It uses a sigmoid function and takes the linear combination of input variables to regress a probability for each class. The class with the highest probability is the estimation of the model (Bishop, 2006).

Label Propagation for mode detection

As a semi-supervised algorithm, Label Propagation with the K-Nearest Neighbors (KNN) kernel was used. The algorithm is described in detail by (Delalleau et al., 2005). The Radial Basis Function (RBF) is another popular kernel used in label propagation in which all observations contribute to changing the label for each observation based on their distance. But KNN is simpler and faster, using a sparse matrix to represent the relations between the observations compared with the RBF

that uses a fully connected graph represented in memory with a dense matrix. Therefore KNN was picked as the kernel to keep the model simple and fast (Zhu and Ghahramani, 2002).

The similarity measure used here for the KNN kernel is Euclidean distance. A main parameter for the algorithm is the number of neighbors considered for each observation. The algorithm also gets a clamping factor (α) which defines the portion of labeled observations that are allowed to be changed. Also, to check the stability of results due to the change in the proportion of labeled and unlabeled data in the training set, the experiment was repeated with different proportions of unlabeled data in the training set.

Algorithm 1 below describes the steps of label propagation with the KNN kernel. The first $m + n$ observations are used as the training set where the first m observations are used as labeled and the labels for the next n observations are not used to simulate unlabeled observations in a semi-supervised learning algorithm. The rest of the observations excluded from the training set are used as test set.

input : $(x^1, y^1), ..., (x^{702}, y^{702}), \epsilon, \alpha$
output: $\hat{y^{m+n+1}}, ..., y^{702}$
Randomly assign labels y^s for $s \in \{(m+1), ..., (m+n+1)\}$
while *portion of labels updated in the last iteration among* $s \in \{1, ..., m+n\}$ *more than* ϵ **do**
  while *portion of labels in updated in the last iteration among* $s \in \{1, ..., m\}$ *less than* α **do**
    for $s \in \{1, ..., m+n\}$ **do**
      Estimate the label y_s by taking vote from K nearest observations to s and update the label if the previous estimation is different;
    end
  end
  for $s \in \{n+1, ..., m+n\}$ **do**
    Estimate the label y_s by taking vote from K nearest observations to a and update the label if the previous estimation is different;
  end
end

Algorithm 1: Label propagation algorithm

4.2 Results

Since there are four methods compared in this study and each of them was tested in many different scenarios and compared with the others, use of confusion matrices would make evaluation

of the different methods difficult due to the high number of values to be compared together. Thus, to keep the comparisons more clear and readable, accuracy of prediction was used as the measure of performance.

4.2.1 Sensitivity to Sampling

The first sensitivity analysis was for the variation of accuracy (on the test set) with respect to a change in the training sample. All the four models were estimated in Python using the package sklearn (Pedregosa et al., 2011). The Decision Tree model was estimated without pruning, the Random Forest generated 30 classifiers and label propagation used 11 as the number of neighbors for its KNN kernel and was fully clamped (no change in the real labels). With these settings and repeating the modeling 100 times, each time splitting the data into an 80% training set and a 20% test set, the Random Forest shows a higher average accuracy of the model according to Figure 4.2. Also it has the lowest variance compared to DT and LP (based on these settings).

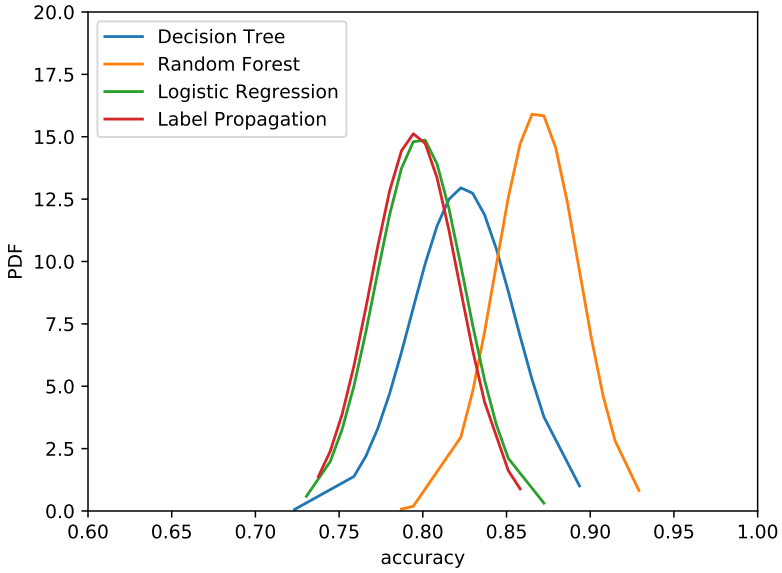

Figure 4.2: Normal distributions fitted to the results of cross-validated accuracy for DT (without pruning), RF, LR and LP methods with iteration on randomly split of dataset into training and validation set

Regarding Figure 4.2, accuracy values are sensitive to changes in the training set. To avoid

the effect of training set selection while modeling given a setting, the dataset was split randomly one hundred times into training and test sets. Each split resulted in a model and its accuracy was based on the corresponding test set. The average of one hundred accuracy values was reported as the accuracy of models with the corresponding setting and this was included for the analyses and comparisons.

4.2.2 Decision Tree Meta-parameters

Decision Trees can grow until there is either only one observation in each leaf or a group of observations with the same label. To prevent over-fitting of the model to the training set, pruning is required[3].

To prune the model, the complexity cost ranged from 0.000001 to 100. The results are shown in the Figure 4.3. The model was fitted 100 times to different random samples, the plot result does not look smooth. Nevertheless, the graph shows a drop in accuracy with an increase in the complexity cost factor. This was expected because higher complexity cost results in a more pruned tree, which would be weaker for explaining instances.

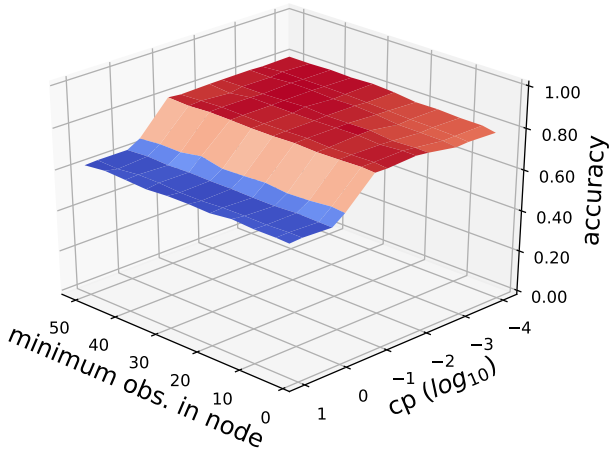

Figure 4.3: Sensitivity of accuracy for DT method with respect to complexity cost and minimum number of observations in leaves

[3]Unfortunately neither sklearn, nor any other package in Python provides pruning. Therefore for this purpose, the package rpart in R was used.

As shown in the figure, accuracy is convex with respect to the logarithm of CP for the lower values of CP. After a turning point, accuracy is concave with respect to logarithm of CP for the higher values of CP. Since the axis in the figure corresponding to CP is logarithmic scaled, the relation between accuracy and CP may either be generally similar (i.e., with a turning point, convex in the beginning and concave in the end) or totally concave. Regardless of concavity and convexity, accuracy is strictly descending with respect to CP and the logarithm of CP. The other meta-parameter that is the minimum number of observations in nodes does not have much effect on accuracy. Based on Figure 4.3, we chose $CP = 0.01$ to prune the Decision Tree for comparison (last section).

The final Decision Tree made with $CP = 0.01$ is shown in Figure 4.4. The boxes represent nodes in the tree and the circles are its leaves. The label that has the largest number of observations in each node or leaf is written on the node or leaf. For example among all (788) observations, 453 of them are transit trips, the dominant label for the root (i.e., node on the top) is "Transit".

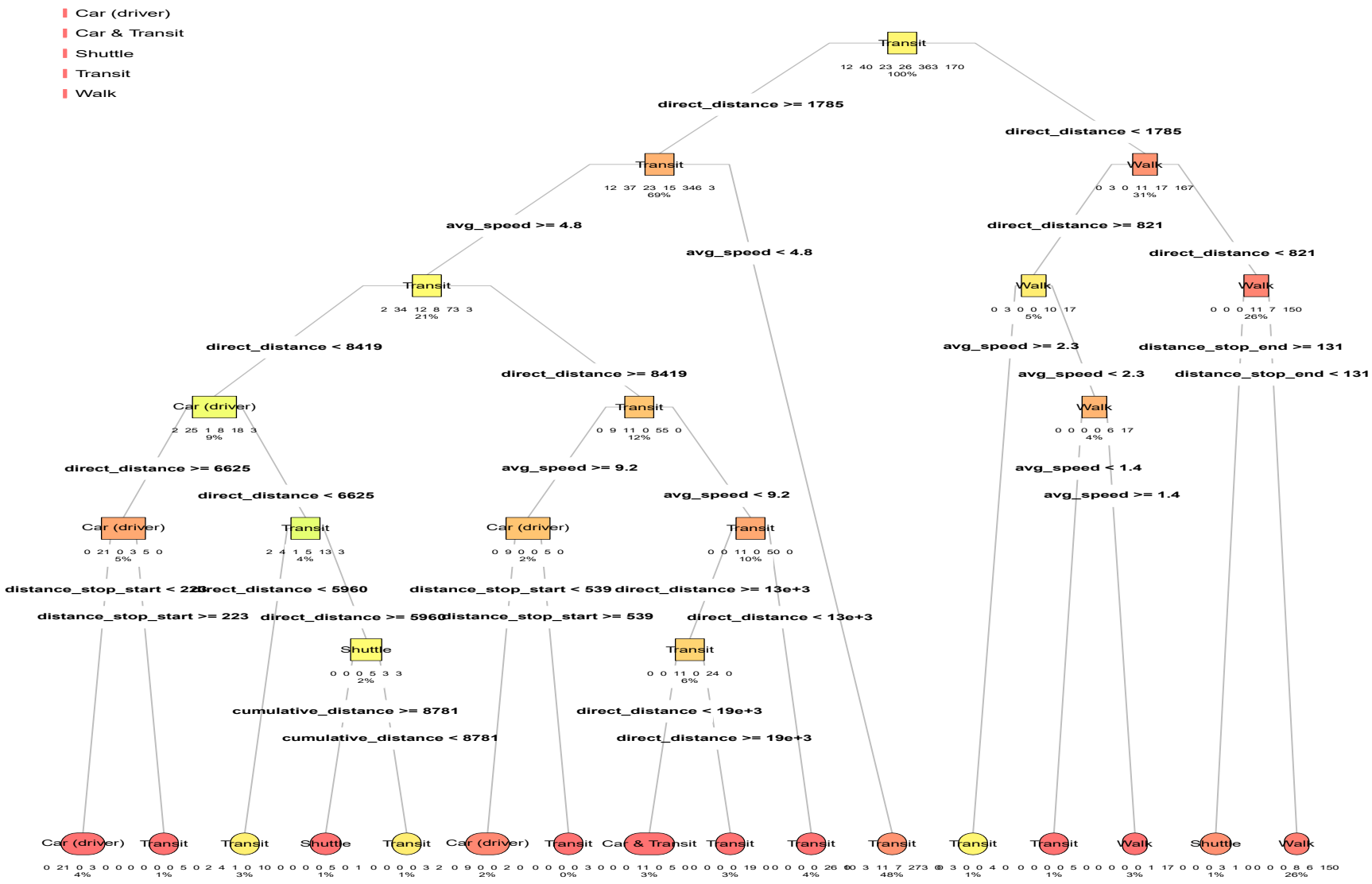

Figure 4.4: An example of pruned Decision Tree

4.2.3 Random Forest Meta-parameters

The main parameter for the Random Forest is the number of trees generated. Figure 4.5 presents the average accuracy of models for different number of classifiers. As was expected, the increase in the number of classifiers improved the accuracy of the model. According to the figure, accuracy converge quickly with increases in the number of classifiers. This could be caused by the double randomization embedded in the algorithm including the random selection of the training set for each classifier and choosing best split based on a random subset of features. Based on the graph, we chose 10 as the number of classifiers for the RF method.

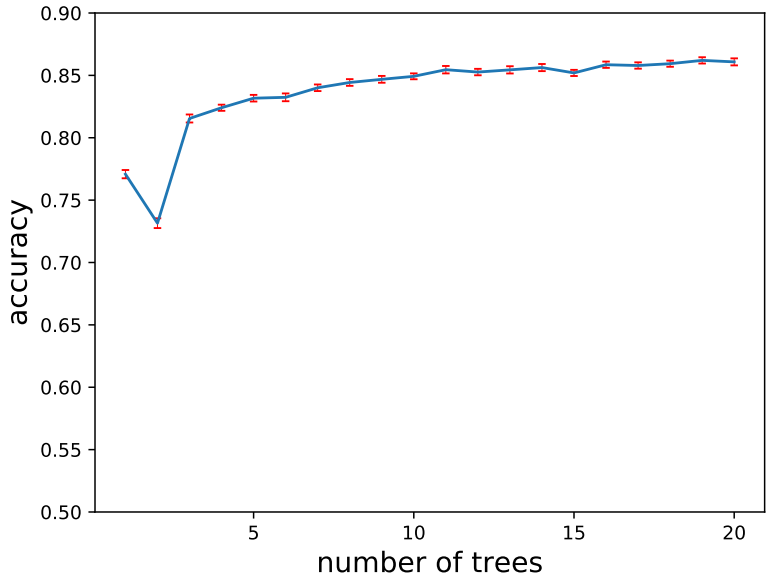

Figure 4.5: RF sensitivity to number of trees

4.2.4 Logistic Regression Meta-parameters

This model takes only input variables, but no meta-parameters. Therefore no meta-parameter assessment was needed.

4.2.5 Label Propagation Meta-parameters

The first parameter to check for this method was the clamping factor that represents the minimum portion of labeled observations kept in the model. It seems that apart from values very close to zero, the parameter does not have a considerable effect on the accuracy (test set) of the model. We kept it equal to 1 (fully clamped) in the rest of LP models.

The second parameter of the LP model was the number of neighbors, which was actually the parameter of the kernel. The LP model is supposed to work with different portions of unlabeled data. Because the first experiments showed the importance of k on the performance of the model, its effect on the accuracy of the model was observed under different scenarios with respect to the portion of unlabeled data.

Figures 4.6 show the average accuracy of the LP models for different values of α and K. In each figure, the blue surface represents the average accuracies for the models on training sets and the red surface represents average accuracies for the test sets. To better visualize the accuracy results for test sets (cross-validated results), a contour is provided that is a 2D projection of the red surface on the surface $z = 0$.

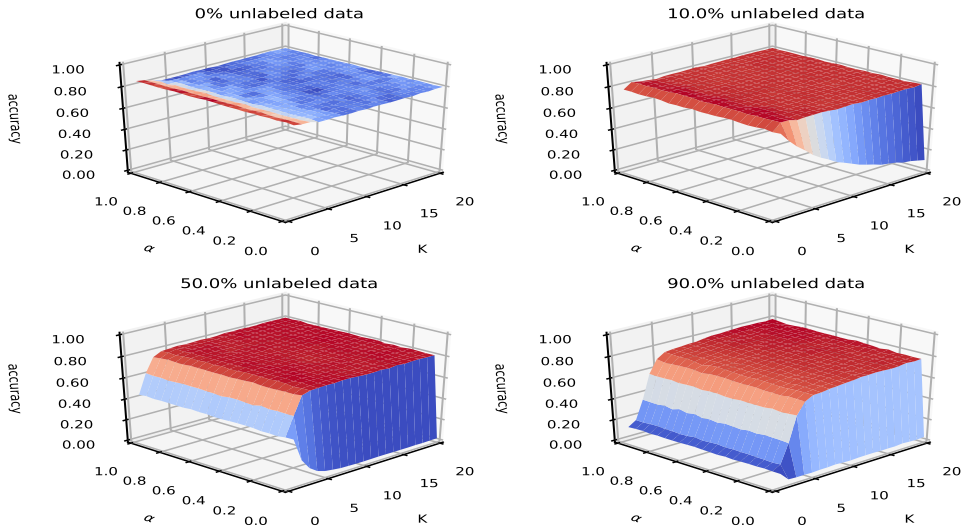

Figure 4.6: Sensitivity of accuracy for Label Propagation method with different portions of unlabeled data (*i.e.*, $\frac{n}{m+n}$) with respect to K and clamping factor

As seen in these figures, fewer neighbors resulted in better models with almost fully labeled (validated) data. But as the proportion of labeled data was reduced, the smaller number of neighbors lost their advantage and a large number of neighbors (K) provided superior results. Because this method is usually adopted in situations with small proportions of validated data, it might be better to decide upon the best K from the plots with less labeled data. Based on the plots below, while the values below 5 for K performed better in mostly labeled situations, they all performed worse than values higher than 5 in situations with more than 20% unlabeled data. Values above 5 barely showed a change in their performance in the different scenarios. Thus, choosing a value (such as K=12) with more reliable performance might be a smart choice.

4.2.6 Comparison Between the Models

The last step of the analysis was to compare the models with the meta-parameters chosen from the previous steps. To compare the algorithms, a portion of the training set was randomly de-labeled (labels were removed from he training set). 100 models were estimated using each algorithm for each de-labeling portion based on 100 different randomly split data into 80% training set and 20% test set. The de-labeling factor for the training set ranged from 0% (fully validated data) to 90%. The steps are explained in Algorithm 2. The Python code for this algorithm is included in Appendix A.

Figure 4.7 shows the average accuracy of algorithms under the different data validation scenarios. In this plot, a change in x-value does not change the total sample size but instead the number of unlabeled trips in the sample. Thus, for label propagation, the size of training set remains the same but the proportion of unlabeled and labeled trips changes. The size of training set for the supervised algorithms reduces with an increase in the number of unlabeled trips since they cannot use unlabeled observations for training a model.

Based on this plot, the RF model has the highest prediction accuracy for most of the scenarios. However, as the portion of unlabeled data increases, its prediction accuracy gets closer to the accuracy of the Label Propagation model. The LP model starts to outperform the RF model when the proportion of unlabeled data is higher than 80%. LP keeps taking advantage of unlabeled data, while RF does not use the valuable information provided by unlabeled (un-validated) data. Average

Data: $\{(x^s, y^s)|s \in S\}$
Result: c: a vector of accuracies with 10 elements for each algorithm
for *Each algorithm* **do**
  for $i \in \{0, 50, 85\}$ **do**
    Define c as a vector with no element;
    for $j \in \{1, ..., 100\}$ **do**
      Split the S randomly into 80% as S_t and 20% as $\overline{S_t}$;
      Split the S_T randomly into $j\%$ as S_t^u and $(100 - j)\%$ as s_t^l;
      if *DT, RF or LR* **then**
        Train a model on $\{(x^s, y^s)|s \in S_t^l\}$;
      else if *LP* **then**
        Train a model on $\{(x^s, y^s)|s \in S_t^l\}, \{x^s|s \in S_t^l\}$;
      Calculate the accuracy of the model on test set;
    end
    Add the average of accuracies (on test sets) for $i\%$ unlabeled training set as the last element of c;
  end
end

Algorithm 2: Sensitivity analysis on portion of unlabeled data.

standard deviations are shown in the figure by vertical bars on the lines. The values for the average standard deviations appear close to each other, which is consistent with Figure 4.2. At the same time, slight increases in the values are observed when average accuracy is decreased. This shows that when a model has lower accuracy, the estimates are more sensitive to training set selection.

The Decision Tree, Random Forests and Logistic Regression, the supervised methods mostly perform as well or better than the semi-supervised LP method even when a considerable portion of the data is unvalidated, that means the supervised algorithms are using a considerably smaller portion of the data for training compared to the semi-supervised algorithm. However, when there is more than 70% unvalidated data, the semi-supervised algorithm performs better than the others and has the highest cross-validated accuracy. It is an empirical question whether it is reasonable to expect being able to collect 30% validated data in a real-world data gathering exercise.While it may be possible to have higher portions of precisely validated data in a big survey, every increase in the proportion of validated data will likely result in increased cost.

To give another perspective on the performance of the different models, Figures 4.8, 4.9 and 4.10 are provided. They show the average accuracy of the models with respect to the size of labeled and unlabeled training sets rather than the proportion of labeled and unlabeled data in the training

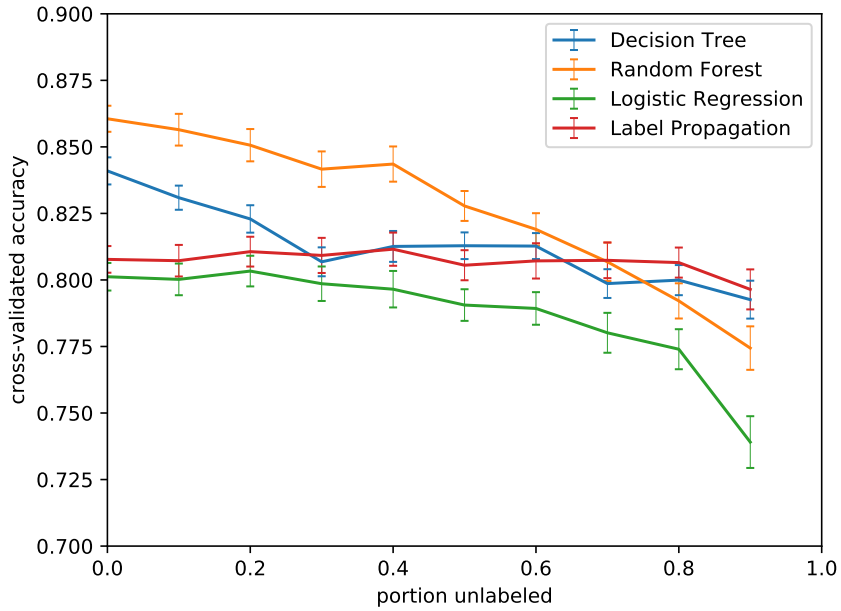

Figure 4.7: Comparison of average accuracy of the models for different portions of data as unlabeled (DT with CP=0.001, RF with 10 trees, fully clamped LP with K=20)

sets. Each figure shows the results for a constant number of labeled observations in the training set, while the number of unlabeled observations in the training set changes.

In these figures, it is not surprising to see that the accuracy of the supervised models does not change with the change in number of unlabeled observations, since these methods do not use unlabeled data for training. The semi-supervised model, however, takes advantage of both labeled and unlabeled observations. As such, we see that given a fixed number of labeled observations, the average accuracy increases with the rise in number of unlabeled observations.

It needs to be considered that the run-time for DT, RF and LR was much longer compared to the LP method. Although run-time is directly related to the parameters of the models (especially for the RF which is very sensitive to the number of trees), given the parameters used here (10 trees for RF and K=12 for the fully clamped LP), the run-time of the RF model was almost 16 times longer than the LP model (8' 20" compared to 30" to produce the curves in the Figure 4.7, each consisting of 1100 models)[4]. While these run-times are not long, for a real-world-sized survey with millions

[4]The algorithms were run on a 64-bit Windows 7 machine with Intel™ Xenon™ X5650 CPU and 12 GB of RAM. The RF and LP algorithms were run in Python 64-bit while for the DT algorithm, R 64-bit was used. Thus, only the RF

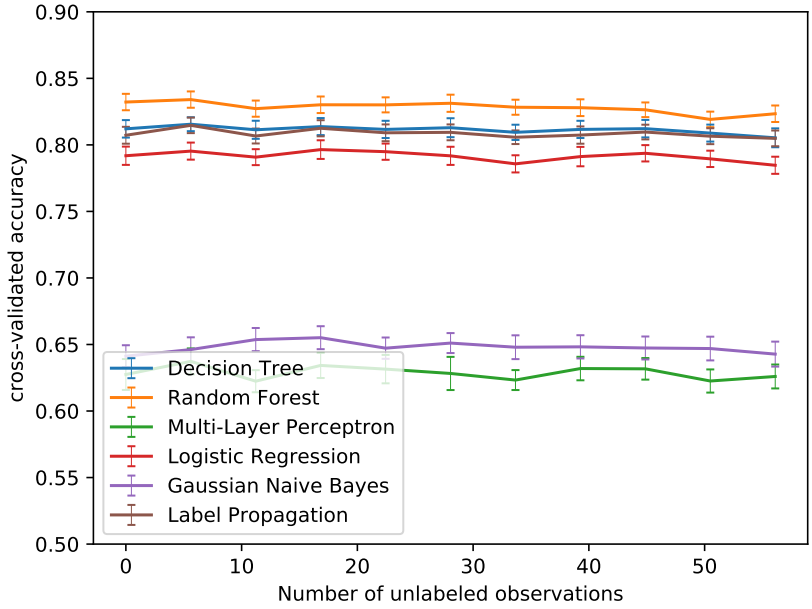

Figure 4.8: Comparison of average accuracy of the models for 570 labeled observations (DT with CP=0.001, RF with 10 trees, fully clamped LP with K=20)

of trips, the difference would be noticeable (especially because KNN run-time, which is used in LP algorithm, is less sensitive to sample size compared to RF).

In this section, the application of semi-supervised and supervised algorithms was studied. The results show that the semi-supervised model has a higher accuracy compared to the supervised models when more than 70% of the observations in the data set are un-labeled. This is due to the fact that the semi-supervised algorithm can take advantage of un-labeled observations in the data set.

and LP can be compared based on run-time confidently

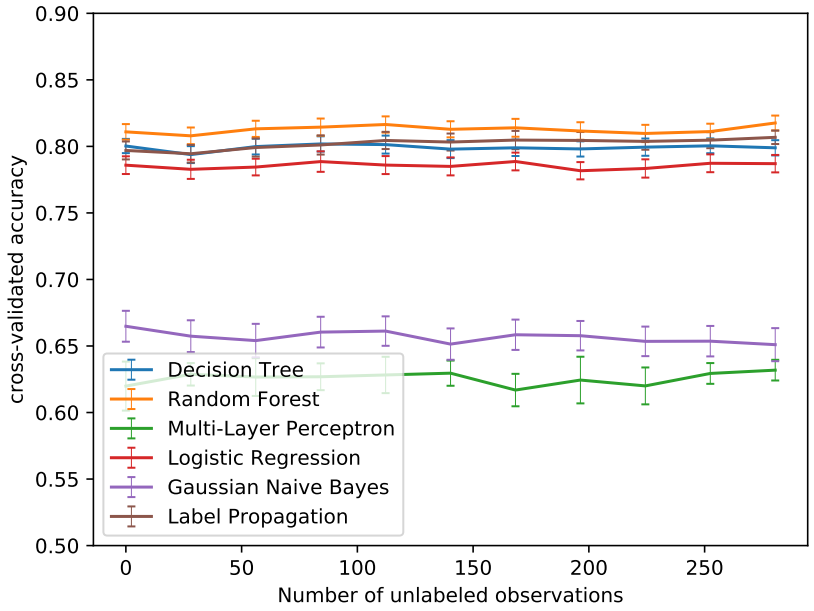

Figure 4.9: Comparison of average accuracy of the models for 315 labeled observations (DT with CP=0.001, RF with 10 trees, fully clamped LP with K=20)

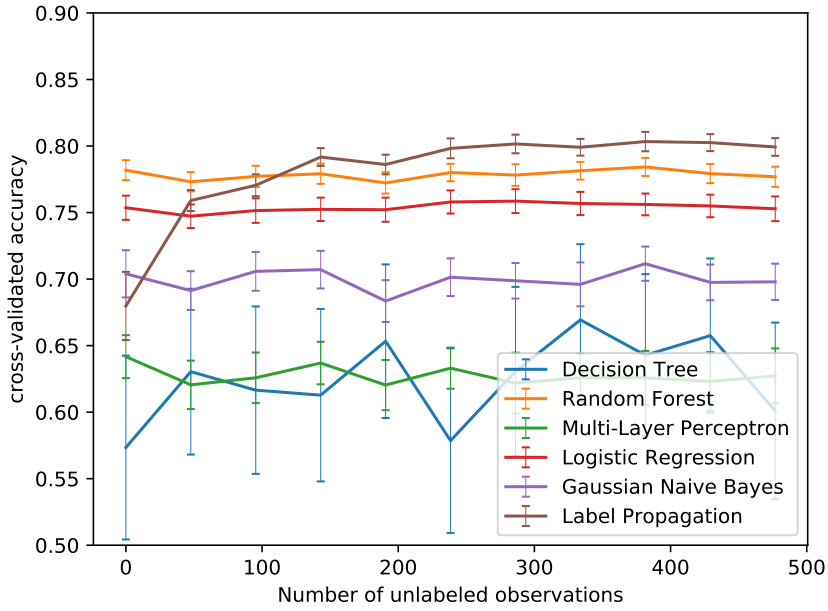

Figure 4.10: Comparison of average accuracy of the models for 100 labeled observations (DT with CP=0.001, RF with 10 trees, fully clamped LP with K=20)

Chapter 5

Transit Itinerary Detection

5.1 Methodology

In order to infer transit route used by respondents, GPS points associated with trips were compared with GTFS data. First, using longitude, latitude and stop departure times, each departure was represented as a point in 2+1 dimensional space-time. Then 3D bus itineraries were made by connecting these points for every scheduled bus trip. Also from data collected with DataMobile, GPS points and their associated timestamps were used to map respondent traces in space-time. Figure 5.1 shows a set of GPS points for a hypothetical trip and a nearby transit route in space-time. The x- and y-axes represent geographical location and the t-axis represents time.

Two algorithms were developed for transit itinerary detection. The first algorithm assumes that start and end point of segments of trips are provided *a priori*. The second transit itinerary detection algorithm estimates the start and end point of each segment of a trip and detects its transit route simultaneously.

Trips were broken into segments where each segment was a section of trip with only one busline being used in it. The first transit itinerary detection algorithm breaks each trip into segments using given start and end point of segments. The second transit itinerary detection algorithm selects subsections of transit routes close to each trip. For each trip, these subsections are considered as its candidate segments. Then the algorithm estimates whether each candidate segment and its corresponding transit route is a true segment and transit route of the trip. Further details are available

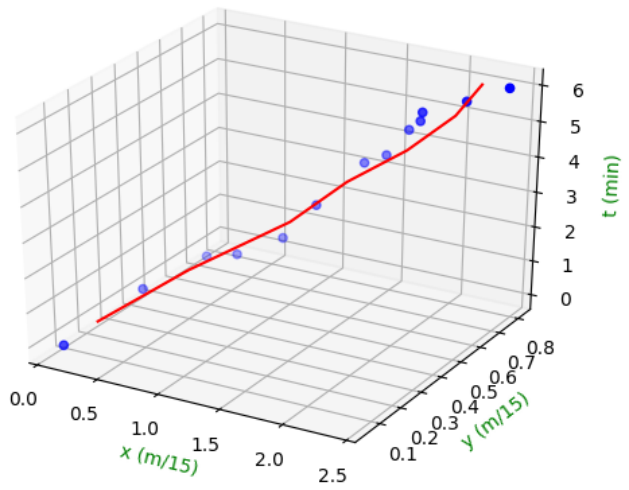

Figure 5.1: GPS records and a candidate transit route in a 2+1 dimensional space-time

in Appendix B.

Then, points for each segment were checked with the bus-lines to see if they fell within a buffer around the lines. Since the data used for buses was a schedule and could be different from real operating times, the spatial distance of GPS points with bus lines was used. If any of the points for a segment was close enough to a line, the line was added to the set of possible transit routes for the segment.

For every possible transit route for each segment, a number of features were calculated. These features include average spatial distance of GPS points with bus-line, average Euclidean distance of GPS points with bus-line in space-time, the number of GPS points in the vicinity of bus lines in space-time and the portion of segment covered by bus-line. Then the dataset was split into an 80% training set and a 20% test set and a Random Forest model was made using these features. The model used features for the observations in the training set to estimate whether a segment and bus-line were matched (bus-line was used in the segment). A segment was considered as incorrectly estimated if any wrong bus line was estimated for the segment or the true bus line was not estimated for the segment.

To detect transit routes in the segments, the GPS points and GIS information related to the transit network needed to be processed to extract more explicit information (features). Features were then used as input data for the Random Forest model to detect if a possible route was used for a segment or not.

5.1.1 Pre-processing

Four tables were built using collected GPS points and GTFS data. Information about transit routes was summarized in table T_1 from stop-times and stops tables in GTFS files. Each row in T_1, r, represents a route in the transit network.

- r_1 *(route-ID)* is a primary key in this file.
- r_2 *(route-line)* is a linestring made by connecting the sequence of stop locations for an arbitrary transit trip for a given transit-route.

A transit route can be projected on a map as a sequence of stop locations regardless of arrival times. T_2 is another table built based on GTFS data. Each row in T_2, b, represents a trip of a transit vehicle during a specific period of time, t, that can be projected in space-time. For each transit route in T_1 depending on the service frequency of the route, there would be a number of transit trips in T_2.

- b_1 *(trip-ID)* is a primary key in this file.
- b_2 *(trip-line)* the sequence of points in space-time from the GTFS *stop-times* table for a given transit trip.

To speed up the queries by checking trip segments only against the transit routes in the same neighborhood, table zone-route was built. For zone z intersecting with m_z routes, there are m_z rows in table T_3 containing the zone-ID and route-ID.

- z_1 *(zone-ID)*
- z_2 *(route-ID)*

The last table created is T_4. To create this table, for each person, p, his/her sequence of GPS points was selected as l_p, ordered by their timestamps. Then, for each route in T_1, the longest sequences of GPS points (possible segment $q^{p,r}_{t_s,t_e}$) in l_p was selected and added to T_4 where all GPS points in $q^{p,r}_{t_s,t_e}$ are close enough to route r and t_s and t_r are timestamps for the first and the last GPS points in each sequence respectively. $q^{p,r}_{t_s,t_e}$ is a possible segment of the trip because this section of a trip by person p can be expected to be done using only transit route r. Also, given a person and a route, no GPS point can appear in more than one sequence, and for any pair of sequential GPS points in $q^{p,r}_{t_s,t_e}$, there would be no more than one GPS point u excluded from $q^{p,r}_{t_s,t_e}$. Each record in T_4, q, is a vector of elements:

- q_1 *(User-ID),* q_2 *(start-time) and* q_3 *(end-time)* are elements that together represent a section of trip of a person, s.
- q_4 *(Route)* is the real transit route associated with section s. It is the most repeated value of v_6 for all $v \in DataMobile$ that v belongs to s.
- q_5 *(Route-ID)* represents the transit route r that is close to the section s and is a candidate transit route for being the real transit-route.

The process to make Table T_4 is described in Algorithm 3:

A Random Forest model was estimated based on the table T_4 where entries in T_4 were fed into the algorithm as observations. The Python code for the whole algorithm is included in Appendix B.

5.1.2 Creating features

For each possible segment $q \in T_4$, a set X^q was made, including features of the possible segment s with respect to route r. These features represent some characteristics of the GPS points in a possible segment such as average distance between GPS points and some mutual characteristics of recorded GPS points and the nearby transit route, such as average distance of GPS points with the transit route. The final features used to estimate the models are:

- x_1 *(metro-boolean)* is a boolean value indicating whether the candidate transit route r is a metro line.

Data: $\{(x^s, y^s)|s \in S\}$
Result: c
for $r \in T_1$ **do**
  From T_3, find the zones that r falls within them;
  for $b \in T_{2,r}$ **do**
    for *GPS point* u *in the same zone as* r **do**
      if $D(u, b) < 100$ **then**
        Add u to a temporary table T_6
      sort the GPS points in T_6 by corresponding user and time-stamp.;
      Make a vector with the time-ordered sequence of GPS points for each user and add it to T_4.;
      for $q \in T_4$ **do**
        for $i \in \{1, \cdots, \|q\|\}$ **do**
          if *There is more than 1 hour difference between* u_i *and* u_{i+1} *or more than 2 points recorded in the time interval between them (excluded from* T_4*)* **then**
            Break the q into to vectors, one ending with u_i and the other one starting with u_{i+1}
        end
      end
    end
  end
end

Algorithm 3: Preparing the input table T_4 for the learning algorithms

- x_2 is the length of section s.
- x_3 is the average distance of points belonging to section s from the candidate route r in time-space.
- x_4 is the average geographical distance of points belonging to section s from the candidate route r on the map.
- x_5 is the average distance between sequential points in S.
- x_6 is a combination of the previous features compared to the other alternative routes. Let c^q be the combination of features for the record q in T_4, calculated as below:

$$c^q = x_1^q/(x_2^q * x_3^q) \tag{6}$$

Then for all $q' \in T_4$ that $q_1 = q'_1$, $q_2 < q'_3$ and $q_3 > q'_2$ we have:

$$x_6 = c^q / \max_{q'} c^{q'} \tag{7}$$

5.1.3 RF model

Observations in T_4 were split into a training set and a validation set based on which participant and day to which they belonged. 80% of participant-days were picked as the training set and 20% as the validation set. A Random Forest model was estimated on the training set and it was used to estimate values for the dependent variable on the validation set. For each GPS point u belonging to a person-day for the validation set, if all of the observations in the validation set are associated with the correct transit line, the GPS points are considered as correctly estimated.

Let Γ^s be the set of $\forall \pi_i^s \in \pi_s$ where π_i^s and π_{i+1}^s are correctly estimated points. The accuracy of a model over the test set is defined as:

$$accuracy \ = \sum_{s \in \{m+n+1,\ldots,702\}} \frac{\sum_{i \in \Gamma^s} \ D\left(\pi_i^s, \pi_{i+1}^s\right)}{\sum_{i \in \{1,\ldots,r(s)-1\}} \ D\left(\pi_i^s, \pi_{i+1}^s\right)} \tag{8}$$

To avoid ovefitting the dataset was split 100 times and a model was made each time. The average of these 100 values for the accuracy is reported as the accuracy of the algorithm.

5.2 Results

To examine the performance of the model, the accuracy measure was calculated for each iteration with a different training and test set. The method developed here resulted in an average cross-validated accuracy of 62% with a standard deviation of 0.6%. When the start and end point of trip-segments was given to the model as input data, the average accuracy was 78% with 0.8% standard deviation, which suggests that pre-processing the data with a segment detection algorithm might improve the results. The pre-processing steps in the transit itinerary detection algorithms were time consuming. Therefore, iteration of the algorithms took a significant amount of time. As a result, it was not possible to undertake a thorough sensitivity analysis. Conclusions about the

methods in the context of mode mentioned in the previous section are also relevant in the context of transit itinerary inference. However, since so little is known about transit itinerary inference, a Machine Learning method whose behavior is well-known and understood (Random Forest) rather than a semi-supervised algorithm was considered wisest in this context.

While there are many studies that have investigated segment detection, they have all focused on change of segment based on the change of characteristics of movement related to mode of transport. Therefore a new segment detection algorithm that is sensitive to the change of transit vehicle can be beneficial.

These trips were made continuously without common activities between trips. Participants started their next trip right after ending the previous trip and did not necessarily take a break between the two trips. This makes it harder to detect the beginning and end of a trip. Therefore on data with daily activities between trips that trips and segments are easier to detect, higher results might be achieved.

Chapter 6

Conclusion

Contributions

The first study in this thesis proposed the application of semi-supervised algorithms in mode detection when using smartphone travel survey data. This can help researchers and planners to use both validated and un-validated data allowing models estimated to be more accurate and realistic as since they would be able to use larger data-sets. These types of algorithms have not yet been tested for information inference in GPS or smartphone travel surveys in the transportation literature. The semi-supervised method used in this study was label propagation with K-nearest neighbors as its kernel. The results of this model were compared with those of three popular supervised learning methods, namely decision tree, random forest and logistic regression. All models were trained and tested using 100 different training and test sets. Several sensitivity analyses were done to choose the best values for the meta-parameters of each model.

The results show that when training data sets include higher proportions of validated data, the models estimated by the supervised algorithms perform better than the model estimated by the semi-supervised algorithm. However, their average accuracies decrease when the proportion of unlabeled data increase, while the label propagation model, as expected, performs almost the same under these circumstances. In general, the study shows that the proposed semi-supervised algorithm outperforms the three supervised algorithms when the proportion of unlabeled data was greater than 70%.

In the second study, an algorithm, based on 100% validated data was developed to detect the transit itinerary. In order to examine the performance of the model, the sum of the distance between consecutive correctly estimated points was divided be the sum of the whole length of all trips. This accuracy measure was calculated for each iteration with a different training and test set. The method developed here resulted in 62% on average cross-validated accuracy with a 0.6% standard deviation. It is worth mentioning that when the start and end point of trip-segments was given to the model as input data, the average accuracy was 78% with 0.8% standard deviation. This suggests that preprocessing the data with a good segment detection algorithm might improve the results.

Limitations

In this study, the dominant transportation mode was considered as the transportation mode for whole trip, which adds error to the estimations and might affect a particular model more than others.

While there are many studies that have investigated segment detection, they have all focused on change of segment based on the change of characteristics of movement related to mode of transport. Therefore a new segment detection algorithm that is sensitive to the change of transit vehicle can be beneficial.

Also this data was collected in continuously without common activities between trips. This makes it harder to detect the beginning and end of a trip. Therefore on a data with daily activities between trips that trips and segments are easier to detect, better results might be achieved.

Future work

This study is a beginning for the application of semi-supervised methods in transportation mode detection using GPS data. As a result, there are quite a few possibilities for improvement. For example, in this study, the dominant transportation mode was considered as the transportation mode for the entire trip. This adds error to the estimations and might affect a particular model more than others. Thus, a more detailed dataset containing mode information by segment, could help to have a better comparison. Also, features used in this study were very simple. Application of more advanced features such as closeness to specific locations of the transportation network might improve results overall.

Most importantly, the methods used in this study were mostly chosen due to their popularity, simple structure and availability in software packages. There remain many other interesting methods, especially in the case of semi-supervised algorithms, that could be considered and tested.

Appendix A

Comparison of Python Code for Supervised and Semi-supervised Algorithms

4.8, 4.9 and 4.10:

```
import csv, psycopg2, datetime, sklearn, random
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.semi_supervised import LabelPropagation
import pyRserve
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
```

```
conn = psycopg2.connect("dbname='DataMobile2014' user='postgres'
    host='localhost' password='0'")
cur = conn.cursor()
dataset=[]
cur.execute("""select cumulative_distance, avg_speed,
    distance_stop_start, distance_stop_end, trip_duration_seconds,
     mode_code
            from datasetfinal
            where direct_distance>250 and trip_duration>'2 min'
        """)
dataset.append([])
for i in cur:
    dataset.append(list(i))
del dataset[0]
conn.close()

u_used = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

conn = pyRserve.connect()

def comp(u_ratio, it):
    print 'start modeling', datetime.datetime.now()
    DTsenstrain = []
    DTsenstest = []
    DTsenstestst d = []
    RFsenstrain = []
    RFsenstest = []
    RFsenstestst d = []
```

```
MLPsenstrain = []
MLPsenstest = []
MLPsenstesteststd = []
LGTsenstrain = []
LGTsenstest = []
LGTsenstesteststd = []
GNBsenstrain = []
GNBsenstest = []
GNBsenstesteststd = []
MNBsenstrain = []
MNBsenstest = []
MNBsenstesteststd = []
LPsenstrain = []
LPsenstest = []
LPsenstesteststd = []
for j in u_used:
        DTtrainacclist=[]
        DTtestacclist=[]
        RFtrainacclist=[]
        RFtestacclist=[]
        MLPtrainacclist=[]
        MLPtestacclist=[]
        LGTtrainacclist=[]
        LGTtestacclist=[]
        GNBtrainacclist=[]
        GNBtestacclist=[]
        MNBtrainacclist=[]
        MNBtestacclist=[]
```

```
LPtrainacclist=[]
LPtestacclist=[]
for i in range(it):
        random.shuffle(dataset)
        train = dataset[0:int(0.8*len(dataset))]
        test = dataset[int(0.8*len(dataset)):len(
            dataset)]

        train_y = [a[-1] for a in train]
        train_x = []
        # number of points, avg distance and point
            ratio as input
        for row in train:
            train_x.append(row[0:5])

        test_y = [a[-1] for a in test]
        test_x = []
        for row in test:
            test_x.append(row[0:5])

        conn.r.x0=np.array([row[0] for row in train_x
            [int(u_ratio*len(train)):]])
        conn.r.x1=np.array([row[1] for row in train_x
            [int(u_ratio*len(train)):]])
        conn.r.x2=np.array([row[2] for row in train_x
            [int(u_ratio*len(train)):]])
        conn.r.x3=np.array([row[3] for row in train_x
            [int(u_ratio*len(train)):]])
```

```
conn.r.x4=np.array([row[4] for row in train_x
    [int(u_ratio*len(train)):]])
conn.r.y=np.array(train_y[int(u_ratio*len(
    train)):])
conn.voidEval('''library(rpart)
     d <- data.frame(y = y, x0 = x0, x1 = x1,
         x2 = x2, x3 = x3, x4 = x4)
     dt <- rpart(y~x0+x1+x2+x3+x4, data=d,
         method="class", control=rpart.control(
         minsplit=20, cp=0.01))
     train_prediction <- predict(dt, d, type="
         class")
     ''')
trainpred = conn.r.train_prediction
trainacc = sklearn.metrics.accuracy_score(
    train_y[int(u_ratio*len(train)):],
    trainpred, normalize=True)
DTtrainacclist.append(trainacc)
conn.r.x0=np.array([row[0] for row in test_x
    ])
conn.r.x1=np.array([row[1] for row in test_x
    ])
conn.r.x2=np.array([row[2] for row in test_x
    ])
conn.r.x3=np.array([row[3] for row in test_x
    ])
conn.r.x4=np.array([row[4] for row in test_x
    ])
```

```
conn.eval('''
    d <- data.frame(x0 = x0, x1 = x1, x2 = x2
        , x3 = x3, x4 = x4)
    test_prediction <- predict(dt, d, type="
        class")
    ''')
testpred = conn.r.test_prediction
testacc = sklearn.metrics.accuracy_score(
    test_y, testpred, normalize=True)
DTtestacclist.append(testacc)

rf = RandomForestClassifier(n_estimators=10,
    n_jobs=-1).fit(train_x[int(u_ratio*len(
    train)):], train_y[int(u_ratio*len(train))
    :])
trainpred = [i for i in rf.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(
    train_y[int(u_ratio*len(train)):],
    trainpred[int(u_ratio*len(train)):],
    normalize=True)
RFtrainacclist.append(trainacc)
testpred = [i for i in rf.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(
    test_y, testpred, normalize=True)
RFtestacclist.append(testacc)
```

```
mlp = MLPClassifier(hidden_layer_sizes=15,
    activation='tanh', solver='sgd', alpha
    =0.0001, batch_size='auto', learning_rate=
    'constant', learning_rate_init=0.001,
    max_iter=50000).fit(train_x[int(u_ratio*
    len(train)):], train_y[int(u_ratio*len(
    train)):])
trainpred = [i for i in mlp.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(
    train_y[int(u_ratio*len(train)):],
    trainpred[int(u_ratio*len(train)):],
    normalize=True)
MLPtrainacclist.append(trainacc)
testpred = [i for i in mlp.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(
    test_y, testpred, normalize=True)
MLPtestacclist.append(testacc)

lgt = LogisticRegression(n_jobs=-1).fit(
    train_x[int(u_ratio*len(train)):], train_y
    [int(u_ratio*len(train)):])
trainpred = [i for i in lgt.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(
    train_y[int(u_ratio*len(train)):],
    trainpred[int(u_ratio*len(train)):],
    normalize=True)
LGTtrainacclist.append(trainacc)
testpred = [i for i in lgt.predict(test_x)]
```

```
testacc = sklearn.metrics.accuracy_score(
    test_y, testpred, normalize=True)
LGTtestacclist.append(testacc)

mnb = MultinomialNB().fit(train_x[int(u_ratio
    *len(train)):], train_y[int(u_ratio*len(
    train)):])
trainpred = [i for i in mnb.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(
    train_y[int(u_ratio*len(train)):],
    trainpred[int(u_ratio*len(train)):],
    normalize=True)
MNBtrainacclist.append(trainacc)
testpred = [i for i in mnb.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(
    test_y, testpred, normalize=True)
MNBtestacclist.append(testacc)

c1, c2, c3, c4, c5 = [1, 3.6, 1, 1, 1]
train_y = [int(a[-1]) for a in train]
# unlabaling some observations
for i in range(int(u_ratio*len(train))):
        train_y[i] = -1
train_x = []
for row in train:
    train_x.append([c1*float(row[0]), c2*
        float(row[1]), c3*float(row[2]), c4*
        float(row[3]), c5*float(row[4])])
```

```
            test_y = [int(a[-1]) for a in test]
            test_x = []
            for row in test:
                test_x.append([c1*float(row[0]), c2*float
                    (row[1]), c3*float(row[2]), c4*float(
                    row[3]), c5*float(row[4])])

            lp = LabelPropagation(kernel='knn',
                n_neighbors=15, alpha=1).fit(train_x[int
                ((1-j)*u_ratio*len(train)):], train_y[int
                ((1-j)*u_ratio*len(train)):])
            trainpred = [i for i in lp.predict(train_x)]
            trainacc = sklearn.metrics.accuracy_score(
                train_y[int(u_ratio*len(train)):],
                trainpred[int(u_ratio*len(train)):],
                normalize=True)
            LPtrainacclist.append(trainacc)
            testpred = [i for i in lp.predict(test_x)]
            testacc = sklearn.metrics.accuracy_score(
                test_y, testpred, normalize=True)
            LPtestacclist.append(testacc)

    DTsenstrain.append(np.mean(DTtrainacclist))
    DTsenstest.append(np.mean(DTtestacclist))
    DTsensteststd.append(np.std(DTtestacclist)*2/10)
    RFsenstrain.append(np.mean(RFtrainacclist))
    RFsenstest.append(np.mean(RFtestacclist))
    RFsensteststd.append(np.std(RFtestacclist)*2/10)
```

```
            MLPsenstrain.append(np.mean(MLPtrainacclist))
            MLPsenstest.append(np.mean(MLPtestacclist))
            MLPsensteststd.append(np.std(MLPtestacclist)*2/10)
            LGTsenstrain.append(np.mean(LGTtrainacclist))
            LGTsenstest.append(np.mean(LGTtestacclist))
            LGTsensteststd.append(np.std(LGTtestacclist)*2/10)
            MNBsenstrain.append(np.mean(GNBtrainacclist))
            MNBsenstest.append(np.mean(GNBtestacclist))
            MNBsensteststd.append(np.std(GNBtestacclist)*2/10)
            LPsenstrain.append(np.mean(LPtrainacclist))
            LPsenstest.append(np.mean(LPtestacclist))
            LPsensteststd.append(np.std(LPtestacclist)*2/10)
    print 'end RF', datetime.datetime.now()

    fig = plt.figure()
    plt.ylim((0.5, 0.9))
    plt.xlabel(r'$n$', fontsize=15)
    plt.ylabel('accuracy', fontsize=15)
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
       DTsenstest, DTsensteststd, label='Decision Tree',
       elinewidth=0.5, capsize=2)
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
       RFsenstest, RFsensteststd, label='Random Forest',
       elinewidth=0.5, capsize=2)
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
       MLPsenstest, MLPsensteststd, label='Multi-Layer Perceptron
       ', elinewidth=0.5, capsize=2)
```

```
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
        LGTsenstest, LGTsensteststd, label='Logistic Regression',
        elinewidth=0.5, capsize=2)
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
        MNBsenstest, MNBsensteststd, label='Multinomial Naive
        Bayes', elinewidth=0.5, capsize=2)
    plt.errorbar(np.multiply(u_used,u_ratio*len(train)),
        LPsenstest, LPsensteststd, label='Label Propagation',
        elinewidth=0.5, capsize=2)
    plt.legend()
    plt.savefig("C:/Users/Mohsen/Dropbox/Mohsen's thesis (tii)/ML
        Project (mode, semi-sup. vs sup.)/output/comparison {}%
        unlabeled ieee.pdf".format(100*u_ratio))

for i in [0.01, 0.5, 0.7, 0.75, 0.8, 0.85]:
    comp(i, 100)
```

4.7

```
import csv, psycopg2, datetime, sklearn, random
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.semi_supervised import LabelPropagation
import pyRserve
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
```

```
conn = psycopg2.connect("dbname='DataMobile2014' user='postgres'
    host='localhost' password='0'")
cur = conn.cursor()
dataset=[]
cur.execute("""select cumulative_distance, avg_speed,
    distance_stop_start, distance_stop_end, trip_duration_seconds,
     mode_code
            from datasetfinal
            where direct_distance>250 and trip_duration>'2 min'
        """)
dataset.append([])
for i in cur:
    dataset.append(list(i))
del dataset[0]
conn.close()

conn = pyRserve.connect()

val = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

print 'start modeling', datetime.datetime.now()
DTsenstrain = []
DTsenstest = []
DTsenststeststd = []
RFsenstrain = []
RFsenstest = []
RFsenststeststd = []
```

```
MLPsenstrain = []
MLPsenstest = []
MLPsensteststd = []
LGTsenstrain = []
LGTsenstest = []
LGTsensteststd = []
GNBsenstrain = []
GNBsenstest = []
GNBsensteststd = []
MNBsenstrain = []
MNBsenstest = []
MNBsensteststd = []
LPsenstrain = []
LPsenstest = []
LPsensteststd = []
for j in val:
        DTtrainacclist=[]
        DTtestacclist=[]
        RFtrainacclist=[]
        RFtestacclist=[]
        MLPtrainacclist=[]
        MLPtestacclist=[]
        LGTtrainacclist=[]
        LGTtestacclist=[]
        GNBtrainacclist=[]
        GNBtestacclist=[]
        MNBtrainacclist=[]
        MNBtestacclist=[]
```

```
LPtrainacclist=[]
LPtestacclist=[]
for i in range(100):
        random.shuffle(dataset)
        train = dataset[0:int(0.8*len(dataset))]
        test = dataset[int(0.8*len(dataset)):len(dataset)
           ]

        train_y = [a[-1] for a in train]
        train_x = []
        for row in train:
            train_x.append(row[0:5])

        test_y = [a[-1] for a in test]
        test_x = []
        for row in test:
            test_x.append(row[0:5])

        conn.r.x0=np.array([row[0] for row in train_x[int
           (j*len(train)):]])
        conn.r.x1=np.array([row[1] for row in train_x[int
           (j*len(train)):]])
        conn.r.x2=np.array([row[2] for row in train_x[int
           (j*len(train)):]])
        conn.r.x3=np.array([row[3] for row in train_x[int
           (j*len(train)):]])
        conn.r.x4=np.array([row[4] for row in train_x[int
           (j*len(train)):]])
```

```
conn.r.y=np.array(train_y[int(j*len(train)):])
conn.voidEval('''library(rpart)
    d <- data.frame(y = y, x0 = x0, x1 = x1, x2 =
        x2, x3 = x3, x4 = x4)
    dt <- rpart(y~x0+x1+x2+x3+x4, data=d, method
       ="class", control=rpart.control(minsplit
       =20, cp=0.01))
##      dt <- prune(dt, cp = 0.01)
    train_prediction <- predict(dt, d, type="
       class")
    ''')
trainpred = conn.r.train_prediction
trainacc = sklearn.metrics.accuracy_score(train_y
   [int(j*len(train)):], trainpred, normalize=
   True)
DTtrainacclist.append(trainacc)
conn.r.x0=np.array([row[0] for row in test_x])
conn.r.x1=np.array([row[1] for row in test_x])
conn.r.x2=np.array([row[2] for row in test_x])
conn.r.x3=np.array([row[3] for row in test_x])
conn.r.x4=np.array([row[4] for row in test_x])
conn.eval('''
    d <- data.frame(x0 = x0, x1 = x1, x2 = x2, x3
        = x3, x4 = x4)
    test_prediction <- predict(dt, d, type="class
       ")
    ''')
testpred = conn.r.test_prediction
```

```
testacc = sklearn.metrics.accuracy_score(test_y,
    testpred, normalize=True)
DTtestacclist.append(testacc)

rf = RandomForestClassifier(n_estimators=10,
    n_jobs=-1).fit(train_x[int(j*len(train)):],
    train_y[int(j*len(train)):])
trainpred = [i for i in rf.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(train_y
    [int(j*len(train)):], trainpred[int(j*len(
    train)):], normalize=True)
RFtrainacclist.append(trainacc)
testpred = [i for i in rf.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(test_y,
    testpred, normalize=True)
RFtestacclist.append(testacc)

mlp = MLPClassifier(hidden_layer_sizes=15,
    activation='tanh', solver='sgd', alpha=0.0001,
     batch_size='auto', learning_rate='constant',
    learning_rate_init=0.001, max_iter=50000).fit(
    train_x[int(j*len(train)):], train_y[int(j*len
    (train)):])
trainpred = [i for i in mlp.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(train_y
    [int(j*len(train)):], trainpred[int(j*len(
    train)):], normalize=True)
MLPtrainacclist.append(trainacc)
```

```
testpred = [i for i in mlp.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(test_y,
    testpred, normalize=True)
MLPtestacclist.append(testacc)

lgt = LogisticRegression(n_jobs=-1).fit(train_x[
    int(j*len(train)):], train_y[int(j*len(train))
    :])
trainpred = [i for i in lgt.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(train_y
    [int(j*len(train)):], trainpred[int(j*len(
    train)):], normalize=True)
LGTtrainacclist.append(trainacc)
testpred = [i for i in lgt.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(test_y,
    testpred, normalize=True)
LGTtestacclist.append(testacc)

mnb = MultinomialNB().fit(train_x[int(j*len(train
    )):], train_y[int(j*len(train)):])
trainpred = [i for i in mnb.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(train_y
    [int(j*len(train)):], trainpred[int(j*len(
    train)):], normalize=True)
MNBtrainacclist.append(trainacc)
testpred = [i for i in mnb.predict(test_x)]
testacc = sklearn.metrics.accuracy_score(test_y,
    testpred, normalize=True)
```

```
MNBtestacclist.append(testacc)


c1, c2, c3, c4, c5 = [1, 3.6, 1, 1, 1]
train_y = [int(a[-1]) for a in train]
# unlabaling some observations
for i in range(int(j*len(train))):
        train_y[i] = -1
train_x = []
for row in train:
    train_x.append([c1*float(row[0]), c2*float(
        row[1]), c3*float(row[2]), c4*float(row
        [3]), c5*float(row[4])])
test_y = [int(a[-1]) for a in test]
test_x = []
for row in test:
    test_x.append([c1*float(row[0]), c2*float(row
        [1]), c3*float(row[2]), c4*float(row[3]),
        c5*float(row[4])])


lp = LabelPropagation(kernel='knn', n_neighbors
   =15, alpha=1).fit(train_x, train_y)
trainpred = [i for i in lp.predict(train_x)]
trainacc = sklearn.metrics.accuracy_score(train_y
   [int(j*len(train)):], trainpred[int(j*len(
   train)):], normalize=True)
LPtrainacclist.append(trainacc)
testpred = [i for i in lp.predict(test_x)]
```

```
                testacc = sklearn.metrics.accuracy_score(test_y,
                    testpred, normalize=True)
                LPtestacclist.append(testacc)


        DTsenstrain.append(np.mean(DTtrainacclist))
        DTsenstest.append(np.mean(DTtestacclist))
        DTsensteststd.append(np.std(DTtestacclist)*2/10)
        RFsenstrain.append(np.mean(RFtrainacclist))
        RFsenstest.append(np.mean(RFtestacclist))
        RFsensteststd.append(np.std(RFtestacclist)*2/10)
        MLPsenstrain.append(np.mean(MLPtrainacclist))
        MLPsenstest.append(np.mean(MLPtestacclist))
        MLPsensteststd.append(np.std(MLPtestacclist)*2/10)
        LGTsenstrain.append(np.mean(LGTtrainacclist))
        LGTsenstest.append(np.mean(LGTtestacclist))
        LGTsensteststd.append(np.std(LGTtestacclist)*2/10)
        MNBsenstrain.append(np.mean(GNBtrainacclist))
        MNBsenstest.append(np.mean(GNBtestacclist))
        MNBsensteststd.append(np.std(GNBtestacclist)*2/10)
        LPsenstrain.append(np.mean(LPtrainacclist))
        LPsenstest.append(np.mean(LPtestacclist))
        LPsensteststd.append(np.std(LPtestacclist)*2/10)
print 'end RF', datetime.datetime.now()


fig = plt.figure()
plt.xlim((0, 1))
plt.ylim((0.5, 0.9))
```

```
plt.xlabel(r'$\frac{n}{m+n}$', fontsize=15)
plt.ylabel('accuracy', fontsize=15)
plt.errorbar(val, DTsenstest, DTsensteststd, label='Decision␣Tree
    ', elinewidth=0.5, capsize=2)
plt.errorbar(val, RFsenstest, RFsensteststd, label='Random␣Forest
    ', elinewidth=0.5, capsize=2)
plt.errorbar(val, MLPsenstest, MLPsensteststd, label='Multi-Layer
    ␣Perceptron', elinewidth=0.5, capsize=2)
plt.errorbar(val, LGTsenstest, LGTsensteststd, label='Logistic␣
    Regression', elinewidth=0.5, capsize=2)
plt.errorbar(val, MNBsenstest, MNBsensteststd, label='Multinomial
    ␣Naive␣Bayes', elinewidth=0.5, capsize=2)
plt.errorbar(val, LPsenstest, LPsensteststd, label='Label␣
    Propagation', elinewidth=0.5, capsize=2)
plt.legend()
plt.savefig("C:/Users/Mohsen/Dropbox/Mohsen's␣thesis␣(tii)/ML␣
    Project␣(mode,␣semi-sup.␣vs␣sup.)/output/comparison.pdf")
```

Appendix B

Python Code for Transit Itinerary Detection

```
import os, csv, psycopg2, datetime, sklearn, random, numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

conn = psycopg2.connect("dbname='
    DataMobileMay2016_plus_segment_detection' user='postgres' host
    ='localhost' password='0'")
cur=conn.cursor()

def f_initialize():
    print 'start f_initialize', datetime.datetime.now()
    cur.execute("""
        create extension if not exists postgis;
        drop table if exists stop_times;
```

```
create table stop_times (trip_id character varying,
    arrival_time interval, departure_time interval ,
    stop_id integer, stop_sequence smallint);
copy stop_times (trip_id, arrival_time, departure_time,
    stop_id, stop_sequence)
        from 'C:\PostgreSQL\9.5\data\DataMobile_2016-05\
            gtfs\stop_times.csv' delimiter ',' csv header;
drop table if exists stops;
create table stops (stop_id integer, stop_code integer,
    stop_name character varying, stop_lat real, stop_lon
    real, stop_url character varying, wheelchair_boarding
    smallint);
copy stops (stop_id, stop_code, stop_name, stop_lat,
    stop_lon, stop_url, wheelchair_boarding)
        from 'C:\PostgreSQL\9.5\data\DataMobile_2016-05\
            gtfs\stops.csv' delimiter ',' csv header;
drop table if exists trips;
create table trips (route_id smallint, service_id varchar
    , trip_id varchar, trip_headsign varchar, direction_id
     boolean, shape_id int, wheelchair_accessible smallint
    , note_fr varchar, note_en varchar);
copy trips (route_id, service_id, trip_id, trip_headsign,
     direction_id, shape_id, wheelchair_accessible,
    note_fr, note_en)
        from 'C:\PostgreSQL\9.5\data\DataMobile_2016-05\
            gtfs\\trips.csv' delimiter ',' csv header;
drop table if exists coordinates;
CREATE TABLE coordinates
```

```
        (id integer, user_id character varying, latitude
            real, longitude real, speed real,
        h_accuracy real, v_accuracy real, point_type
            integer, t varchar,
        inst_acc character varying);
set datestyle to 'YMD';
COPY coordinates (id, user_id, latitude, longitude, speed
    , h_accuracy, v_accuracy, point_type, t,
        inst_acc)
        FROM 'C:\PostgreSQL\9.5\data\DataMobile_2016-05\
            coordinates.csv' DELIMITER ',' CSV HEADER;
CREATE INDEX if not exists i_stops
        ON stops (stop_id, stop_lat, stop_lon);
CREATE INDEX if not exists i_stop_times
        ON stop_times (trip_id, stop_id, arrival_time);
CREATE INDEX if not exists i_trips
        ON trips (route_id, trip_id);
delete from coordinates where (substring(t FROM '[0-9]+')
    ::smallint)<1990;
alter table coordinates alter column t type timestamp
    USING t::timestamp without time zone;
delete from coordinates where latitude<40 or latitude>50
    or longitude<-80 or longitude>-65;
alter table coordinates add column p geometry;
update coordinates set p=st_transform(st_setsrid(st_point
    (longitude, latitude), 4326), 32618)
--      drop table if exists metro_lines;
```

```
--          select route_id, trip_id, st_makeline(
    transit_route_points) m_lines into metro_lines from
    transit_routes
--              where route_id<10 group by route_id, trip_id
        """)
    conn.commit()
    print 'end_f_initialize', datetime.datetime.now()

def f_transit_trip():
    print 'start_f_timestamp_correction', datetime.datetime.now()
    cur.execute("""
        drop table if exists transit_trip;
        select route_id, trip_id, min(t) mint, max(t) maxt,
            st_makeline(p) trip_line into transit_trip from
             (select route_id, a.trip_id, t, st_makepoint(st_x(gps
                ), st_y(gps), t) p from
                 (select route_id, trip_id from trips) a,
                 (select trip_id, extract(epoch from(arrival_time)
                     ) t, stop_id from stop_times) b,
                 (select stop_id, st_transform(st_setsrid(
                     st_makepoint(stop_lon, stop_lat), 4326),
                     32618) gps from stops) c
                 where a.trip_id=b.trip_id and b.stop_id=c.stop_id
                      order by t) aa
            group by route_id, trip_id;
        CREATE INDEX if not exists i_transit_trip
                ON transit_trip (route_id, trip_id, (md5(
                    trip_line::text)));
```

```
        """)
    conn.commit()
    print 'end_f_transit_trip', datetime.datetime.now()

def f_transit_route():
    print 'start_f_transit_route', datetime.datetime.now()
    cur.execute("""select route_id, count(0) freq from trips
       group by route_id
        """)
    route_list=[]
    for row in cur:
        route_list.append(list(row))
    cur.execute("""
        drop table if exists transit_route;
        create table transit_route (route_id integer, trip_id
           varchar, route_freq integer, dur integer, route_avg_t
           real, stop_freq real, route_speed real, trans_line
           geometry)
        """)
    conn.commit()
    for i in range(len(route_list)):
        cur.execute("""
            insert into transit_route
                select route_id, trip_id, route_freq, dur,
                   route_avg_t, st_length(trans_line)/n_points
                   stop_ferq, st_length(trans_line)/dur
                   route_speed, trans_line from
```

```
                (select route_id, trip_id, {} route_freq,
                    count(0) n_points, max(tt)-min(tt) dur,
                    avg(tt) route_avg_t,
                     st_makeline(st_transform(st_setsrid(
                        st_makepoint(stop_lon, stop_lat),
                        4326), 32618)) trans_line from
                     (select route_id, f.trip_id, stop_lon,
                        stop_lat, tt from
                            (select *, extract(epoch from
                                arrival_time) tt from
                                stop_times order by tt) f,
                            (select * from stops) g,
                            (select route_id, trip_id from
                                trips where route_id={} limit
                                1) h
                            where f.stop_id=g.stop_id and f.
                                trip_id=h.trip_id
                            order by arrival_time asc) a
                            group by route_id, trip_id) aa
            """.format(route_list[i][1], route_list[i][0]))
        conn.commit()
        print i, route_list[i], datetime.datetime.now()
cur.execute("""
    CREATE INDEX if not exists i_transit_route
            ON transit_route (route_id, trans_line)
    """)
conn.commit()
del route_list
```

```
    print 'end_f_transit_route', datetime.datetime.now()

def f_zone_route():
    print 'start_f_zone_route', datetime.datetime.now()
    cur.execute("""
        drop table if exists zone_route;
        create table zone_route (sm13 integer, route_id integer);
        insert into zone_route
            select sm13, route_id from
                transit_route a,
                sm_eod13 b
                where st_intersects(a.trans_line, st_transform(b.
                    geom, 32618))
        """)
    conn.commit()
    print 'end_f_zone_route', datetime.datetime.now()

def f_datamobile_data_fetch():
    print 'start_f_data_fetch', datetime.datetime.now()
    cur.execute("""
        drop table if exists tii;
        CREATE TABLE tii
            (id integer, user_id varchar, latitude real,
               longitude real, speed real,
            h_accuracy real, v_accuracy real, point_type varchar,
                t timestamp,
            inst_acc varchar, ipere integer, modes varchar, route
                varchar,
```

```
        zone smallint, ord integer)
    """)
conn.commit()
for file in os.listdir("C:\PostgreSQL\9.5\data\
    DataMobile_2016-05\Data"):
    if not file.endswith(".csv"):
        continue
    with open("C:\PostgreSQL\9.5\data\DataMobile_2016-05\Data
        \{}".format(file), 'rU') as f:
        reader = csv.DictReader(f)
        dataset=[]
        for row in reader:
            if row['id']!='':
                dataset.append([row['id'], row['user_id'],
                    row['latitude'], row['longitude'], row['
                    speed'],
                                row['h_accuracy'], row['
                                    v_accuracy'], row['
                                    point_type'], row['
                                    timestamp'],
                                row['inst_acc'], row['ipere'
                                    ], row['mode'], row['route
                                    ']])
    if '/' in dataset[0][8] and [int(s) for s in dataset
        [0][8].split('/') if s.isdigit()][1] in [5,6,7]:
        cur.execute("""set datestyle to 'DMY'""")
    elif '-' in dataset[0][8] and [int(s) for s in dataset
        [0][8].split('-') if s.isdigit()][1] in [5,6,7]:
```

```
            cur.execute("""set datestyle to 'DMY'""")
        else:
            cur.execute("""set datestyle to 'MDY'""")
        del dataset[0]
        cur.executemany("""INSERT INTO tii (id, user_id, latitude
            , longitude, speed,
                        h_accuracy, v_accuracy, point_type, t,
                            inst_acc, ipere, modes, route, zone,
                            ord)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %
                            s, %s, %s, %s, %s, 0, 0)""", dataset)
    conn.commit()
    print 'end_f_datamobile_data_fetch', datetime.datetime.now()

def f_tii_correction():
    print 'start_f_tii_correction', datetime.datetime.now()
    cur.execute("""
        update tii a set t=b.t from coordinates b where
            a.user_id=b.user_id and a.latitude=b.latitude and
            a.longitude=b.longitude and a.t<b.t+interval'2 min'
                and a.t>b.t-interval'2 min';
        delete from tii where id IN
            (select id from
                (select id, row_number() over (partition by
                    user_id, t order by id) as rnum
                    FROM tii) a
                where a.rnum > 1);
```

```
        delete from tii where t<timestamp '2016-05-01 00:00:01'
            or t>timestamp '2016-07-01 00:00:01';
        update tii set modes=lower(regexp_replace(modes, '\s+',
            ''));
        update tii set modes='unknown' where modes='';
        update tii set modes='unknown' where modes='unkown';
        update tii set modes='walk' where modes='wait';
        update tii set route=lower(regexp_replace(route, '\s+',
            ''));
        update tii set route='unknown' where route='';
        update tii set modes='car' where
                modes='passengercar' or modes='carpassenger' or
                    modes='passenger';
        update tii set route=regexp_replace(route, '[^a-zA-Z0
            -9]+', '');
        update tii set route=modes where modes!='bus';
        update tii a set zone=sm13 from sm_eod13 b
             where st_intersects(st_transform(st_setsrid(
                st_makepoint(longitude, latitude), 4326), 32618),
                st_transform(b.geom, 32618))
        """)
    conn.commit()
    cur.execute("""
        select user_id, t from tii order by user_id, t
        """)
    tii_list=[]
    for row in cur:
        tii_list.append(row)
```

```
    i=0
    for row in tii_list:
        cur.execute("""
            update tii set ord={} where user_id='{}' and t=
                timestamp '{}'
            """.format(i, row[0], row[1]))
        conn.commit()
        i+=1
    cur.execute("""
        CREATE INDEX if not exists i_tii
                ON tii (ord, user_id, latitude, longitude, t)
        """)
    conn.commit()
    print 'end_f_tii_correction', datetime.datetime.now()

def f_seg_route_pairs():
    print 'start_f_seg_route_pairs', datetime.datetime.now()
    cur.execute("""
        select route_id, trans_line from transit_route
        """)
    route_list=[]
    for row in cur:
        route_list.append(row)
    cur.execute("""
        drop table if exists seg_route_pairs;
        create table seg_route_pairs (id integer, user_id varchar
            , start_t timestamp, end_t timestamp, start_ord
            integer, end_ord integer, route_id smallint)
```

```
        """)
    conn.commit()
    j=0
    for route in route_list:
        temp_list=[]
        cur.execute("""
            select aaa.user_id, aaa.t, aaa.ord from
                (select aa.* from
                        tii aa,
                        (select sm13 from
                                (select * from transit_route
                                   where route_id={}) a,
                                sm_eod13 b
                                where st_intersects(a.trans_line,
                                     st_transform(b.geom, 32618)))
                                     bb
                        where aa.zone=bb.sm13
--                                  and aa.ipere!=99999
                        ) aaa,
                (select * from transit_route where route_id={})
                   bbb
                where st_distance(trans_line, st_transform(
                   st_setsrid(st_makepoint(longitude, latitude),
                   4326), 32618))<100
                order by user_id, t
            """.format(route[0], route[0]))
        for row in cur:
            temp_list.append(row)
```

```
n=len(temp_list)
if n>1:
    start_t=temp_list[0][1]
    start_ord=temp_list[0][2]
    for i in range(n):
        if i!=n-1:
            if temp_list[i][0]!=temp_list[i-1][0] or
               temp_list[i][1]-temp_list[i-1][1]>datetime
               .timedelta(hours=1) or temp_list[i][2]-
               temp_list[i-1][2]>3:
                end_t=temp_list[i-1][1]
                end_ord=temp_list[i-1][2]
                cur.execute("""
                    insert into seg_route_pairs (id,
                       user_id, start_t, end_t, start_ord
                       , end_ord, route_id)
                        values (%s, %s, %s, %s, %s, %s, %
                           s)
                    """, [j, temp_list[i][0], start_t,
                       end_t, start_ord, end_ord, route
                       [0]])
                conn.commit()
                start_t=temp_list[i][1]
                start_ord=temp_list[i][2]
                j+=1
        else:
            end_t=temp_list[i][1]
            end_ord=temp_list[i][2]
```

```
                    cur.execute("""
                        insert into seg_route_pairs (id, user_id,
                            start_t, end_t, start_ord, end_ord,
                           route_id)
                            values (%s, %s, %s, %s, %s, %s, %s)
                        """, [j, temp_list[i][0], start_t, end_t,
                            start_ord, end_ord, route[0]])
                    conn.commit()
                    j+=1
    cur.execute("""
        drop table if exists seg_route_list;
        select id, user_id, start_t, end_t, start_ord, end_ord,
           route_id into seg_route_list from seg_route_pairs
        """)
    conn.commit()
    print 'end_f_seg_route_pairs', datetime.datetime.now()

def f_update_output():
    print 'start_f_update_output', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column modes varchar,
            add column route smallint,
            add column ipere smallint,
            add column doy smallint,
            add column output smallint,
            add column route_type smallint;
        update seg_route_pairs set
```

```
    modes='unknown',
    route=0,
    ipere=0,
    doy=0,
    output=0;
update seg_route_pairs a set
    modes=g.m
    from (select f.id, f.modes m from
        (select id, max(n) max from
            (select c.id, b.modes, count(1) n
                from (select user_id, t, ord, modes from
                    tii) b,
                     (select * from seg_route_list) c
                where b.ord>c.start_ord and b.ord<c.
                    end_ord
                group by id, modes) d
            group by id) e,
        (select c.id, b.modes, count(1) n
            from (select user_id, t, ord, modes from tii)
                b,
                (select * from seg_route_list) c
            where b.ord>c.start_ord and b.ord<c.end_ord
            group by id, modes
            ) f
        where e.id=f.id and e.max=f.n
        ) g
    where a.id=g.id;
update seg_route_pairs a set
```

```
    route=g.r
    from (select f.id, f.route r from
        (select id, max(n) max from
            (select c.id, b.route, count(1) n
                from (select user_id, t, ord, cast(
                    substring(route,'[0-9]+') as int)
                    route from tii) b,
                     (select * from seg_route_list) c
                where b.ord>c.start_ord and b.ord<c.
                    end_ord
                group by id, route) d
            group by id) e,
        (select c.id, b.route, count(1) n
            from (select user_id, t, ord, cast(substring(
                route,'[0-9]+') as int) route from tii) b,
                 (select * from seg_route_list) c
            where b.ord>c.start_ord and b.ord<c.end_ord
            group by id, route
            ) f
        where e.id=f.id and e.max=f.n
        ) g
    where a.id=g.id;
update seg_route_pairs a set
    output=d.o
    from (select c.id, round((1.0*count(1)/(end_ord-
        start_ord))+0.35) o
```

```
        from (select user_id, t, ord, modes, cast(
            substring(route,'[0-9]+') as int) r from tii)
            b,
             (select * from seg_route_list) c
        where b.ord>c.start_ord and b.ord<c.end_ord and b
            .modes='bus' and c.route_id=b.r
        group by id, end_ord, start_ord) d
    where a.id=d.id;
update seg_route_pairs a set
    output=d.o
    from (select c.id, round((1.0*count(1)/(end_ord-
        start_ord))+0.35) o
        from (select user_id, t, ord, modes, cast(
            substring(route,'[0-9]+') as int) r from tii)
            b,
             (select * from seg_route_list) c
        where b.ord>c.start_ord and b.ord<c.end_ord and b
            .modes='metro' and c.route_id<10 and c.
            route_id>0
        group by id, end_ord, start_ord) d
    where a.id=d.id;
update seg_route_pairs a set
    ipere=d.o
    from (select c.id, round(1-(1.0*count(1)/(end_ord-
        start_ord))) o
        from (select user_id, t, ord, ipere from tii) b,
             (select * from seg_route_list) c
```

```
                where b.ord>c.start_ord and b.ord<c.end_ord and b
                    .ipere=99999
                group by id, end_ord, start_ord) d
            where a.id=d.id;
        update seg_route_pairs a
            set doy= date_part('doy', b.t)
            from (select user_id, t, ord, date_part('doy', t) d
                from tii) b
            where b.ord=round((start_ord+end_ord)/2);
        update seg_route_pairs set route_type=0 where route_id>0
            and route_id<10;
        update seg_route_pairs set route_type=1 where route_id
            >=10 and route_id<250;
        update seg_route_pairs set route_type=2 where route_id
            >=250 and route_id<300;
        update seg_route_pairs set route_type=3 where route_id
            >=300 and route_id<400;
        update seg_route_pairs set route_type=4 where route_id
            >=400
        """)
    conn.commit()
    print 'end_f_update_output', datetime.datetime.now()

def f_simple_feature():
    print 'start_f_simple_feature', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column length real,
```

```
            add column duration real,
            add column points integer;
        update seg_route_pairs a set
            length=l/1000.0,
            duration=dur/1000.0,
            points=c
            from (select c.id,
                    st_length(st_makeline(st_transform(st_setsrid
                        (st_makepoint(longitude, latitude), 4326),
                         32618))) l,
                    extract(epoch from(max(t)-min(t))) dur,
                    count(0) c
                from (select user_id, t, ord, longitude, latitude
                    from tii order by user_id, t) b,
                    (select * from seg_route_list) c
                where b.ord>c.start_ord and b.ord<c.end_ord
                group by id, end_ord, start_ord) d
            where a.id=d.id;
        """)
    conn.commit()
    print 'end_f_simple_feature', datetime.datetime.now()

def f_delete_record():
    print 'start_f_delete_record', datetime.datetime.now()
    cur.execute("""
        delete from seg_route_pairs where ipere!=1;
        delete from seg_route_pairs where length<0.2 or points<2;
```

```
        delete from seg_route_pairs where modes='metro' and
            length<0.3;
        delete from seg_route_pairs where modes!='metro' and
            points<5;
        -- segment duration shouldn't exceed 2 hours (7200
            seconds)
        delete from seg_route_pairs where duration>7.2
        """)
    conn.commit()
    print 'end_f_delete_record', datetime.datetime.now()

def f_feature_in_buffer():
    print 'start_f_feature_in_buffer', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column points_freq real,
            add column seg_speed real;
        update seg_route_pairs set
            points_freq=length/points,
            seg_speed=length/duration;
        alter table seg_route_pairs
--              add column b_length real,
--              add column b_duration real,
--              add column b_points integer,
            add column dist real,
            add column dist_3d real,
            add column route_freq integer,
            add column route_dur integer,
```

```
        add column route_speed real,
        add column stop_freq real
    """)
conn.commit()
cur.execute("""select id, route_id, start_ord, end_ord from
   seg_route_pairs
    """)
seg_route_list=[]
for row in cur:
    seg_route_list.append(list(row))
for row in seg_route_list:
    cur.execute("""
        update seg_route_pairs aaa set
        --                  b_length=l/1000,
        --                  b_duration=dur/1000.0,
        --                  b_points=c,
                dist=bbb.avg_dist,
                route_freq=bbb.route_freq,
                route_dur=bbb.route_dur,
                route_speed=bbb.route_speed,
                stop_freq=bbb.stop_freq,
                dist_3d=bbb.min_avg_dist3d
                from (select bb.id, bb.route_id, bb.
                    route_freq, bb.route_dur, bb.route_speed,
                    bb.stop_freq, bb.l, bb.dur, bb.c, bb.
                    avg_dist, min(bb.avg_dist3d)
                    min_avg_dist3d
```

```
from (select c.id, d.route_id, e.trip_id,
    d.route_freq, d.dur route_dur, d.
    route_speed, d.stop_freq,
        st_length(st_makeline(
            st_transform(st_setsrid(
            st_makepoint(longitude,
            latitude), 4326), 32618))) l,
        extract(epoch from(max(t)-min(t))
            ) dur,
        count(0) c,
        avg(st_distance(d.trans_line,
            st_transform(st_setsrid(
            st_makepoint(longitude,
            latitude), 4326), 32618)))
            avg_dist,
        avg(st_3ddistance(e.trip_line,
            st_makepoint(st_x(st_transform
            (st_setsrid(st_makepoint(
            longitude, latitude), 4326),
            32618)), st_y(st_transform(
            st_setsrid(st_makepoint(
            longitude, latitude), 4326),
            32618)), date_part('hour', t)
            *3600+date_part('minute', t)
            *60+date_part('second', t))))
            avg_dist3d
```

```
                    from (select user_id, t, ord,
                        longitude, latitude from tii where
                         ord>{} and ord<{} order by ord) b
                        ,
                         (select * from seg_route_list
                            where route_id={}) c,
                         (select * from transit_route
                            where route_id={}) d,
                         (select * from transit_trip where
                              route_id={}) e
            --            where b.ord>c.start_ord and b.ord<c.
                end_ord and c.route_id=d.route_id and d.route_id=e
                .route_id
                            group by c.id, d.route_id, e.trip_id,
                                d.route_freq, d.dur, d.
                                route_speed, d.stop_freq) bb
                        group by bb.id, bb.route_id, bb.
                            route_freq, bb.route_dur, bb.
                            route_speed, bb.stop_freq, bb.l, bb.
                            dur, bb.c, bb.avg_dist)bbb
                     where aaa.id=bbb.id and aaa.id={};
                 """.format(row[2], row[3], row[1], row[1], row
                    [1], row[0]))
        conn.commit()
    print 'end_f_feature_in_buffer', datetime.datetime.now()

def f_delete_record_2():
    print 'start_f_delete_record_2', datetime.datetime.now()
```

```
    cur.execute("""
        -- delete pairs with only one point (route crossing
           segment)
        delete from seg_route_pairs where points=1;
        -- delete if dist_3d is empty!
        delete from seg_route_pairs where COALESCE(dist_3d::text
            , '') = '';
        alter table seg_route_pairs
            add column route_t  real,
            add column seg_t real;
        update seg_route_pairs a set
            route_t=route_avg_t
            from transit_route b
            where a.route_id=b.route_id;
        update seg_route_pairs a set
            seg_t=(date_part('hour', start_t)*3600+date_part('
                minute', start_t)*60+date_part('second', start_t)+
                date_part('hour', end_t)*3600+date_part('minute',
                end_t)*60+date_part('second', end_t))/2.0
        """)
    conn.commit()
    print 'end_f_delete_record_2', datetime.datetime.now()

def f_feature_diff():
    print 'start_f_feature_diff', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column freq_diff real,
```

```
            add column speed_diff real,
            add column t_diff real;
        update seg_route_pairs set
            freq_diff=(points_freq-stop_freq)/points_freq,
            speed_diff=(seg_speed-route_speed)/seg_speed,
            t_diff=seg_t-route_t
        """)
    conn.commit()
    print 'end_f_feature_diff', datetime.datetime.now()

def f_feature_combo():
    print 'start_f_feature_combo', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column combo1 real,
            add column combo2 real,
            add column combo3 real,
            add column combo4 real,
            add column combo5 real;
        update seg_route_pairs set
            combo1=length/dist,
            combo2=length/dist_3d,
            combo3=length/abs(t_diff),
            combo4=length/(dist*abs(t_diff)*abs(freq_diff)),
            combo5=length/(dist_3d*abs(freq_diff))
        """)
    conn.commit()
    print 'end_f_feature_combo', datetime.datetime.now()
```

```
def f_feature_ratio():
    print 'start_f_feature_ratio', datetime.datetime.now()
    cur.execute("""
        alter table seg_route_pairs
            add column length_r real,
            add column duration_r real,
            add column points_r real,
            add column dist_r real,
            add column dist3d_r real,
            add column freq_r real,
            add column speed_r real,
            add column combo1_r real,
            add column combo2_r real,
            add column combo3_r real,
            add column combo4_r real,
            add column combo5_r real;
        update seg_route_pairs aa set
            --multiply by 1.0 to get real result rather than
                integer
            length_r=length/l_max,
            duration_r=duration/dur_max,
            points_r=1.0*points/c_max,
            dist_r=dist/d_min,
            dist3d_r=dist_3d/d3d_min,
            freq_r=freq_diff/freq_diff_min,
            speed_r=speed_diff/v_diff_min,
            combo1_r=combo1/combo1_max,
```

```
                combo2_r=combo2/combo2_max,
                combo3_r=combo3/combo3_max,
                combo4_r=combo4/combo4_max,
                combo5_r=combo5/combo5_max
                from (select b.id, max(a.length) l_max, max(a.
                    duration) dur_max, max(a.points) c_max,
                        min(a.dist) d_min, min(a.dist_3d) d3d_min,
                            min(a.freq_diff) freq_diff_min, min(a.
                            speed_diff) v_diff_min,
                        max(a.combo1) combo1_max, max(a.combo2)
                            combo2_max, max(a.combo3) combo3_max, max(
                            a.combo4) combo4_max, max(a.combo5)
                            combo5_max
                     from seg_route_pairs a,
                         seg_route_pairs b
                     where a.start_ord<b.end_ord and b.start_ord<a.
                        end_ord and a.user_id=b.user_id
                     group by b.id) bb
                where aa.id=bb.id
            """)
        conn.commit()
        print 'end_f_feature_ratio', datetime.datetime.now()


def f_learning(learning_method, n_trees, n_iteration, variables):
    global train, test, test_acc, test_acc_len, z, z_len
    print 'start_f_learning', datetime.datetime.now()
```

```
    print "method:{}, iteration:{}, variables:{}".format(
        learning_method, n_iteration, variables)
    cur.execute("""select user_id, doy, count(1) n from
                    (select * from seg_route_pairs
--                          where route_id<400 and cast(substring(
    route,'[0-9]+') as integer)<400
                        ) a
--                  where
--                      ipere!='99999'
--                      and
--                      modes='bus' or modes='metro'
--                      and
--                      modes!='unknown'
                group by user_id, doy""")
    u_seg = []
    for row in cur:
        u_seg.append(list(row))

    train_acc=[]
    test_acc=[]
    test_acc_len=[]
    for it in range(n_iteration):
        random.shuffle(u_seg)

        # 80% of segments as train set
        train_ratio=0.8
        train=[]
        for i in range(int(train_ratio*len(u_seg))):
```

```
        cur.execute("""select user_id, id, modes, route,
            route_id, length, {}, output from seg_route_pairs
                    where user_id='{}' and doy={}""".format(
                        variables, u_seg[i][0], u_seg[i][1]))
        for j in cur:
            train.append(list(j))
    # Boolean
    train_y = [a[-1] for a in train]
    train_x = []
    # number of points, avg distance and point ratio as input
    for row in train:
        train_x.append(row[6:-1])

    if learning_method == 'rf':
        model = RandomForestClassifier(n_estimators=n_trees,
            n_jobs=-1).fit(train_x, train_y)
    elif learning_method == 'logit':
        model = LogisticRegression().fit(train_x, train_y)
    del train, train_x, train_y

    # 20% of segments as test set
    test=[]
    for i in range(int(train_ratio*len(u_seg)), len(u_seg)):
        cur.execute("""select user_id, id, modes, route,
            route_id, length, {}, output from seg_route_pairs
                    where user_id='{}' and doy={}""".format(
                        variables, u_seg[i][0], u_seg[i][1]))
        for j in cur:
```

```
            test.append(list(j))
    test_y = [a[-1] for a in test]
    test_x = []
    for row in test:
        test_x.append(row[6:-1])

    pred = [i for i in model.predict(test_x)]
    acc = sklearn.metrics.accuracy_score(test_y, pred,
       normalize=True)

    cur.execute("""
        drop table if exists u_seg_accuracy;
        create temp table u_seg_accuracy (id integer,
            start_ord integer, end_ord integer, route smallint
            , route_id smallint, output smallint, pred
            smallint)
        """)
    conn.commit()
    for i in range(len(test)):
        cur.execute("""
            insert into u_seg_accuracy select id, start_ord,
                end_ord, route, route_id, output, {} from
                seg_route_pairs where id={}
            """.format(pred[i], test[i][1]))
        conn.commit()
    cur.execute("""
        drop table if exists tii_accuracy;
```

```
create table tii_accuracy (ord integer, gps geometry,
    test boolean, test_lag boolean, correct boolean,
    correct_lag boolean, length real, route smallint,
    route_id smallint, output smallint, pred smallint)
    ;
insert into tii_accuracy select ord, st_transform(
    st_setsrid(st_makepoint(longitude, latitude),
    4326), 32618), false, false, false, false, 0, cast
    (substring(route,'[0-9]+') as int), 0, 0, 0 from
    tii;
update tii_accuracy a set
    length=l
    from (select ord, st_distance(gps, lag(gps) over
        (order by ord)) l  from tii_accuracy) b
    where a.ord=b.ord;
update tii_accuracy a set
    test=true,
    correct=true
    from (select ord, count(1) n
            from (select ord from tii_accuracy) b,
                (select * from u_seg_accuracy) c
            where b.ord>c.start_ord and b.ord<c.
                end_ord
            group by ord) d
    where a.ord=d.ord;
update tii_accuracy a set
    test_lag=lag
```

```
    from (select ord, lag(test) over (order by ord)
        lag from tii_accuracy) b
    where a.ord=b.ord;
update tii_accuracy a set
    correct=false,
    route=e.route,
    route_id=e.route_id,
    output=e.output,
    pred=e.pred
    from (select ord, route, route_id, output, pred
        from (select b.ord, b.route, c.route_id, c.
            output, c.pred, row_number() over (
            partition by b.ord order by c.output desc)
             as rnum
             from (select ord, route from tii_accuracy
                ) b,
                 (select * from u_seg_accuracy where
                    output!=pred) c
             where b.ord>c.start_ord and b.ord<c.
                end_ord
             ) d
        where rnum=1) e
    where a.ord=e.ord;
update tii_accuracy a set
    route=e.route,
    route_id=e.route_id,
    output=e.output,
    pred=e.pred
```

```
    from (select ord, route, route_id, output, pred
        from (select b.ord, b.route, c.route_id, c.
            output, c.pred, row_number() over (
            partition by b.ord order by c.output desc)
             as rnum
             from (select ord, route from tii_accuracy
                  where correct=True) b,
                  (select * from u_seg_accuracy where
                      output=pred and output=1) c
             where b.ord>c.start_ord and b.ord<c.
                 end_ord
             ) d
        where rnum=1) e
    where a.ord=e.ord;
update tii_accuracy set
    correct=false where correct=True and route!=
        route_id;
update tii_accuracy a set
    correct=true
    from (select b.ord, count(1) n
        from (select ord, route from tii_accuracy
            where correct=false) b,
             (select ord, route from tii_accuracy
                 where correct=true) c
        where abs(b.ord-c.ord)<16 and b.route=c.route
        group by b.ord
        ) d
    where a.ord=d.ord;
```

```
            update tii_accuracy a set
                correct_lag=lag
                from (select ord, lag(correct) over (order by ord
                    ) lag from tii_accuracy) b
                where a.ord=b.ord
            """)
        conn.commit()
        cur.execute("""
            select sum(case when correct and correct_lag then
                length else 0 end)/sum(case when test and test_lag
                then length else 0 end)
                from tii_accuracy where test=true and test_lag=
                    true group by test
            """)
        for row in cur:
            test_acc_len.append(list(row)[0])

    print 'test_accuracy={}, std={} (length)'.format(np.mean(
        test_acc_len), np.std(test_acc_len))
    print 'end_f_learning', datetime.datetime.now()

f_initialize()
f_transit_trip()
f_transit_route()
f_zone_route()
f_datamobile_data_fetch()
f_tii_correction()
f_seg_route_pairs()
```

```
f_update_output()
f_simple_feature()
f_delete_record()
f_feature_in_buffer()
f_delete_record_2()
f_feature_diff()
f_feature_combo()
f_feature_ratio()
f_learning('rf', 100, 10, 'route_type, length, dist_3d,
    points_freq, combo1_r, combo2_r, combo3_r, combo4_r, combo5_r'
    )
```

Bibliography

Agence métropolitaine de transport (2013). Enquête origine-destination 2013: La mobilité des personnes dans la région de Montréal - Faits Saillants. Technical report, Agence métropolitaine de transport, Montréal.

Bachu, P., Dudala, T., and Kothuri, S. (2001). Prompted recall in global positioning system survey: Proof-of-concept study. *Transportation Research Record: Journal of the Transportation Research Board*, (1768):106–113.

Bierlaire, M., Chen, J., and Newman, J. (2013). A probabilistic map matching method for smartphone GPS data. *Transportation Research Part C: Emerging Technologies*, 26:78–98.

Biljecki, F. (2010). Automatic segmentation and classification of movement trajectories for transportation modes.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bohte, W. and Maat, K. (2009). Deriving and validating trip purposes and travel modes for multi-day GPS-based travel surveys: A large-scale application in the Netherlands. *Transportation Research Part C: Emerging Technologies*, 17(3):285–297.

Bolbol, A., Cheng, T., Tsapakis, I., and Haworth, J. (2012). Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification. *Computers, Environment and Urban Systems*, 36(6):526–537.

Bose, B. K. (1994). Expert system, fuzzy logic, and neural network applications in power electronics and motion control. *Proceedings of the IEEE*, 82(8):1303–1323.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Bricka, S. and Bhat, C. (2006). Comparative analysis of global positioning system-based and travel survey-based data. *Transportation Research Record: Journal of the Transportation Research Board*, (1972):9–20.

Bricka, S., Zmud, J., Wolf, J., and Freedman, J. (2009). Household travel surveys with GPS: An experiment. *Transportation Research Record: Journal of the Transportation Research Board*, (2105):51–56.

Bricka, S. G., Sen, S., Paleti, R., and Bhat, C. R. (2012). An analysis of the factors influencing differences in survey-reported and GPS-recorded trips. *Transportation research part C: emerging technologies*, 21(1):67–88.

Burges, C. J. et al. (1996). Simplified support vector decision rules. In *ICML*, volume 96, pages 71–77. Citeseer.

Carrel, A., Lau, P. S., Mishalani, R. G., Sengupta, R., and Walker, J. L. (2015). Quantifying transit travel experiences from the users perspective with high-resolution smartphone and vehicle location data: Methodologies, validation, and example analyses. *Transportation Research Part C: Emerging Technologies*, 58:224–239.

CBC-News (2016). Waze collaboration with montreal.

Delalleau, O., Bengio, Y., and Le Roux, N. (2005). Efficient non-parametric function induction in semi-supervised learning. In *AISTATS*, volume 27, page 100.

Dietterich, T. G. (2002). Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125.

Feng, T. and Timmermans, H. J. (2013). Map matching of GPS data with bayesian belief networks. *Journal of the Eastern Asia Society for Transportation Studies*, 10:100–112.

Gelfand, S. B., Ravishankar, C., and Delp, E. J. (1989). An iterative growing and pruning algorithm for classification tree design. In *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*, pages 818–823. IEEE.

Giaimo, G., Anderson, R., Wargelin, L., and Stopher, P. (2010). Will it work? Pilot results from first large-scale global positioning system-based household travel survey in the United States. *Transportation Research Record: Journal of the Transportation Research Board*, (2176):26–34.

Gilani, H. (2005). Automatically determine route and mode of tranport using a GPS enabled phone.

Gonzalez, P. A., Weinstein, J. S., Barbeau, S. J., Labrador, M. A., Winters, P. L., Georggi, N. L., and Perez, R. (2010). Automating mode detection for travel behaviour analysis by using global positioning systems-enabled mobile phones and neural networks. *IET Intelligent Transport Systems*, 4(1):37–49.

Greaves, S., Fifer, S., Ellison, R., and Germanos, G. (2010). Development of a global positioning system web-based prompted recall solution for longitudinal travel surveys. *Transportation Research Record: Journal of the Transportation Research Board*, (2183):69–77.

Handy, S., Cao, X., and Mokhtarian, P. (2005). Correlation or causality between the built environment and travel behavior? Evidence from Northern California. *Transportation Research Part D: Transport and Environment*, 10(6):427–444.

Ho, T. K. (1995). Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE.

Horvath, G. (2003). CMAC neural network as an SVM with B-Spline kernel functions. In *Instrumentation and Measurement Technology Conference, 2003. IMTC'03. Proceedings of the 20th IEEE*, volume 2, pages 1108–1113. IEEE.

Hudson, J. G., Duthie, J. C., Rathod, Y. K., Larsen, K. A., and Meyer, J. L. (2012). Using smartphones to collect bicycle travel data in Texas. Technical report.

Joubert, J. W. and Axhausen, K. W. (2011). Inferring commercial vehicle activities in Gauteng, South Africa. *Journal of Transport Geography*, 19(1):115–124.

Liao, L., Patterson, D. J., Fox, D., and Kautz, H. (2007). Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Miller, A., Blott, B., et al. (1992). Review of neural network applications in medical imaging and signal processing. *Medical and Biological Engineering and Computing*, 30(5):449–464.

Montini, L., Rieser-Schüssler, N., Horni, A., and Axhausen, K. (2014). Trip purpose identification from GPS tracks. *Transportation Research Record: Journal of the Transportation Research Board*, (2405):16–23.

Murphy, K. P. (2012). Machine learning: A probabilistic perspective, adaptive computation and machine learning.

Nitsche, P., Widhalm, P., Breuss, S., Brändle, N., and Maurer, P. (2014). Supporting large-scale travel surveys with smartphones: A practical approach. *Transportation Research Part C: Emerging Technologies*, 43:212–221.

Ogle, J., Guensler, R., Bachman, W., Koutsak, M., and Wolf, J. (2002). Accuracy of global positioning system for determining driver performance parameters. *Transportation Research Record: Journal of the Transportation Research Board*, (1818):12–24.

Osuna, E. and Girosi, F. (1998). Reducing the run-time complexity of support vector machines. In *International Conference on Pattern Recognition (submitted)*.

Patterson, D. J., Liao, L., Fox, D., and Kautz, H. (2003). Inferring high-level behavior from low-level sensors. In *UbiComp*, pages 73–89. Springer.

Patterson, Z. (2017). MTL Trajet 2016. Paper presented at the 11th International Conference on Travel Survey Methods, Esterel, Quebec. Available at: itinerum.ca/documents.html.

Patterson, Z. and Fitzsimmons, K. (2016). Datamobile: Smartphone travel survey experiment. *Transportation Research Record: Journal of the Transportation Research Board*, (2594):35–43.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., and Srivastava, M. (2010). Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):13.

Saddier, S., Patterson, Z., Johnson, A., and Chan, M. (2016). Mapping the jitney network with smartphones in Accra, Ghana: The AccraMobile experiment. *Transportation Research Record: Journal of the Transportation Research Board*, (2581):113–122.

Safi, H., Assemi, B., Mesbah, M., Ferreira, L., and Hickman, M. (2015). Design and implementation of a smartphone-based travel survey. *Transportation Research Record: Journal of the Transportation Research Board*, (2526):99–107.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press.

Schüssler, N. and Axhausen, K. W. (2009). Processing GPS raw data without additional information. *Transportation Research Record*, 2105:28–36.

Shafique, M. A. and Hato, E. (2015). Use of acceleration data for transportation mode prediction. *Transportation*, 42(1):163–188.

Shahin, M. A., Jaksa, M. B., and Maier, H. R. (2001). Artificial neural network applications in geotechnical engineering. *Australian geomechanics*, 36(1):49–62.

Shai, S. and Shai, B. (2014). Understanding machine learning: From theory to algorithms.

Shen, L. and Stopher, P. (2013). Should we change the rules for trip identification for GPS travel records? In *Proceedings of the 36th Australasian Transport Research Forum ATRF, Brisbane, Australia*, pages 2–4.

Shen, L. and Stopher, P. R. (2014). Review of GPS travel survey and GPS data-processing methods. *Transport Reviews*, 34(3):316–334.

Stenneth, L., Wolfson, O., Yu, P. S., and Xu, B. (2011). Transportation mode detection using mobile phones and GIS information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM.

Stopher, P., FitzGerald, C., and Xu, M. (2007). Assessing the accuracy of the Sydney household travel survey with GPS. *Transportation*, 34(6):723–741.

Stopher, P., FitzGerald, C., and Zhang, J. (2008). Search for a global positioning system device to measure person travel. *Transportation Research Part C: Emerging Technologies*, 16(3):350–369.

Stopher, P. R. and Greaves, S. P. (2007). Household travel surveys: Where are we going? *Transportation Research Part A: Policy and Practice*, 41(5):367–381.

Therneau, T. M., Atkinson, E. J., et al. (1997). An introduction to recursive partitioning using the RPART routines. Technical report, Technical report Mayo Foundation.

Tourangeau, R., Zimowski, M., and Ghadialy, R. (1997). *An introduction to panel surveys in transportation studies*. National Opinion Research Center.

Tsui, J. B.-Y. (2005). *Fundamentals of global positioning system receivers: A software approach*, volume 173. John Wiley & Sons.

Tsui, S. and Shalaby, A. (2006). Enhanced system for link and mode identification for personal travel surveys based on global positioning systems. *Transportation Research Record: Journal of the Transportation Research Board*, (1972):38–45.

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231.

Wagner, D. P. (1997). Lexington area travel data collection test: GPS for personal travel surveys. *Final Report, Office of Highway Policy Information and Office of Technology Applications, Federal Highway Administration, Battelle Transport Division, Columbus*, pages 1–92.

White, C. E., Bernstein, D., and Kornhauser, A. L. (2000). Some map matching algorithms for personal navigation assistants. *Transportation research part c: emerging technologies*, 8(1):91–108.

Witayangkurn, A., Horanont, T., Ono, N., Sekimoto, Y., and Shibasaki, R. (2013). Trip reconstruction and transportation mode extraction on low data rate GPS data from mobile phone. In *Proceedings of the international conference on computers in urban planning and urban management (CUPUM 2013)*, pages 1–19.

Wolf, J., Guensler, R., and Bachman, W. (2001). Elimination of the travel diary: Experiment to derive trip purpose from global positioning system travel data. *Transportation Research Record: Journal of the Transportation Research Board*, (1768):125–134.

Wolf, J., Oliveira, M., and Thompson, M. (2003). The impact of trip underreporting on vmt and travel time estimates: Preliminary findings from the California statewide household travel survey GPS study. *Transportation Research Record*, 1854:189–198.

Wolf, J. L. (2000). *Using GPS data loggers to replace travel diaries in the collection of travel data*. PhD thesis, School of Civil and Environmental Engineering, Georgia Institute of Technology.

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390.

Wong, B. K. and Selvi, Y. (1998). Neural network applications in finance: A review and analysis of literature (1990–1996). *Information & Management*, 34(3):129–139.

Wu, L., Yang, B., and Jing, P. (2016). Travel mode detection based on GPS raw data collected by smartphones: A systematic review of the existing methodologies. *Information*, 7(4):67.

Xiao, Y., Low, D., Bandara, T., Pathak, P., Lim, H. B., Goyal, D., Santos, J., Cottrill, C., Pereira, F., Zegras, C., et al. (2012). Transportation activity analysis using smartphones. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 60–61. IEEE.

Yazdizadeh, A., Patterson, Z., and Farooq, B. (2018). An automated approach from GPS traces to

complete trip information. Submitted to the *International Journal of Transportation Science and Technology*. Submitted: February, 2018.

Yoldi, C. and Lebena, A. (2002). Reduced acquisition time for GPS cold and warm starts. US Patent App. 09/945,451.

Zahabi, S. A. H., Ajzachi, A., and Patterson, Z. (2017). Transit trip itinerary inference with GTFS and smartphone data. Technical report.

Zheng, Y., Liu, L., Wang, L., and Xie, X. (2008). Learning transportation mode from raw GPS data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256. ACM.

Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation.