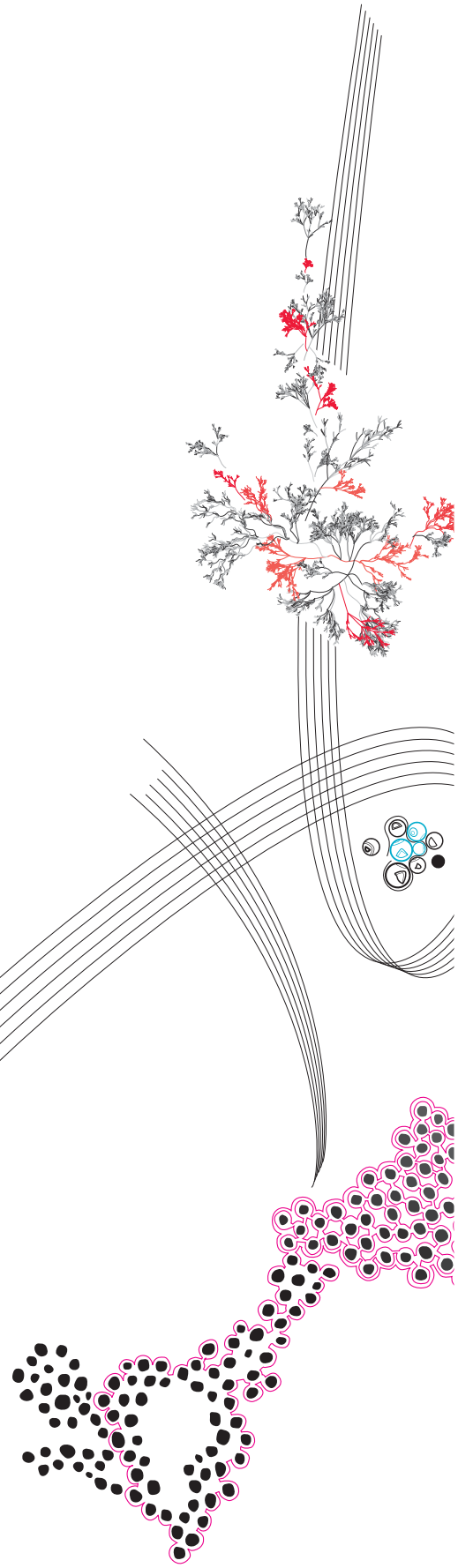

Graduation at Mobidot

A Markovian approach to mobile user classification with semantic location data in mobility chains

Jop Zwienenberg

Supervisors:
Mobidot: Johan Koolwaaij
UT: Maria Vlasiou and Marc Uetz

February-August 2022

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

This thesis, titled 'A Markovian approach to mobile user classification with semantic location data in mobility chains' was written to fulfill the graduation requirements of the master Applied Mathematics at the University of Twente. The research was performed from February 2022 up to August 2022 at the company Mobidot in Enschede.

First of all, I would like to thank my supervisor Johan Koolwaaij from Mobidot for his essential support during the entire time of my Master thesis. He was always willing to answer data-related questions, which often ended up in interesting discussions. These discussions really helped me in making progress. I also want to thank my other colleagues at Mobidot for providing me with a good working environment. I really liked the fact that during my graduation period, Jean Morillo was also graduating (working on his Bachelor final thesis) at the company. Since the topics of our assignments were overlapping, we could help each other where needed.

Furthermore, I would like to thank my two supervisors from the University of Twente: Maria Vlasiou and Marc Uetz. Marc Uetz was already my supervisor during the internship. Maria Vlasiou joined as a second supervisor, since at the start of the graduation period we realised someone who would feel more at home in the field of stochastic Markov chains would be welcome. You complemented each other well and provided me with useful insights from a more mathematical perspective. I really appreciated the luxury of having multiple supervisors with all their own crucial inputs. I want to thank Hanyuan Hang for being part of the graduation committee.

Finally, I would like to thank my friends and family, providing me with positive energy and necessary distractions. I want to explicitly thank Rutger Mauritz, who was graduating during the same period as me. In the frequent meetings we had, we discussed our progress, which made me realise that I was not the only one who was struggling at times. I also really appreciate it that you read and provided feedback on my some parts of my thesis. I hope you will enjoy reading my thesis.

Jop Zwienenberg, August 2022

Abstract

The large amount of mobility behavior data of people collected by companies can provide insights into how people move over the day. The software of Mobidot captures via GPS 24/7 the mobility behavior of Dutch individuals and groups over the whole world via their mobile phones. Mobidot has a rule-based method to determine the purposes of trips during the activity recognition phase. Examples of activities are working, shopping, and leisure. The company's ultimate overall goal is to find a method to build a complete user profile which serves as an input to an improved activity recognition via the rule-based method. The improved activity recognition would provide valuable answers to Mobidot's customers for market research, panel surveys, urban policy development, and impact evaluation. As an example, the data can be used as an input for government policies to achieve a smoother and better distributed traffic flow.

In this thesis, we work partially towards this goal by developing two different binary classification models for classifying two types of users, users with and without a job. We use the discrete time Markov model (DTMC) and the long short-term memory (LSTM) neural network for this. The choice for this specific binary classifier is - sort of - arbitrary, yet serves as a building block for more sophisticated and more extensive classifiers. For training the classifiers, we use a user's daily semantic location sequences over a certain period with known job status as input. It is shown that a DTMC performs well in classifying these types of users by comparing its performance in the form of the balanced accuracy with the performance of the baseline LSTM neural network. Classification is done by calculating the Jensen-Shannon distance between users, which can be interpreted as a distance between their Markov models.

The outcomes of this thesis suggest that DTMC models can be a useful building block in the design of methods to generate user profiles.

Keywords: user classification, Markov models, LSTM, semantic location, mobility chains, mobile phone, NVP

Contents

Glossary

accuracy the fraction of predictions the model got right (in accordance with the ground truth).

activity purpose of a trip. Working, shopping, and leisure are examples of possible activities.

chain total travel, including both directions, such as home-work-home (which is not necessarily a full day from 00:00-23:59). For an illustration, see Figure 1.

chain type shows the start and end location of the whole chain, with all the other locations (invisible) in between. Includes: home-home, home-overnight, overnight-home, overnight-overnight. For an illustration, see Figure 1.

destination the end point of a trip.

destination type the functionality of the end point of a trip (e.g. office or supermarket).

geographical location place on earth, represented by two coordinates, latitude and longitude, which specify the north-south and east-west position of a place on the earth's surface, respectively.

ground truth actual demographic characteristics of users, opposed to estimations of them.

journey travel in one direction from origin to destination using one or more modalities with a significant stay. For an illustration, see Figure 1.

Kantar international market research agency specialized in data, insights and consultancy.

map matching allocating a sequence of sensed location measurements onto a infrastructural map, determining the path that has been travelled.

modality may refer to two things. In general: mode of transport. This includes: bike, car, foot, plane and public transport. In Chapter 3 in the context of stay- and trip-modalities it means the functionality of a location.

modality detection automatically deducing which modality is used during certain trips and for how long by means of a device that measures travel behaviour.

origin the starting point of a trip.

panel group of people who have given their consent to taking part in different surveys or feedback investigation processes.

point of interest a specific location that is regarded as useful or interesting, such as restaurants, bars and shops.

semantic location this location carries, next to the coordinates, additional information about the meaning of the location. For example, semantic locations reveal if a location is a home, office or restaurant, thereby adding human relevance and functionality to the geographic location. The semantic destination type of a user during a certain time period is in the remainder of this thesis referred to as 'location'.

significant stay depending on the stay type, having a long stay duration in a place. So, it is objectively defined for different stay types. For most types, a stay of at least 15 minutes is seen as significant. But there are also types for which a shorter duration is already seen as significant, such as an office building, and types for which a longer duration is needed, such as the train station..

stay a residence in a place. For an illustration, see Figure 1.

stay type the functionality of a residence in a place (e.g. office or supermarket).

trajectory sequence of locations over time.

trip travel in one direction from origin to destination using one or more modalities. For an illustration, see Figure 1.

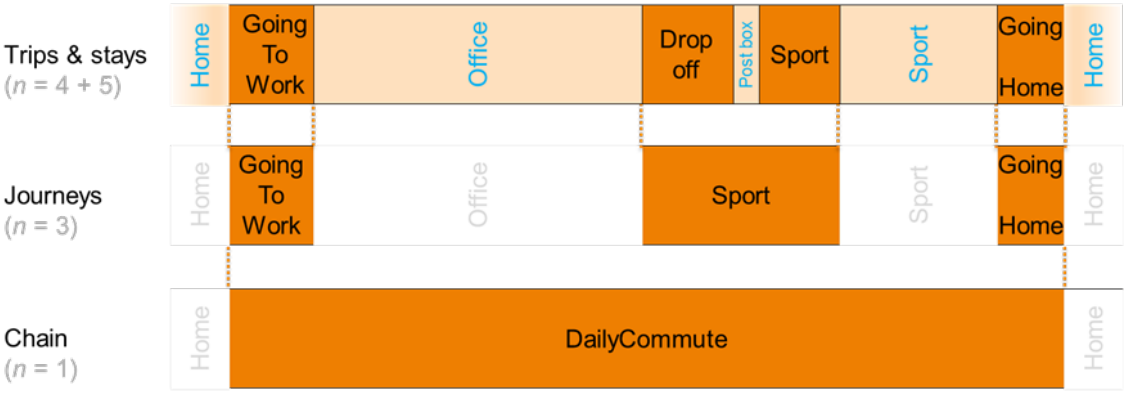

FIGURE 1: An example of how a home-home daily commute chain consists of 3 journeys, where the second journey has a non-significant stay and consists of 2 trips. For example, the activity after the second journey is 'going to sports after work'. There are five stays: home, office, post box, sport, and home again.

1 Introduction

Note that we started this thesis with a glossary, which is an alphabetical list of important terms with the definitions for those terms which are useful for the remainder of this report.

1.1 The use of location data in classifying users

Mobile phones have become indispensable in our lives, as we take them everywhere we go. Phones have build-in functionalities to keep track of their location, such as GPS and Wi-Fi. Location data are being sent to mobile-phone operators and to companies that gain access to the data through mobile-phone applications (app). Many companies nowadays are able to not only keep track of the geographical locations (latitude and longitude) of the user but also to give semantic meaning to those locations, adding human relevance to the locations. This process is also referred to as *reverse geocoding*. To this end, companies also use points of interest (POIs) and even the user's personal mobility data. The large amount of mobility behavior data of people collected by companies can provide insights into how people move over the day.

Mobidot is an example of such a company. Mobidot's MoveSmarter software platform captures via GPS 24/7 the mobility behavior of Dutch individuals and groups over the whole world via their mobile phones. In this way, Mobidot collects travel data of many travelers on a daily basis, from which half are in the 'Nederlands Verplaatsingspanel (NVP)', thus resulting in many trips to be analyzed every day. This NVP-panel is a subset of the NIPObase-panel of Kantar, so the other half is only in the NIPObase-panel. In this representative panel, the panel members give permission to Kantar to anonymously collect their mobility, location and activity data via the app NIPObase. Trips are sensed on the mobile phone using different sensing strategies, and submitted to the back-end for further analysis, where the analysis process is modular and consists of multiple phases, including map matching, modality detection and activity recognition. Thus, in addition to keeping track of the user's geographical locations, Mobidot also gives semantic meaning to them. This is determined with the help of third party data of OpenStreetMap (OSM), which includes many points of interest (POIs). OSM is a collaborative project that provides a free editable geographic database of the world. Based on *Geographic Information Systems (GIS) layers* and mapping techniques, each end location of a trip is tagged as the most likely functionality of that location by using algorithms. GIS is a type of database consisting of geographic data, combined with software tools for managing, analyzing, and visualizing those data in layers. Tagging end locations of trips is not part of this research, since it is a whole research in itself how to derive the semantic locations from geographical locations [1, 2, 3]. The semantic locations data (in the form of the functionality of a location, such as home, office or restaurant) can give an indication about a user's mobility behavior and function as an input for classifying a user's demographic characteristics.

1.2 Goals and scope

Mobidot has a rule-based method to determine the purposes of trips during the activity recognition phase. Examples of activities are working, shopping, and leisure. The company's ultimate overall goal is to find a method to build a complete user profile which serves as an input to an improved activity recognition via the rule-based method. The improved activity recognition would provide valuable answers to Mobidot's customers for market research, panel surveys, urban policy development, and impact evaluation. As an

example, the data can be used as an input for government policies to achieve a smoother and better distributed traffic flow. Another example, where a high quality activity recognition is crucial, is the impact evaluation of the advice to work remotely during the Covid-19 lockdowns in 2020 and 2021. It gives the government insights into the effectiveness of this measure. Note that we did not include a more elaborate description of the context of Mobidot regarding the rule-based method used for activity recognition due to confidentiality.

1.3 Approach

In this thesis, we work partially towards this goal by developing two different binary classification models for classifying two types of users, users with and without a job. We use the discrete time Markov model (DTMC) and the long short-term memory (LSTM) neural network for this. The choice for this specific binary classifier is - sort of - arbitrary, yet serves as a building block for more sophisticated and more extensive classifiers. For training the classifiers, we use a user's daily semantic location sequences over a certain period with known job status as input. A simple example of a semantic location sequence between 7 am - 10 am for 2 days at an hourly resolution is shown in Figure 2. It is shown that a DTMC performs well in classifying these types of users by comparing its performance in the form of the balanced accuracy with the performance of the baseline LSTM neural network. Classification is done by calculating the Jensen-Shannon distance between users, which can be interpreted as a distance between their Markov models.

7:00	**8:00**	**9:00**
Home		School
Home	School	Office

FIGURE 2: Input: User's location sequences between 7 am-10 am for 2 days at an hourly resolution.

2 Related work

There are several classification and clustering models known for classifying a user. As we stated in Chapter 1, for training the classifiers, we use a user's daily semantic location sequences over a certain period with a known classification for a certain demographic characteristic as input (in this thesis job status). The output is a classification, and its accuracy is assessed in the form of the so-called balanced accuracy.

2.1 Classification models

2.1.1 Standard machine learning models

Users can be classified according to different criteria and data-sets used in related work contain a wide variety of information about the users and their trips made.

In one paper, users are divided into four classifications according to user movement patterns and trajectory complexity [4]. They define four different user types, namely residents, regular commuters, irregular commuters and others. The characteristics of different features are defined to build a machine learning data set. The user type is mainly determined by features that deal with the number of uniquely visited geographical locations and the stay duration of the user at different locations over different time periods. The naïve Bayes, decision tree and K-nearest neighbor (KNN) algorithms in the scikit-learn toolkit are used as supervised machine learning classification models [5].

In a different paper, the naïve Bayes classifier is used to classify university student and professors based on their features: visit in types of place, speed of movement or transportation mode and user movement patterns [6]. They are divided into five categories, namely undergraduate student, graduate student, research student, employee/professor and non-residential students. Both papers [4, 6] do not have access to the semantic locations of users, only geographical locations. Moreover, they do not have access to known users' classifications for a certain demographic characteristic, since they developed classification criteria themselves, mainly based on experience. We shortly explain the aforementioned machine learning models.

Naïve Bayes classification: The naïve Bayes classifier is based on Bayes' theorem. The 'naïve' in the name of this classifier comes from the simplifying assumption this classifier makes: it assumes that every pair of features, given the value of the class variable, is independent. Bayes' theorem states the following: given a class variable y and a dependent feature vector $(x_1, \ldots, x_n)$ [7], using the independence assumption that for all i

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

we get the relationship,

$$P(y|x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \ldots, x_n)}. \tag{2}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, classification takes place via the maximum-likelihood method, that is

$$\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i|y). \tag{3}$$

Decision trees: Decision trees can also be used for classification. A model is created that predicts the value of a target variable y by learning simple decision rules deduced from the data features $(x_1, \ldots, x_n)$. Figure 3 provides a graphical example of a decision tree, which indicates whether or not a customer is likely to purchase a computer [8]. The categorical features in this example are 'age', 'student' and 'credit rating':

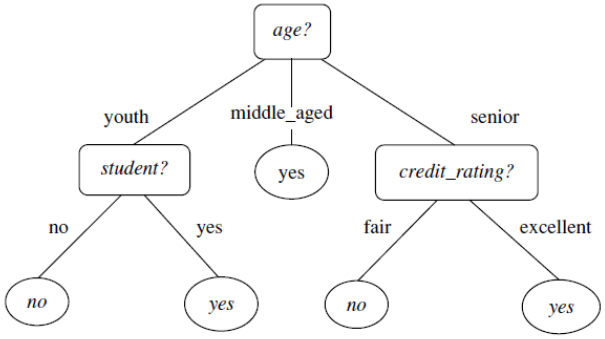

FIGURE 3: A decision tree which indicates whether or not a customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either 'buys computer'=yes or 'does not buy computer'=no).

There are several types of decision tree algorithms, of which Iterative Dichotomiser 3 ($ID3$) was the first one. The successors of this algorithm are $C4.5$ and $C5.0$. The difference between these newer algorithms and $ID3$ is that $C4.5$ and $C5.0$ are able to handle not only categorical features, but also continuous features. The algorithm Classification and Regression Trees (CART) is very similar to $C4.5$ and $C5.0$, but it differs in that it supports regression as well. In the scikit-learn toolkit an optimized version of the CART algorithm is used [5].
For classification the Gini index function G is used which provides an indication of how 'pure' the leaf nodes are, that is, how mixed the training data assigned to each node is, where p_k is the proportion of training instances with class k in the leaf node of interest:

$$G = \sum_k p_k(1 - p_k). \tag{4}$$

A node that has all classes of the same type (perfect class purity) will have $G = 0$, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a $G = 0.5$.

K-nearest neighbors: K-nearest neighbor (KNN) classifiers are based on a distance metric that measures the similarity between two feature vectors. The default metric used in the scikit-learn toolkit is the Euclidean distance function. Let A and B be the feature vectors $A = (x_1, x_2, \ldots, x_n)$ and $B = (z_1, z_2, \ldots, z_n)$. To determine the distance between A and B, the normalized Euclidean distance [5] is often used, so

$$dist(A, B) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - z_i)^2}{n}}. \tag{5}$$

This function is used to calculate the distance between feature values of new observations in A and feature values of the data points from the training set in B. The 'K' in K-nearest

neighbor stands for the number of relevant neighbors from which we consider their classes. So in K-nearest neighbors, given a vector A of features that are needed to classify and vectors $B_1, \dots B_j$ of potential users' classes feature vectors, A is classified not as being from class $k \in \{1, \dots, j\}$, but as being from a subset of $\{1, \dots, j\}$ of cardinality K. The subset is chosen as those vectors from B_1 to B_j with the smallest Euclidean distance from A.

When using $K = 1$, the new observation is assigned to the class of the closest data point from the training set. When $K > 1$, an observation is classified by a plurality vote of its neighbors, with it being assigned to the class most common among its K nearest neighbors.

2.1.2 Markov model

The model we discuss in depth in this thesis is a discrete time Markov chain, which is represented by a conditional distribution function of visited locations given the visited locations of the user in the past. It is assumed that the semantic location sequences have the first-order Markov property, which means that the probability of moving to the next location in the sequence at a certain time period $t+1$ depends only on the present location at period t and not on the previous locations. The reasonableness of this Markov assumption is commented on in Chapter 4.2.1. In other papers this model is mainly used for location prediction [4, 9, 10, 11, 12, 13]. The Markov model will be defined and explained at length in Chapter 4.

2.1.3 Long-Short Term Memory network

One of the models we also discuss in depth in this thesis is the Long Short-Term Memory (LSTM) network designed by Hochreiter and Schmidhuber [14]. This is an artificial recurrent neural network (RNN) capable of learning both short-term and long-term dependencies. The architecture of this network has been improved to deal with the long-term dependency problem even better [15, 16, 17, 18]. How RNNs were changed in order to set up an LSTM network is explained in detail in [19, 20]. A common LSTM classifier is build out of a cell state, an input gate, an output gate and a forget gate. The three gates coordinate what information goes into and out of the cell A, which is shown in Figure 29. The gates consist of a sigmoid layer σ and a pointwise multiplication operation $\times$. The layer gives a value between 0 and 1 as output, which describes how much of the information should be let through to eventually the new cell state. A value of 0 means that nothing is going through, while 1 means that everything is let through. The cell stores values over arbitrary time intervals. In this way, information that is let through the gates is sent down the chain of sequences in order to make predictions. More information on this follows in Chapter 6.

2.2 Clustering methods

As previously mentioned, papers [4, 6] do not have access to known users' classifications for a certain demographic characteristic and solved this by developing classification criteria themselves, mainly based on experience. Another way to solve this is by applying clustering methods. *Clustering* a set of data points means grouping them in such a way that data points in the same group (called a cluster) are more similar (e.g. by using Euclidean distance between points) to each other than to those in other clusters. Since we focus on classification methods in this thesis, we do not elaborate on these clustering methods. We will briefly mention some of these methods. In paper [21] K-means is used to cluster users

based on their daily activity sequences during the weekdays and the weekend.

K-means: clusters data by separating samples in n groups of equal variance. The algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster *centroids*. The algorithm aims to choose centroids that minimise the *inertia*, or *within-cluster sum-of-squares criterion*:

$$\sum_{i=0}^{n} min_{\mu_j \in C}(||x_i - \mu_j||^2) \tag{6}$$

Inertia can be recognized as a measure of how internally coherent clusters are.

Other standard clustering methods such as K-medoids, hierarchical clustering and spectral clustering can be applied to cluster users based on similar mobility patterns that do not exactly have to match over time [22].

K-medoids: similar to the K-means method, but differs from the K-means method in that K-medoids chooses actual data points as centers (called medoids). In the K-means method, the center of a cluster is not necessarily one of the input data points.

Hierarchical clustering: a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- Agglomerative: This is a *bottom-up* approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- Divisive: This is a *top-down* approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Spectral clustering: this is a partitioning method, just like the K-means and K-medoids methods. It differs from the K-means method in that it first makes use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions (e.g. by K-means). The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset.

In paper [22] clustering is done with the Lifestyle-based clustering (LBC) approach, which follows a three-phase procedure. First, each user is profiled by a stochastic Markov model using their geographical location sequences as an input. Secondly, the Jensen-Shannon distance for every pair of users over their Markov models (explained in Chapter 5) is calculated and represented in a pairwise distance matrix. Finally, one of the standard clustering algorithms are performed on the matrix, grouping the users.

2.3 Contribution

The contribution of this thesis is threefold. The contribution is the design of a classifier for classifying users with and without a job. Therefore, we decided to use a classification algorithm instead of a clustering one (or a classification based on self-made criteria), because we already know many of the demographic characteristics of the users in the panel. The second contribution is the design of the specific Markov model. In other papers location sequences are used as an input for the prediction of a next location [4, 11, 12, 13].

However, there has been no paper that does the classification for a demographic question as the output of the model. Therefore, developing the specific Markov model is new. In order to evaluate its merits, we also compare it with a neural network (LSTM).
The third contribution is the unique dataset that is studied. It is a real dataset that contains all the trips and stays of 171 users over 3 months with corresponding information about the time, duration, modality, location etc. of the trip. More on this in Chapter 3.

2.4 Structure of the report

The remainder of this report is structured as follows. In Chapter 3, we give more information about the data-set used for evaluation of the models, which are presented in Chapters 4 and 6. In Chapter 4, we give a mathematical overview of discrete time Markov chains. We describe several modeling options, including which location type(s) to incorporate in the model and explain the choice of the distance metric to measure differences between Markov models for classification by means of an example. In Chapter 6, we discuss the benchmark model, the long short-term memory (LSTM) neural network and describe which layers we added to the model, in order to obtain good performance. Chapter 7 contains descriptions of metrics for evaluating and comparing the performances of both models. In Chapter 8, we present the final results. Finally, in Chapter 9, we draw the main conclusions and discuss the possible limitations of our approach. In this section we also present ideas for future research.

3 Data understanding and preparation

In this section the data is described, in particular how the raw data has been transformed into a useful data-set to be used as input for the classification models.

3.1 Data description

Before the raw data from the panel users can be useful, we first need a good understanding of it. Our evaluation is conducted on a real-life data-set of mobile phone users provided by Mobidot. The raw data contains the mobility trip data of 171 anonymous cellular users over a period of three months from 1 November 2021 until 31 January 2022. It is anonymous, in that each user has been assigned a random user identity number. As described in Chapter 1, the trips of panel users are registered on the mobile phone using different sensing strategies. Afterwards, these trips are submitted to the back-end office for further analysis, with as goal to determine all trip features as accurately as possible. The data covers also places outside of the Netherlands, for example if a panel user goes abroad on a holiday.
To get a better understanding of the data, we look at a few rows of a user's mobility pattern in which a temporal resolution of 1 hour is taken. An hourly resolution here means that per hour all trips/stays are recorded, even when there is an hour in which no location change is detected (mostly during night hours). However, even when taking an hourly resolution, there are hours in which no user's location was detected. This means that the higher the resolution, the more time periods we have with missing location's data. Note that in Figure 4 this is not the case. Also, for the other 170 users the data is almost perfect, that is, almost each hour during each of the 91 days has a location detection. Note that when taking a different (random) sample of users, the dataset would potentially be less perfect leading to less reliable computational results, which explains why this dataset was taken. From the parts of the data and results we provide in this report it is not possible to trace back the corresponding user, since we do not use the geographical locations.

intervaltime	seconds	intervalduration	user_id	trip_modality	trip_destination_type	hod	hod_decimal	dow	weekno	Age	Work
1-11-2021 16:00	0	1390	5220	Stay@home	house	16	16.0000	1	2021-44	65.0	2.0
1-11-2021 16:00	1390	1558	5220	Foot	house	16	16.3861	1	2021-44	65.0	2.0
1-11-2021 16:00	2948	652	5220	Stay@home	house	16	16.8189	1	2021-44	65.0	2.0
1-11-2021 17:00	0	3600	5220	Stay@home	house	17	17.0000	1	2021-44	65.0	2.0
1-11-2021 18:00	0	3600	5220	Stay@home	house	18	18.0000	1	2021-44	65.0	2.0
1-11-2021 19:00	0	3600	5220	Stay@home	house	19	19.0000	1	2021-44	65.0	2.0
1-11-2021 20:00	0	3600	5220	Stay@home	house	20	20.0000	1	2021-44	65.0	2.0
1-11-2021 21:00	0	708	5220	Stay@home	house	21	21.0000	1	2021-44	65.0	2.0
1-11-2021 21:00	708	970	5220	Car	industrial	21	21.1967	1	2021-44	65.0	2.0
1-11-2021 21:00	1678	1922	5220	Stay@office	industrial	21	21.4661	1	2021-44	65.0	2.0

FIGURE 4: Mobility pattern of a panel user with ID 5220 on 1 November 2021 between 16:00-21:00.

Many trip/stay features are collected, but we only include those that are relevant for our research. Note that '*hod*' and '$hod_{decimal}$' are redundant features, since we can derive them from the 'intervaltime' and 'seconds' columns. For the sake of clarity, we still included them in the trip/stay features:

- Intervaltime: Date (dd-mm-yyyy) and hour (hh:00) at which the trip took place.

- Seconds: Seconds counted from the start of the hour (hh:00) at which the trip/stay started.
- Intervalduration: The interval time length of a trip/stay.
- User-id: Random user identity (ID) number.
- Trip-modality: The trip/stay modalities of the (end) location of a trip/stay as described below.
- Trip-destination-type: The trip destination types, which reveal the functionalities of the (end) stay locations.
- Hour of day (hod): Hour of the day (0-24) at which a trip/stay takes place.
- Hour of day decimal (hod-decimal): Hour of the day at which a trip/stay takes place, including the seconds counted from the start of the hour.
- Day of week (Dow): Day of the week (1-7).
- Week-number (weekno): Year and the week-number (0-52).
- Age: Age of the panel user, starting at 19 years old.
- Work: User characteristic of the panel user known via Kantar. Below we mention the subcategories with their corresponding number.

With the help of OpenStreetMap (OSM), each end location of a trip is tagged by the most likely functionality of that location (also called 'modality'), namely the stay- and trip modalities. The stay modalities are: 'stay@home', 'stay@office' and 'stay'. Here, 'stay' can be divided into different types of stays, for example: community centre, shop, house or supermarket. The trip modalities are 'foot','bike', 'car', 'public transport', 'plane' and 'other'. There are 274 different locations, also including less 'popular' locations like e.g. a history museum and a toys shop. In Chapter 10 a full list of the 274 locations is included. Via Kantar, different user characteristics are known, namely gender, urbanisation, family cycle, educational level, job, car possession, gross annual income household, driver's licence possession, postcode, age and household size. Since in this research, we focus on the question of classifying users based on their employment status, because of confidentiality we only mention the corresponding subcategories:

```
Job
  ENTREPRENEUR(1),
  EMPLOYED(2),
  GOVERNMENTAL(3),
  INCAPACITATED(4),
  UNEMPLOYED(5),
  RETIRED(6),
  STUDYING(7, "15+"),
  AT_HOME(8, "Incl. < 15 jr"),
  UNKNOWN(9);
```

FIGURE 5: User's job characteristics that are known at Kantar.

We now discuss the preprocessing steps that are performed before using this data as an input for the classification models, which are introduced in Chapters 4 and 6.

3.2 Data preprocessing

As seen in Figure 5, users can take on 9 different job characteristics. Users without a label and with the label 'unknown' (9) are removed from the dataset. The characteristics 'entrepreneur' (1), 'employed' (2) and 'governmental' (3) represent users with a job, also referred to as 'job' users. 'Incapacitated' (4), 'unemployed' (5), 'retired' (6) and 'at-home' (8) represent users without a job, also referred to as 'jobless' users. We also leave out users classified as 'studying' (7), since there are many students who both study and have a job. Users can only choose one main job characteristic when joining the panel (with an update being possible every year). As such, having such a student in the training set of the 'jobless' users could result in a bad classifier. Then there is also the case of users being a 'jobless' user according to Kantar, since they perform voluntary work. Looking at a user's mobility behavior during a certain period, it could be very difficult to distinguish those volunteers from employed workers.
As mentioned above, the job characteristics of users are updated every year. This implies that it may occur that a user changes his/her characteristic during the year and thus also changes his/her mobility behavior accordingly, while they still have the old characteristic in the database. It is not in the scope of this research to detect those 'outliers', so we have to deal with possibly noisy data.

3.3 Location sequence complexity analysis

To get more insights into the 'complexity' of the location sequences in our dataset, we determine the *mobility location sequence entropy* (MLSE) [23]. In information theory, physics and other fields, entropy is used to determine the (lack of) chaos of a system. In the context of our research, it can be used to determine the 'complexity' of location sequences. A larger MLSE implies that the next location of the user is more uncertain, thus the sequences are more 'complex' and most probably, more difficult for classification. On the other hand, a smaller MLSE means a more regular mobility behavior of a user and a more predictable mobility pattern. The MLSE is calculated in the following way.

- We assume an hourly resolution, that is, we have 24 time periods each day, $t_i, i = 1, 2, \ldots, 24$ and n different locations $j = 1, 2, \ldots, n$ visited during the whole time span of 91 days.
- We determine the probability of visiting a certain location during a certain time period t_i by

$$p_{ij} = \frac{T_{ij}}{T_i} \ \forall i, j, \tag{7}$$

  where T_i is the total duration of the time period, and T_{ij} the total duration of staying at a location j during period t_i. Both are accumulated again over the whole time span of 91 days. So $p_{ij} = \frac{3}{91}$ for a certain time period t_i and location j if the user stays at location j for 3 hours ($T_{ij} = 3$) during the total time period of 91 hours ($T_i = 91$).

Definition 3.1. *The MLSE of a user during time period t_i is*

$$H(t_i) := \sum_{j=1}^{n} p_{ij} log(\frac{1}{p_{ij}}). \tag{8}$$

Thus, if during a time period t_i a user only visits one location on every single day of the 91 days, $log(\frac{1}{p_{ij}}) = log(\frac{T_i}{T_{ij}}) = log(1) = 0$, which leads to $H(t_i) = 0$. However, when a users visits more than one location in a fixed time period t_i and over a given time horizon, MLSE increases.

Definition 3.2. *The cumulative MLSE of a user over all days is determined in the following way.*

$$H_{day} := \sum_{i=1}^{24} H(t_i). \tag{9}$$

So, H_{day} is a number that expresses a user's 'complexity' of location sequences over the time horizon of 91 days.

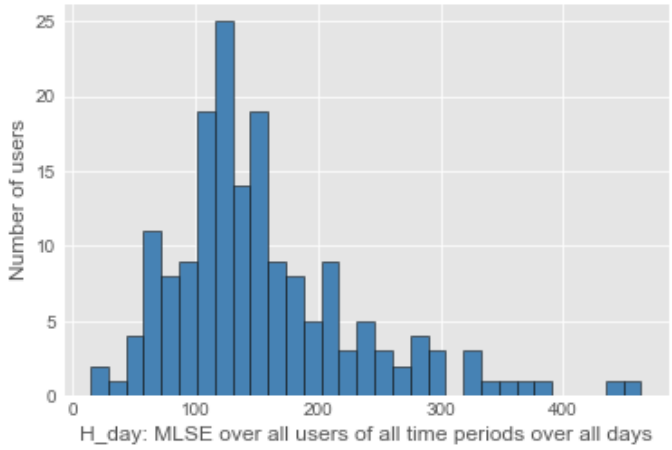

FIGURE 6: The H_{day} distribution over 91 days

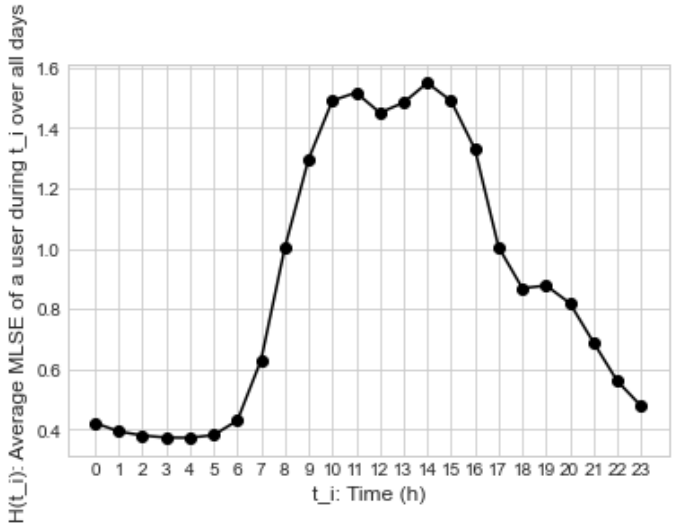

FIGURE 7: Average $H(t_i)$ of a user during t_i over 91 days

We can draw several conclusion from the above figures. It can be seen in Figure 6 that the daily MLSEs H_{day} are relatively small and most values are between 0 and 150. These

numbers reveal that most users do not switch their mobility patterns that often, since they are mostly at the same locations most days, for a given time t_i. Hence, their location sequences are not too complicated. Note that complicated here refers to the 'stability' of any pattern over the days. However, a small number of users has a high MLSE, up to 463, which reveals that they switch their pattern regularly, and thus have relatively complicated location sequences.

The shape of the graph in Figure 7 reveals that there are large differences between the stability of location patterns between certain time periods t_i. We see an increasing $H(t_i)$ starting from 6 am in the morning with a peak at 11 am and 2 pm to go down again to reach the lowest point at 3 and 4 am. This shows that during the peak hours 11 am and 2 pm many users visit different locations between different days. At night users are mainly at home to sleep, which implies a high stability of locations patterns between different days and thus a low $H(t_i)$.

4 Markov models

A Markov chain allows us to represent a user by a vector of location sequences over a given time period. These individual location sequences show the mobility behavior of the corresponding user. In a nutshell, a Markov chain is described by a conditional distribution function of visited locations given the visited locations of the user in the past. First, we have to introduce the concepts of Markov chains. A Markov chain can be defined on a discrete-time scale or a continuous-time scale. Also, we can use either finite discrete-time Markov chains or continuous-time Markov chains. To model the relationships of the locations in different time periods, we use a first-order Markov chain as mentioned in Equation (10). We elaborate on the previously mentioned choices in Chapters 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5 and 4.2.6. We first highlight some relevant definitions.

4.1 Preliminary definitions

Definition 4.1. *A finite discrete-time Markov chain is a sequence of random variables $X_1, X_2, X_3, \ldots$ with the Markov property which states that the probability of moving to the next state depends only on the present state and not on previous states:*

$$Pr(X_{t+1} = x_{t+1} | X_t = x_t, \ldots, X_1 = x_1) = Pr(X_{t+1} = x_{t+1} | X_t = x_t), \tag{10}$$

given that both conditional probabilities are well defined, that is

$$Pr(X_t = x_t, \ldots, X_1 = x_1) > 0. \tag{11}$$

The possible values of X_i form a finite set S called the state space of the chain. Since S is finite, the transition probability distribution can be represented by a matrix, called the transition matrix, with the (i, j)th element being equal to

$$p_{ij}(t) = Pr(X_{t+1} = j | X_t = i). \tag{12}$$

Since each row of this matrix sums to one and all elements are non-negative, it is a stochastic matrix.
Let

$$A = Pr(X_{t+1} | X_t) := (p_{ij}(t))_{i,j \in S}. \tag{13}$$

We can also distinguish time homogeneous and time non-homogeneous Markov chains. We compare both to see which one works best for our purpose.

Definition 4.2. *For time homogeneous Markov chains it holds that*

$$p_{ij}(t) = Pr(X_{t+1} = j | X_t = i) = Pr(X_t = j | X_{t-1} = i), \tag{14}$$

for all t.

Given a process, and after verifying the Markov property, one is interested in the steady state of the process, which leads to forming equations, called *balance equations* that have π as a solution.

Definition 4.3. *A probability measure π on set S is a stationary distribution for the Markov chain $(X_t)_{t \geq 0}$ if*

$$\pi(i) = \sum_j \pi(j) p_{ji} \ , i \in S. \tag{15}$$

In matrix notation, this equation can be written as

$$\pi = \pi \boldsymbol{A}, \tag{16}$$

implying that π is an eigenvector of the matrix A. Since π is a probability measure, the entries $\pi(i)$ are non-negative and sum to 1, that is:

$$\sum_i \pi(i) = 1. \tag{17}$$

Finally, this stationary distribution equals its limiting distribution

$$\lim_{t\to\infty} Pr(X_t = i),\ i \in S, \tag{18}$$

under certain conditions. Let p_{ij}^n denote the $(i,j)th$ entry of A^n (so $A^n = (p_{ij}^n)_{i,j\in S}$). Then by the definition of matrix multiplication,

$$p_{ij}^n = \sum_{i_1,\ldots,i_{n-1}\in S^{n-1}} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{n-1},j}. \tag{19}$$

The probability $Pr(X_n = j|X_0 = i)$ is the sum of the probabilities of all paths of the form $i, i_1, \ldots, i_{n-1}, j$, which is the sum in Equation (19). Consequently,

$$Pr(X_n = j|X_0 = i) = p_{ij}^n. \tag{20}$$

This probability can be obtained upon computing A^n.

Definition 4.4. *States i and j communicate with each other if state j is accessible from state i ($i \to j$) and state i is accessible from state j ($j \to i$). That is, $p_{ij}^n > 0$ and $p_{ji}^n > 0$ for some $n \geq 1$.*

Definition 4.5. *A Markov chain is **irreducible** if state i communicates with state $j (i \leftrightarrow j)$ for any $i, j \in S$, where S is a countable set of states.*

Definition 4.6. *The period d_i of a state i is the greatest common divisor (gcd) of all n that satisfy $p_{ii}^n > 0$. In other words, d_i is the largest integer such that $p_{ii}^n > 0$ if and only if n is a multiple of d_i. State i is **aperiodic** if $d_i = 1$.*

Theorem 4.7 (Theorem 59 of Chapter 1 in [24])**.** *If a Markov chain is ergodic, that is, irreducible and aperiodic, then its stationary distribution is its limiting distribution, which is positive:*

$$\lim_{t\to\infty} Pr(X_t = i) = \pi(i)\ ,i \in S. \tag{21}$$

4.2 Modeling choices

Now we have introduced the concept of a Markov model for a (group of) users, we can elaborate on some modelling choices, each of which gives different results. We believe it is reasonable that our problem is modelled by a Markov chain, since it was previously shown that user behavior, in the form of visited locations, exhibits strong periodic patterns [25]. Furthermore, we distinguish between a discrete-time scale and continuous-time scale (continuous Markov chains are explained in [24]), a time homogeneous and time non-homogeneous model, different temporal resolutions of the locations, different choices of locations for an assumed resolution and different location types in combination with days of the week. The choice between a time homogeneous and time non-homogeneous model is based on the goal of the research, while the other choices depend on the performance of the model.

4.2.1 Choice of the order

In this research we only focus on the first-order Markov model. In practice, the location of a user at period $t+1$ does often not only depend on the location at t, but also on the locations at $t-1$ and/or even $t-4$ etc. Consider the following example of a user who in a day goes to school to drop off kids, then goes to office, supermarket and finally arrives back home. However, if he has dropped off kids, sometimes he goes to the supermarket first, then office and then home. Other times, he goes to school, office, home, supermarket and finally home. In this example, if a Markov chain would be formed only based on the last location, it would be too strong of an assumption. If we would form a Markov chain on more locations, we would be able to have perhaps a higher accuracy up to a point. However, it is true that a user's current location probably is independent or extremely weakly dependent of the past one hundred locations or so. So at some point independence is not an absurd assumption, but maybe one location is too strong. Still, a first-order chain highlights the important patterns in the user's behavior, while it is also relatively easy to compute and perform computations on. Moreover, in an experiment it was found that in fact, the first-order Markov model outperforms the higher order Markov models that are often overfitted [22]. So, clearly the fact that we use first-order models neglects long term dependencies. But the choice was made to maintain simplicity of the model.

4.2.2 Choice of time scale

The first choice we make is whether we use a discrete-time scale or continuous-time scale. We choose the discrete-time scale, since (depending on the questions we want to answer) it has added value to know the exact times at which certain locations are visited. As an example, if we consider the question: 'is a certain user travelling during rush hours?', we want to distinguish the user's transition probabilities and stationary distributions at different hours in a day. In this thesis we consider the question 'does a user have a job?', where it can be useful as well to distinguish user's locations (such as 'office') during different hours.

4.2.3 Dependence on time

The easiest model is a time homogeneous discrete first-order Markov model in which the transition probabilities are constant over time as can be seen in Equation (14). In a time non-homogeneous Markov model the probabilities are not constant over time. This implies for example that for a user the probability to be at his office at $t = 8$, conditioned on being at home at $t = 7$ can be different from the probability of being at office at $t = 10$, conditioned on being at home at $t = 9$. The state space of these non-homogeneous models not only includes the locations, but also the notion of time, here each hour of the day. Thus, such a model can also answer questions predicting for example the user's location at $t = 17$ on a Friday. Going back to the questions we considered in Chapter 4.2.2, we see that in both cases it potentially has added value to include the time in the state space. If we consider the question: 'is a certain user travelling during rush hours?', we want to distinguish the user's transition probabilities and stationary distributions at different hours in a day. In this thesis we consider the question 'does a user have a job?', where it can be useful as well to distinguish user's locations (such as 'office') during different hours. However, in this thesis we mainly highlight the time homogeneous model to maintain simplicity of the model. Yet, it could also be that the time homogeneous model is the better classifier.

4.2.4 Temporal resolution of locations

The raw geographic mobility data of a user consists of all trips and stays at locations during each time period of the day. The time period is variable, that is, we can choose different temporal resolutions. The higher the temporal resolution, the less time in each time period in which only one visited location is chosen. In an ideal case, one chooses a resolution that is as high as possible so that the final performance of the model is as accurate as possible. However, even when taking an hourly resolution, there are hours in which no user's location was detected. This means that the higher the resolution, the more time periods we have with missing location's data. In this case we take the last known location, so we assume that a user does not move to another location if we have no new data. This is because it is difficult to predict to which other location the user potentially moved. So, we have to find a good balance between having a high resolution on the one hand and a small amount of missing data on the other hand. A high resolution also means that the rarer locations (that is, all locations except for home) are better represented in the data. In this case, we run the model for the resolutions of 15 minutes and 1 hour. The results will be presented in Chapter 8.1.2.

4.2.5 Choice of location in each time period

For the Markov model we can only choose a single location in each time period, also if in that period there are multiple location changes. There are multiple rules possible, think of: taking the first location visited in every hour, taking the second location (if there are at least two locations visited), taking the third location etc. Another choice is to take the location which the user visited for the longest amount of time in that time period. We also assume that if there is a non-home location in a time period, the home location is always ignored. This is done since mainly the non-home locations distinguish users from each other in their location sequences. As an example, we consider the mobility pattern of a panel user in Figure 4. For the intervaltime '1-11-2021 21:00', we see that the user visits three different locations: 'home', 'car' and 'office'. Considering the fact we can only choose a single location, we disregard the home location and let the choice for either 'car' or 'office' depend on if we take the location with the longest duration or the first location, since we evaluate the model for the choices: [Longest duration, 1st location]. At the intervaltime '1-11-2021 17:00' 'home' is the only location, so we do choose the home location in this case. The results will be presented in Chapter 8.1.2.

4.2.6 Location types and days of week choices

As mentioned in Chapter 3.1, we have many different locations next to the stay-locations 'stay@home' or 'stay@office'. Therefore it could be beneficial to the model's performance to only choose (a set of) locations that distinguish the users in the positive and negative classes best, subject to the specific demographic characteristic question that is asked. This can be done in combination with the days of the week that are considered in the model.

An option could be to try many combinations of location types and days of the week to see which set achieves the best performances. Better is to first write an algorithm that potentially gives an indication for the most discriminating set of locations for the specific binary question. In this algorithm, the trip summary statistics 'visit_count', 'duration_h' as represented in Figure 8 are obtained from the data set introduced in Chapter 3.1. For each individual user $i \in I_1 \cup I_2$, (I_1 and I_2 represent users from the two different training sets, users with job and without job respectively) we only consider the trips with home

as the overnight stay. Then, for each location j at day-type k (weekday or weekend) we know how often (= visit count, $vc_{(i,j,k)}$), how long (= duration, $dur_{(i,j,k)}$) and how many days (= day count, $dc_{(i,j,k)}$) they are visited for users $i_1 \in I_1$ and $i_2 \in I_2$. For the purpose of the Markov model, the location(s) with high difference values for the visit count per user and duration per user lead in the right direction. In Figure 8 we see the visit count $vc_{(1365,j,k)}$ and duration $dur_{(1365,j,k)}$) for user ID 1365 for five different destination types, namely 'any', 'office', 'residential', 'commercial' and 'house' on different day types 'any', 'weekday' and 'weekend':

user_id	day_type	stay_overnight_type	destination_type	visit_count	duration_h	duration	day_count
1365	any	home	any	119	198.477500	714519	74
1365	weekday	home	any	91	155.463333	559668	55
1365	weekend	home	any	28	43.014167	154851	19
1365	any	home	commercial	7	15.629167	56265	7
1365	weekday	home	commercial	6	15.003056	54011	6
1365	weekend	home	commercial	1	0.626111	2254	1
1365	any	home	house	38	145.898333	525234	30
1365	weekday	home	house	12	18.297222	65870	10
1365	weekend	home	house	2	8.628333	31062	2
1365	any	home	residential	5	1.817222	6542	5
1365	weekday	home	residential	3	1.231389	4433	3
1365	weekend	home	residential	2	0.585833	2109	2

FIGURE 8: Trip summary statistics on the visit count $vc_{(1365,j,k)}$ and duration $dur_{(1365,j,k)}$ for all j, k.

So, we decided to choose those individual location(s) j to be included in the state space S of the Markov chain at day-type k for which the users of the two different training sets I_1 and I_2 have the largest differences between the visit count per user and duration per user, where

$$vc_{(I_1,j,k)} = \sum_{i_1 \in I_1} vc_{(i_1,j,k)} \tag{22}$$

$$vc_{(I_2,j,k)} = \sum_{i_2 \in I_2} vc_{(i_2,j,k)}, \tag{23}$$

and

$$dur_{(I_1,j,k)} = \sum_{i_1 \in I_1} dur_{(i_1,j,k)} \tag{24}$$

$$dur_{(I_2,j,k)} = \sum_{i_2 \in I_2} dur_{(i_2,j,k)}, \tag{25}$$

$$(j^*, k^*) := \arg\max_{j,k}(|vc_{(I_1,j,k)} - vc_{(I_2,j,k)}|) \tag{26}$$

$$\arg\max_{j,k}(|dur_{(I_1,j,k)} - dur_{(I_2,j,k)}|). \tag{27}$$

This results in the location j^* 'any', which are all other locations than home (non-home locations), including k^* 'any' daytype in the week being the most discriminating. Also, 'weekdays' are more discriminating than 'weekend' days. Basically, this comes down to the fact that we use the mobility of a user between his home and outside of his home, no matter which exact locations, to determine if he has a job or not.

4.3 Markov model building

With the help of OpenStreetMap (OSM), each end location of a trip is tagged by the most likely functionality of that location (also called 'modality'), namely the stay- and trip modalities. The stay modalities are: 'stay@home', 'stay@office' and 'stay'. Here, 'stay' can be divided into different types of stays, for example: community centre, shop, house or supermarket. The trip modalities are 'foot','bike', 'car', 'public transport', 'plane' and 'other'. In the case of a time homogeneous Markov chain, let

$$X_t \in S, t \in 0, 1, \ldots, 23, S = \{Home, \ldots, School\}, \tag{28}$$

where X_t is a random variable that represents one of the possible 274 user locations at time period t, where the set of locations S is semantic and finite. In this chain, the user location at time period $t+1$ only depends on the user location at time period t, where t is identified with period $[t, t+1)$. So, when having an hourly resolution, $t = 6$ indicates the time period 6am−7am. Note that, unless stated otherwise, we assume t to represent a time period of an hour.

In the case of a time non-homogeneous Markov chain, the state space not only includes the location, but also the time, which leads to

$$X_t \in S, t \in 0, 1, \ldots, 23, S = \{(Home, 0), (Home, 1), \ldots, (School, 22), (School, 23)\}. \tag{29}$$

Let us next illustrate a user's mobility sequence, between 7 am - 10 am for 2 days at an hourly resolution in Figure 9. This sequence can be mapped to a diagram representing the corresponding three-state Markov chain as presented in Figure 10. Each number represents the probability of the Markov chain changing from one state to another state, with the direction indicated by the arrow. At the same time, the sequence can also be mapped to both a time non-homogeneous first order Markov chain in Figure 11 and a time non-homogeneous chain in Figure 13. Note that the user visited the 'home' location at two consecutive hours on the first day, at 7 am and 8 am, followed by the 'school' location at 9 am. On the second day the user visited the 'home' location at 7 am, followed by 'school' and 'office' at 8 am and 9 am respectively. This leads to a 0.333 transition probability from the 'home' location to the 'home' location and a 0.667 probability from the 'home' location to the 'school' location. In a similar way, given that the user is in school, the probability of him being at home in the next hour is $Pr(X_{t+1} = home | X_t = school) = 0.5$.

7:00	**8:00**	**9:00**
Home		School
Home	School	Office

FIGURE 9: Input: User's location sequences between 7 am-10 am for 2 days at an hourly resolution.

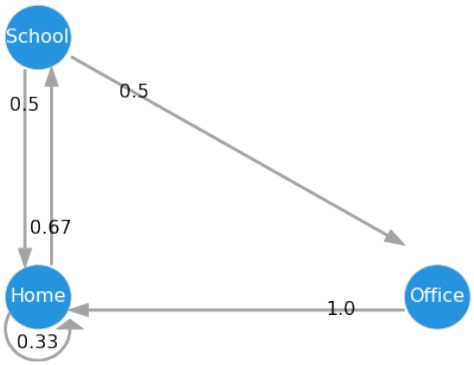

FIGURE 10: A diagram representing the three-state Markov chain as presented in Figure 13. Each number represents the probability of the Markov chain changing from one state to another state, with the direction indicated by the arrow.

Formally, we can refer to individual locations in the sequences with the following notation. This notation will be needed when we introduce the algorithm for setting up the transition matrices and stationary distributions in Algorithm 1. Let $seq\ u(i)$, with i the location index, be the location sequence of a user u. As an example, for the location sequences in Figure 9, $i = 3$ refers to the location at $t = 9$ during the first day, with location 'school', while after the transition $i + 1(= 4)$ refers to the location at $t = 7$ during the second day, with location 'home'. Consequently, $A_u[seq\ u(i), seq\ u(i+1)]$ denotes the number of transitions from location $seq\ u(i)$ to location $seq\ u(i+1)$ until index i in the sequence.

We also need the location sequences of a group of users, e.g. training group U_1 representing users with jobs, defined in Algorithm 1. Let $seq\ U_1(i)$, with i the location index, be the location sequence of the user set U_1. This implies that we want to obtain a single transition matrix A_{U_1} and single stationary distribution π_{U_1} for the location sequences of all the users in the user set U_1. As a result, A_{U_1} and π_{U_1} will be based on sequence $seq\ U_1$, which is u_1 times longer than the sequence of user u, $seq\ u$. As an example, if we now assume that the sequences in Figure 9 are not from the same user on two different days, but from two different users on the same day, $i = 3$ refers to the location at $t = 9$ of the first user, with location 'school'. After the transition $i + 1(= 4)$ refers to the location at $t = 7$ of the second user, with location 'home'. Consequently, $A_{U_1}[seq\ U_1(i), seq\ U_1(i+1)]$ denotes the number of transitions from location $seq\ U_1(i)$ to location $seq\ U_1(i+1)$ until index i in the sequence.

The rows in Figure 11 and Figure 13 represent the locations of the user at hour $t \in 7, 8, 9$, while the columns in the matrix represent the locations of the user at hour $t+1$. The matrix entries in transition matrices in these figures are the conditional probabilities according to Equation (10):

$$Pr(X_{t+1} = x_{t+1} | X_t = x_t). \tag{30}$$

Consequently, a user's u transition matrix is defined according to Equation (13) as A_u and a user's u stationary distribution according to Equation (16) as π_u.

		$\mathbf{X_{t+1}}$								
		(Home,7)	(Home,8)	(Home,9)	(Office,7)	(Office,8)	(Office,9)	(School,7)	(School,8)	(School,9)
$\mathbf{X_t}$	(Home,7)	0	0.50	0	0	0	0	0	0.50	0
	(Home,8)	0	0	0	0	0	0	0	0	1
	(Home,9)	0	0	1	0	0	0	0	0	0
	(Office,7)	0	0	0	1	0	0	0	0	0
	(Office,8)	0	0	0	0	1	0	0	0	0
	(Office,9)	1	0	0	0	0	0	0	0	0
	(School,7)	0	0	0	0	0	0	1	0	0
	(School,8)	0	0	0	0	0	1	0	0	0
	(School,9)	1	0	0	0	0	0	0	0	0

FIGURE 11: Output: Time non-homogeneous first order Markov chain of the sequences in Figure 9.

$\mathbf{X_t}$	$\pi(\mathbf{X_t})$
(Home,7)	0.333
(Home,8)	0.167
(Home,9)	0
(Office,7)	0
(Office,8)	0
(Office,9)	0.167
(School,7)	0
(School,8)	0.167
(School,9)	0.167

FIGURE 12: Output: stationary distribution of the Markov chain in Figure 11.

		$\mathbf{X_{t+1}}$		
		Home	Office	School
$\mathbf{X_t}$	Home	0.333	0	0.667
	Office	1	0	0
	School	0.5	0.5	0

FIGURE 13: Output one: time homogeneous first order Markov chain of the sequences in Figure 9.

$\mathbf{X_t}$	$\pi(\mathbf{X_t})$
Home	0.5
Office	0.167
School	0.333

FIGURE 14: Output two: stationary distribution of the Markov chain in Figure 13.

One can also use higher order Markov chains or even a variable order chain (VOBN) to model the mobility behavior of a user [26, 27].

Definition 4.8. *A Markov chain of order o, where o is finite, is a process satisfying:*

$$\begin{aligned} &Pr(X_t = x_t \mid X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \ldots, X_1 = x_1) \\ &= Pr(X_t = x_t \mid X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \ldots, X_{t-o} = x_{t-o}) \text{ for } t > o \end{aligned} \tag{31}$$

In other words, the future state depends on the past o states.

In order to be able to define the job classifier, we now need to define the Markov model for a user u, used to determine differences between users. For this we use that for two successive locations X_t and X_{t+1} it holds that $Pr(X_t, X_{t+1}) = Pr(X_t) \cdot Pr(X_{t+1}|X_t)$. Note that when we mention 'model' without the context of being used for calculating differences between a (set of users) we mean the complete model build by Algorithms 1 and 2 for which we can calculate its performance.

Definition 4.9. *The **Markov model for a user u**, B_u, is the joint probability of any two successive locations $X_t \in S$ and $X_{t+1} \in S$, where*

$$B_u(X_t, X_{t+1}) := Pr_u(X_t, X_{t+1}) = Pr_u(X_t) \cdot Pr_u(X_{t+1}|X_t) = \pi_u(X_t) \cdot A_u(X_{t+1}|X_t), \quad (32)$$

and $\pi_u \in [0,1]^{|S|}, A_u \in [0,1]^{|S|^2}, B_u \in [0,1]^{|S|^2}$.

This formula shows the joint probability of two successive locations $X_t \in S$ and $X_{t+1} \in S$ in the complete location sequence of a user u. For example, what is the chance at any time period t that the locations office and school happen after each other.

We now explain the algorithm that is used for determining the transition matrix A_u for a group of users user u, and stationary distribution π_u, shown in Algorithm 1. The data set described in Chapter 3.1 is split into training and testing sets and for the training set we split the data based on the ground truth, that is the job status of the users in this set. Per group that is trained, Markov chains are defined by computing the transition matrices per group after which the limiting distribution per group is computed. In this way, user profiles for different groups are built. Then, for each user in the test set, a Markov chain is defined where the same algorithm to compute its transition matrix is used as was done for a group of users. Finally, the joint probability of each two successive locations of each user, called the Markov model of a user as in Equation (32), is compared to the joint probability of each two successive locations of user profiles that are built.

Algorithm 1: The algorithm for setting up the transition matrices and stationary distributions to obtain the Markov model of (a set of) users

Input: Set of location sequences of u_1 training users of length $u_1 \cdot \mathbf{n}$ with a job, set of sequences of u_2 training users of length $u_2 \cdot \mathbf{n}$ without a job, $\mathbf{M}$ sequences of test users, each one with length of $\mathbf{n}$ and maximum of $\mathbf{L}$ unique locations taken together from all location sequences.

Output: Transition matrices A_{U_1}, A_{U_2} and A_u and stationary distributions π_{U_1}, π_{U_2} and $\pi_u, U_1 = \{1, \ldots, u_1\}, U_2 = \{u_1 + 1, \ldots, u_1 + u_2\}$ and $U = \{u_1 + u_2 + 1, \ldots, u_1 + u_2 + \mathbf{M}\}$ where U_1 is the set of users with jobs used for training, U_2 the set of users without jobs used for training and U the $\mathbf{M}$ test users, $u \in U$.

initialize an empty list of transition matrices and stationary distributions
for *user sets* U_1 *and* U_2 **do**
    initialize zero transition matrices A_{U_1} and A_{U_2}
    for *each index* i *in the location sequences of user sets* U_1 *and* U_2*: seq* $U_1(i)$ *and seq* $U_2(i)$ **do**
        add 1 to $A_{U_1}[seq\ U_1(i), seq\ U_1(i+1)]$
        add 1 to $A_{U_2}[seq\ U_2(i), seq\ U_2(i+1)]$
    end
    normalize the transition matrices
    solve $\pi_{U_1} = \pi_{U_1} A_{U_1}$ for π_{U_1}
    solve $\pi_{U_2} = \pi_{U_2} A_{U_2}$ for π_{U_2}, if there is no solution or unique solution π_{U_1} and/or π_{U_2}, see explanation below
    determine B_{U_1} as in Definition (32).
    determine B_{U_2}
end
for *each user* u **do**
    initialize a zero transition matrix A_u
    for *each index* i *in the location sequence of user* u*: seq* $u(i)$ **do**
        add 1 to $A_u[seq\ u(i), seq\ u(i+1)]$
    end
    normalize the transition matrix
    solve $\pi_u = \pi_u A_u$ for π_u
    determine B_u, many steps are explained more elaborately below
end

- One of the assumptions we make is that we include transitions from the end of a location sequence to the beginning of a new sequence. In the case of having a sequence of 24 hours this implies that there is a transition included from $t = 23$ to $t = 0$ on the next day as explained for the example in Figure 9. However, a modelling option could also be to include only the mobility behavior during a part of each day, for example each day from 7 am - 22 pm. In this case also a transition from 22 pm - 7 am is included with the reason that during the night hours often not many location changes take place, justifying a transition over more than one hour. Only including the mobility behavior during a part of each day/week in the model could be beneficial for the model performance. For the purpose of classifying whether a user has a job or not, one can come up with the suggestion to exclude location data from the weekends, since most people do not work in weekends and have irregular mobility patterns. Note that we did not test the modelling choice to exclude certain

hours in the day, but we tested the choice to exclude weekend days from the model in Chapter 4.2.6 with the results in Chapter 8.1.2.

As mentioned in the input of Algorithm 1, we take all the locations together from location sequences of both the training set and the test set as states in the transition matrices. These states are presented in the matrices in the same order. For example, assume that locations A, B, C, D appear in the training set, but for a specific user only locations A and B appear in the test set. However, the corresponding transition matrix of this user in the test set still includes the locations C and D. This is done so that when we want to calculate the differences between two Markov models, we have no problem of having different dimensions of the matrices. However, the consequence of this could be that we have a row in the transition matrix which consists of only zeroes. The function *scipy.stats.entropy* in Python, which determines the Kullback-Leibler (KL) divergence (introduced in Chapter 5) if we compare two distributions, is only able to calculate the entropy if both distributions can be normalized to sum to 1. We assume that if there is no transition from the a certain state to another one (also not to itself via a self-loop), the user remains in this state. We call this state an *absorbing state*.

Definition 4.10. *A set of states C in S is said to be closed if no state outside of C is accessible from any state in C. That is, $p_{ij} = 0$ for any $i \in C$, $j \notin C$. If $C = i$, which is a singleton set, is closed, then i is an **absorbing state**, that is $p_{ii} = 1$.*

Proceeding with the example we just introduced, locations C and D would be absorbing states for the user in the test set.
As mentioned in Equation (21), a stationary distribution equals its limiting distribution if the Markov chain is irreducible and aperiodic. The limiting distribution can here be seen as the relative frequencies in the locations in the long run. This means that for each irreducible (set of) states we have to calculate the transition matrix A_u and the stationary distribution π separately. So within a Markov chain we can have so-called 'islands' of states. We get 'islands' if certain locations can not be reached from every other location anymore, which happens for the absorbing locations C and D for the user in the test set.

- Normalizing the transition matrix means that the row values in A_u, which represent the number of transitions from each location to each other location, are divided by the sum of the row values. In this way the count of the location frequencies is changed to the transition probability matrix $Pr_u(X_{t+1}|X_t)$.

- Next to the Markov chain (in the form of the transition probability matrix A), we also determine the steady-state probability vector (stationary distribution), namely $\pi(X_t)$. Those are measured by the relative frequency of each location over the whole time period, see Equation (15) and Equation (16).

- See Definition 32. Multiply the stationary distribution $\pi_u(X_t)$ with the transition matrix $Pr_u(X_{t+1}|X_t)$ to get the Markov model B_u of user u.

4.4 Initial conditions

The initial distribution of a Markov chain is the probability distribution of the chain at time 0. For each state $i \in S$, we denote by $\pi_0(i)$ the probability $Pr(X_0 = i)$ that the

Markov chain starts out in state i. We suppose our Markov chain starts out with initial distribution $\pi_0(i) = \pi$. As we just mentioned, location patterns are strongly periodic [25]. In the case of a user with a job, most of the weeks he goes to the office five days a week for eight hours and has similar patterns between office and home. We only consider a period of three months in this thesis and there is a high chance that the location patterns are similar if we included even more weeks before the initial date. So, when we have that a Markov chain's stationary distribution equals its limiting distribution, it is reasonable to assume that the stationary distribution also equals the initial distribution. As an example, in the case of a time non-homogeneous Markov chain, it implies that the probability of being at office at 9 am ($Pr(\text{Office}, 9)$) is equal on all days during the period of three months. Then, by Equation (15), we also have $\pi_1 = \pi$, ..., $\pi_{23} = \pi$. That is, if the distribution at $t = 0$ is π, then the distribution at $t = 1$ is still π. Thus, $Pr(X_t = i) = \pi(i)\ \forall i$.

5 Assessing the 'differences' between users

The purpose of this section is to introduce the Jensen-Shannon (JS) distance (for which we need the Kullback-Leibler divergence) after which we justify the choice for the JS distance in Section 5.1. To do so, we refer to some examples to illustrate the use of the JS distance as given in Algorithm 2 for a time homogeneous Markov model, in comparison to the three other metrics, the Euclidean distance, Hamming distance and longest common subsequence (LCSS). In this example, in Figure 15, we show the mobility behavior of four users over one day during 6:00 - 22:00.

After setting up a stochastic Markov model for (a group of) individual users, the difference between those users can be represented as a distance between their Markov models as defined in Equation (32), which are probability distributions. The Kullback-Leibler (KL) divergence [28] D_{kl} is a measure which is suited to determine such difference. The KL divergence between two discrete probability distributions P and Q is defined as in Equation 33. Note that the double bar ($||$) between distributions P and Q emphasises that the order of the arguments matters. This reminder is helpful because KL is used much like a distance, but it is not symmetric, so it is not a distance. Therefore, it is also called a 'pseudo-distance'.

$$D_{kl}(P||Q) = \sum_{x \in X} P(x) log\left(\frac{P(x)}{Q(x)}\right), \tag{33}$$

where $P(x)$ is the probability of entry x under the distribution P and $Q(x)$ is the probability of entry x under the distribution Q. The KL divergence is also called the relative entropy, because it can be interpreted as the amount of information lost when P is used to approximate Q.
During the modelling phase π_u and A_u were already determined, once for all users of the same class in the training set and once for each individual user in the test set based on their location sequences as explained in Algorithm 1. The KL divergence between two joint probability distributions of discrete random variables X and Y, $P(X,Y)$ and $Q(X,Y)$, can be calculated as follows. The proof is based on the chain rule:

Theorem 5.1. *[28]*

$$D_{kl}[P(X,Y)||Q(X,Y)] = D_{kl}[P(X)||Q(X)] + D_{kl}[P(Y|X)||Q(Y|X)]$$

Proof.

$$\begin{aligned}
&D_{kl}[P(X,Y)||Q(X,Y)] = \sum_{x \in X} \sum_{y \in Y} P(x,y) log\left(\frac{P(x,y)}{Q(x,y)}\right) \\
&= \sum_{x \in X} \sum_{y \in Y} P(x)P(y|x) log\left(\frac{P(x)P(y|x)}{Q(x)Q(y|x)}\right) \\
&= \sum_{x \in X} \sum_{y \in Y} P(x)P(y|x) log\left(\frac{P(x)}{Q(x)}\right) + \sum_{x \in X} \sum_{y \in Y} P(x)P(y|x) log\left(\frac{P(y|x)}{Q(y|x)}\right) \\
&= \sum_{x \in X} P(x) log\left(\frac{P(x)}{Q(x)}\right) \sum_{y \in Y} P(y|x) + \sum_{x \in X} P(x) \sum_{y \in Y} P(y|x) log\left(\frac{P(y|x)}{Q(y|x)}\right) \\
&= \sum_{x \in X} P(x) log\left(\frac{P(x)}{Q(x)}\right) + \sum_{x \in X} P(x) \sum_{y \in Y} P(y|x) log\left(\frac{P(y|x)}{Q(y|x)}\right) \\
&= D_{kl}[P(X)||Q(X)] + D_{kl}[P(Y|X)||Q(Y|X)]
\end{aligned}$$

□

The KL divergence has a few disadvantages, namely that it is non-symmetric, that is $D_{kl}(P||Q) \neq D_{kl}(Q||P)$ and that it can take infinite values, which already happens if $Q(x) = 0$ for some state x. This is not desired for the results, since we cannot compare infinite values with each other. These disadvantages imply that the KL divergence is not a metric. To cope with these problems, we use a refinement of KL divergence, namely the Jensen-Shannon distance (JSD) between two joint probability distributions $P(X, Y)$ and $Q(X, Y)$ for the classification phase, which is the square-root of the Jensen-Shannon divergence [29]:

$$D_{jsd}(P(X,Y)||Q(X,Y)) = \sqrt{D_{js}(P(X,Y)||Q(X,Y))} \tag{34}$$

$$= \sqrt{\frac{1}{2}D_{kl}(P(X,Y)||M(X,Y)) + \frac{1}{2}D_{kl}(Q(X,Y)||P(X,Y))}, \tag{35}$$

where $M(X, Y)$ is defined as

$$M(X,Y) = \frac{1}{2}(P(X,Y) + Q(X,Y)). \tag{36}$$

We now give the algorithm used for calculating distances between the models of users as in Equation 32 for classifying a user.

Algorithm 2: The classification algorithm

Input: Transition matrices A_{U_1}, A_{U_2} and A_u and stationary distributions π_{U_1}, π_{U_2} and $\pi_u, U_1 = \{1, \ldots, u_1\}, U_2 = \{u_1 + 1, \ldots, u_1 + u_2\}$ and $U = \{u_1 + u_2 + 1, \ldots, u_1 + u_2 + \mathbf{M}\}$ where U_1 is the set of users with jobs used for training, U_2 the set of users without jobs used for training and U the $\mathbf{M}$ test users, $u \in U$.

Output: job classifications

for *u in range*$(u_1 + u_2 + 1, u_1 + u_2 + \boldsymbol{M})$ **do**
  $M_1 = \frac{1}{2}(A_{U_1} + A_u)$
  $M_2 = \frac{1}{2}(A_{U_2} + A_u)$ (i.e. add the corresponding matrix entries, and place this sum in the corresponding position in the matrix which results)
  $\pi_{M_1} = \frac{1}{2}(\pi_{U_1} + \pi_u)$
  $\pi_{M_2} = \frac{1}{2}(\pi_{U_2} + \pi_u)$
  for *row r out of* $\boldsymbol{k}$ **do**
    $D_{kl}(B_{U_1}, M_1) += \pi_{U_1}[r] \cdot D_{kl}(A_{U_1}[r], M_1[r])$
    $D_{kl}(B_u, M_1) += \pi_u[r] \cdot D_{kl}(A_u[r], M_1[r])$
    $D_{kl}(B_{U_2}, M_2) += \pi_{U_2}[r] \cdot D_{kl}(A_{U_2}[r], M_2[r])$
    $D_{kl}(B_u, M_2) += \pi_u[r] \cdot D_{kl}(A_u[r], M_2[r])$
  end
  $D_{kl}(B_{U_1}, M_1) += D_{kl}(\pi_{U_1}, \pi_{M_1})$
  $D_{kl}(B_u, M_1) += D_{kl}(\pi_u, \pi_{M_1})$
  $D_{kl}(B_{U_2}, M_2) += D_{kl}(\pi_{U_2}, \pi_{M_2})$
  $D_{kl}(B_u, M_2) += D_{kl}(\pi_u, \pi_{M_2})$
  $D_{jsd}(B_{U_1}, B_u) = D_{jsd}(B_u, B_{U_1}) := \sqrt{D_{js}(B_{U_1} || B_u)} := \sqrt{\frac{1}{2}(D_{kl}(B_{U_1}, M_1) + D_{kl}(B_u, M_1))}$
  $D_{jsd}(B_{U_2}, B_u) = D_{jsd}(B_u, B_{U_2}) := \sqrt{D_{js}(B_{U_2} || B_u)} := \sqrt{\frac{1}{2}(D_{kl}(B_{U_2}, M_2) + D_{kl}(B_u, M_2))}$
  if $D_{jsd}(B_{U_1}, B_u) < D_{jsd}(B_{U_2}, B_u)$ **then**
    user u with Markov model B_u classified having a job
  else
    user u with Markov model B_u classified having no job
  end
end

The first metric of comparison we consider is the *Euclidean distance*. This distance (in the context of location sequences) is measured between two location frequency vectors F_m and $F_{m'}$ based on location sequences of users u and u', $m := seq\ u$ and $m' := seq\ u'$ respectively, $m, m' \in (1, \ldots, M)$. Given M location sequences of length I, the elements of the frequency vectors $F_{m,s} \in \mathbb{R}^L$ are shown in Equation (37). Note that $i \in (1, \ldots, I)$ are the indices of the locations in the location sequences in each time period, and $s \in S$, where S is a finite set of L unique location names.

$$F_{m,s} = \frac{1}{L} \sum_{i=1}^{I} I(m_i = s), \tag{37}$$

Then we can define a location frequency vector based on a location sequence m, gathering all $F_{m,s}$.

$$F_m := (F_{m,s})_{s \in S} \tag{38}$$

Definition 5.2. *The Euclidean distance between two location frequency vectors F_m and $F_{m'}$ based on two location sequences m and m' of two different users (sets), is defined as:*

$$||F_m - F_{m'}||_2 = \sqrt{\sum_{s=1}^{L}(F_{m,s} - F_{m',s})^2}. \tag{39}$$

Secondly, we introduce the Hamming distance.

Definition 5.3. *The Hamming distance between two vectors of location sequences F_m and $F_{m'}$ of both length I, is the fraction of disagreeing locations in F_m and $F_{m'}$:*

$$\frac{\sum_{i=1}^{I} \mathbb{1}(F(i) \neq F'(i))}{I}, \tag{40}$$

where $i \in (1, \ldots, I)$ is again the index of the location in each time period in the location sequences.

The longest common subsequence (LCSS) is another possible distance metric. Let m and m' be two location sequences with size I and I' respectively, where $m = (m(1), m(2), \ldots, m(I))$ and $m' = (m'(1), m'(2), \ldots, m'(I'))$. For location sequence m, let $Head(m)$ be the sequence $Head(m) = (m(1), m(2), \ldots, m(I-1))$.

Definition 5.4. *The longest common subsequence between location sequences m and m': $LCSS(m, m')$ can be found by the following dynamic programming calculation:*

$$LCSS(m, m') = \begin{cases} 0 & \textbf{\textit{if}}\ m \textit{ or } m' \textit{ is empty} \\ 1 + LCSS(Head(m), Head(m')) & \textbf{\textit{if}}\ |I - I'| = 0\ \textbf{\textit{and}} \\ & |m(I) - m'(I')| = 0 \\ max(LCSS(m, Head(m')), LCSS(Head(m), m')) & \textit{otherwise} \end{cases}$$

If m or m' is empty, there are no similar locations between the sequences, so $LCSS(m, m') = 0$. If $|I - I'| = 0$ and $|m(I) - m(I')| = 0$, it means that both sequences have the same length and that the locations at the last index are the same, that is $I = I'$. This implies that we can already add 1 to the value of the LCSS of the sequences without the last index, that is $LCSS(m, m') = 1 + LCSS(Head(m), Head(m'))$. Finally, in all other cases the locations at the last index are not the same between the two sequences, thus we can consider one sequence m' without the last index and m with the last index and vice versa. Then the maximum of those two values is taken as the value for LCSS, that is $LCSS(m, m') = max(LCSS(m, Head(m')), LCSS(Head(m), m'))$.

Definition 5.5. *The similarity function S1 between two location sequences m and m' is:*

$$S1(m, m') = \frac{LCSS(m, m')}{min(I, I')}. \tag{41}$$

Given the similarity $S1$ between two location sequences, which is a rational number between 0 and 1, the distance between the sequences is:

$$D1(m, m') = 1 - S1(m, m'). \tag{42}$$

As an example, take the following two sequences of letters, where each letter for example represents the location at a different hour [30]. The LCSS is marked in blue:
AABBBCBAAA
AAABCAABBA
The LCSS similarity is $S1(A, B) = \frac{7}{10}$ and the LCSS distance $D1(A, B) = \frac{3}{10}$.
LCSS is used in other papers to find 'similar users', where each paper calculates it in a different way [31, 32, 33, 34, 35, 36].

5.1 Justification of using JS distance

In this section we justify why the Jensen-Shannon (JS) distance is probably better suited for our research than some other distance metrics.
To do so, we refer to an example to illustrate the use of the JS distance as given in Algorithm 2, in comparison to the three other metrics, the Euclidean distance, Hamming distance and longest common subsequence (LCSS). In this example, in Figure 15, we show the mobility behavior of four users over one day during 6:00 - 22:00.
We now define 'similarity' among users based on the Markov model of a user defined in Equation (32).

Definition 5.6. *We assume a high similarity among users when they have the following:*

1. *Share a similar frequency of locations, which results in a small distance between the stationary distributions of the training and test users.*

2. *Share similar location transitions in their location sequences, which results in a small distance between the transition probability matrices of the training and test users.*

Note that, for users to be considered similar, it is thus not needed to share the same locations at the same time periods. Specifically, this means that for example users 'train 1' and 'test 1' in Figure 15 have the exact same location sequence. This is achieved by shifting the location 'school' from $t = 7$ to $t = 8$ for user 'train 1' and increasing the time at home from only 1 hour at $t = 6$ to 2 hours at $t = 6$ and $t = 7$. This similarity condition is considered in most of the previous works.

We give one example testifying that the JS distance as defined in Equation 34 and Equation 36 is a good distance metric for classifying users. Distances between Markov models of users are determined by calculating the distances $d(A_{U_1}, A_u)$ and $d(A_{U_2}, A_u)$ for each $u \in [u_1+u_2+1, u_1+u_2+\mathbf{M}]$ as explained in Algorithm 2. Using a straightforward example with only one user in the training sets $U_1 = 1$ and $U_2 = 2$ is sufficient for the purpose of showing the differences in results between using different distance metrics. In this example, the rows 'train 1' and 'train 2' represent location patterns of the users in the training set of users with a job and without a job respectively during 6:00-22:00. The rows 'test 1' and 'test 2' represent a pattern of the users in the corresponding test set:

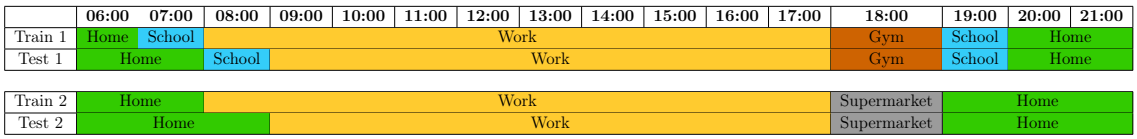

FIGURE 15: Example of four users' location sequences during 6:00-22:00.

Following the meaning of similarity in Definition 5.6, we would expect user 'test 1' to be closer to user 'train 1' than to 'train 2', because 'train 1' and 'test 1' both visit school before work (probably to bring their young children there) and they both go to the gym immediately after work. Likewise, we would expect user 'test 2' to be more similar to user 'train 2', because they do not visit school before work and go to the supermarket immediately after. The results of the distance calculations can be seen below. What strikes is that all metrics are symmetric.

Euclidean distance: original				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.088	0.198	0.250
Test 1		0	0.177	0.198
Train 2			0	0.088
Test 2				0

Hamming distance: original				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.118	0.176	0.235
Test 1		0	0.176	0.176
Train 2			0	0.059
Test 2				0

FIGURE 16: Distances between users 'train 1', 'test 1', 'train 2' and 'test 2' based on the the Euclidean distance (left) and Hamming distance metric (right) for the original example.

LCSS: original				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.059	0.176	0.235
Test 1		0	0.176	0.176
Train 2			0	0.059
Test 2				0

JS distance: original				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.064	0.522	0.535
Test 1		0	0.513	0.519
Train 2			0	0.052
Test 2				0

FIGURE 17: Distances between users 'train 1', 'test 1', 'train 2' and 'test 2' based on the the LCSS (left) and JS distance metric (right) for the original example.

We can conclude that in this example all four distance metrics work as we desire: user 'train 1' and user 'test 1' are closest to each other. The same goes for user 'train 2' and user 'test 2'. However, we can already observe the strength of the JS distance compared to the other metrics: it finds users 'train 1' and 'test 1' to be almost identical (the distance between them is very close to 0, namely 0.064), while 'test 1' has a relatively high distance to user 'train 2' (with distance 0.513). The same happens for users 'train 2' and 'test 2'.

According to the first assumption of our model, we want the classification methods to be able to define users as similar when they share a similar frequency of locations. It is not needed that they have location sequences that exactly match over time. Thus, we want to have a metric that has a low sensitivity to a time shift in visited locations. In this case, there is no change in the frequency of locations visited (assumption 1) and the patterns when moving between locations (assumption 2). In the following example, we shifted users' 'test 1' and 'test 2' locations forward by 2 hours, while keeping users' 'train 1' and 'train 2' mobility behavior identical to Figure 15:

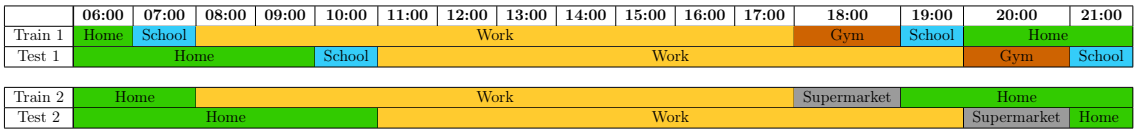

FIGURE 18: Example of four users' location sequences during 6:00-22:00 for the time-shifting example.

We expect that the JS distance works best here of the four distance metrics, by design of the Markov model (it is time homogeneous, thus time independent). The other distance metrics are not time independent. We want the distance metric to be *robust* over time, which means that the distance values between certain location sequences do not change (much) after a time-shift in the data. A distance metric which is robust over time still classifies user 'test 1' as being more similar to user 'train 1' than to user 'train 2' and user 'test 2' being more similar to 'train 2'. Figures 19 and 20 show the distance metrics for

the data with the time shift.

Euclidean distance: Time shift				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.088	0.198	0.250
Test 1		0	0.177	0.198
Train 2			0	0.088
Test 2				0

Hamming distance: Time shift				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.471	0.176	0.412
Test 1		0	0.412	0.176
Train 2			0	0.353
Test 2				0

FIGURE 19: Distances between users based on the the Euclidean distance (left) and Hamming distance metric (right) for the time shift example.

LCSS: Time shift				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.176	0.176	0.294
Test 1		0	0.294	0.176
Train 2			0	0.176
Test 2				0

JS distance: Time shift				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.064	0.522	0.535
Test 1		0	0.513	0.519
Train 2			0	0.052
Test 2				0

FIGURE 20: Distances between users based on the the LCSS (left) and JS distance metric (right) for the time shift example.

This example shows that the Euclidean and JS distance are less sensitive to time shifts of visited locations compared to the Hamming distance and LCSS. In other words, the Euclidean and JS distance are relatively robust over time. They still classify user 'test 1' to be most similar to user 'train 1' (Euclidean distance: $0.088 < 0.177$ and JS distance: $0.064 < 0.513$) and 'test 2' to be most similar to user 'train 2' (Euclidean distance: $0.088 < 0.250$ and JS distance: $0.052 < 0.535$). However, the Hamming distance now classifies user 'test 1' to be most similar to user 'train 2' ($0.412 < 0.471$), while 'test 2' is still most similar to user 'train 2' ($0.353 < 0.412$). The LCSS metric has similar classifications for the test users as the Euclidean and JS distance, but the distances between the corresponding training users are not close to 0 anymore (distance between 'train 1' and 'test 1': 0.176, between 'train 2' and 'test 2': 0.176).

Moreover, we want to have a metric that has a low sensitivity to the length of staying at the different locations, since the pattern when moving between locations is important. If, for example, we classified a user, which goes to the supermarket after work and then goes home, in a group where most users go to the supermarket after work and then they go home, we do not want that working a little less or shopping longer results in a change in classification. In the following example we decreased the work-time of users 'test 1' and 'test 2' by 2 hours:

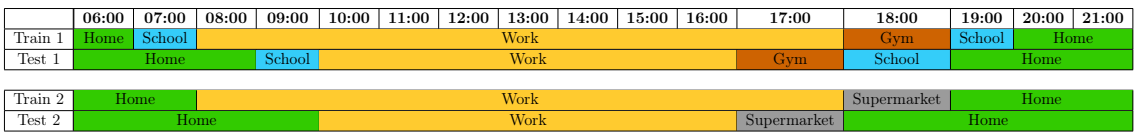

FIGURE 21: Example of four users' location sequences during 6:00-22:00 for the decreased work-time example.

A robust metric in the sense just explained should still classify user 'test 1' as being

more similar to user 'train 1' than to 'train 2' and user 'test 2' being more similar to 'train 2'. Figures 22 and 23 show the distance metrics as a result of the decreased work-time:

Euclidean distance: decr. work time				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.265	0.198	0.395
Test 1		0	0.250	0.198
Train 2			0	0.265
Test 2				0

Hamming distance: decr. work time				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.353	0.176	0.353
Test 1		0	0.235	0.176
Train 2			0	0.235
Test 2				0

FIGURE 22: Distances between users based on the the Euclidean distance (left) and Hamming distance metric (right) for the decreased work-time example.

LCSS: decr. work time				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.176	0.176	0.353
Test 1		0	0.235	0.176
Train 2			0	0.176
Test 2				0

JS Distance: decr. work time				
	Train 1	**Test 1**	**Train 2**	**Test 2**
Train 1	0	0.176	0.522	0.571
Test 1		0	0.522	0.516
Train 2			0	0.150
Test 2				0

FIGURE 23: Distances between users based on the the LCSS (left) and JS distance metric (right) for the decreased work-time example.

This example shows that the JS distance is indeed the most robust. It is less sensitive to a decreased work-time compared to the other distance metrics. The distance values in the corresponding distance matrix change, but only by a relatively small amount. The JS distance still classifies user 'test 1' to be most similar to user 'train 1' (JS distance: $0.176 < 0.522$) and 'test 2' to be most similar to user 'train 2' (JS distance: $0.150 < 0.571$). While the Euclidean distance still provided the desired classifications for the time shift example, it now classifies 'test 1' being closer to 'train 2' than to 'train 1' ($0.250 < 0.265$). We can summarize this finding in an 'observation'.

Observation: *When assessing the 'similarity' of vectors of semantic locations, examples suggest that the JS divergence between joint probability distributions of two successive locations in a Markov chain are more robust against time-shifts and manipulations of lengths of activities when compared to other, frequently used distance measures such as the Euclidean distance, Hamming distance and LCSS.*

6 LSTM

6.1 Standard Neural network

We start with shortly introducing standard neural networks. Neural networks are by now a standard and powerful tool for all kinds of classification problems. In a standard neural network, a set of input and output nodes are connected in which each connection has a weight associated with it. A neural network with only an input layer and an output layer is called a *perceptron* network or *cell*. Because real life problems do not fit into a perceptron network, the Multi-layer Perceptron (MLP) is more popular. The layers between the input and output layers are called hidden layers. An MLP cell is illustrated in Figure 24. Each arrow in this figure represents a connection, where each connection has a weight and a bias attached to it. In combination with the 'activation functions', in the three neurons in the hidden layer, they are used to calculate the output value. These activation functions can be any non-linear function $f(\cdot)$. The weight and bias are calculated using backpropagation. This is a neural network algorithm that iteratively processes a training dataset, where for each data point it compares the network's prediction with the known target value. The weights and the biases are updated accordingly, with the goal to minimize the mean-squared error.

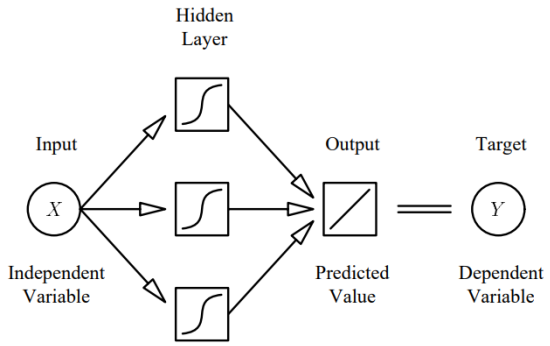

FIGURE 24: Representation of a feed-forward Multi-layer Perceptron cell [37].

6.2 An overview of Recurrent Neural Networks (RNN)

As we saw in Chapter 6.1, in a standard neural network, an output at a certain time step t is not used as an input for the next time step $t+1$. However, in reality their are many situations in which the final output not only depends on the external inputs, but also on output of previous time steps. Consider a person who reads a book. If he wants to understand each sentence, this understanding does not only depend on the words in the current sentence, but also on the understanding and the context created by previous sentence(s). Humans use this knowledge and context to understand something. Recurrent Neural Networks (RNN) are invented to address this limitation. They have feedback connections, which means that RNNs do not only take single external input into account (such as single locations) but also earlier outputs in form of sequences of data from the recent past (such as location sequences). The figure below shows such a RNN with a feedback connection, which is as shorthand notation for the network on the right, after unrolling the network on the left. Here A is a part of the RNN, with input x_t and output h_t, $t \in \mathbb{Z}^+$:

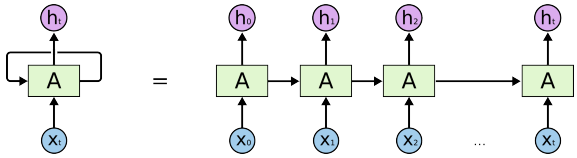

FIGURE 25: Representation of a RNN in a chain of repeating cells A [38], where the loop on the left is unrolled.

A is just a map $f(\cdot)$ which is usually an 'activation function'. We see that, at time step t for some input x_t, the RNN generates an output h_t. In the next time step $t+1$, the RNN has two external inputs X_{t+1} and h_t to produce the output h_{t+1}. Thus, the feedback loop enables information to be stored in cells and thus sent through the whole network.
However, RNNs also have their limitations. RNNs work well when the 'context' is from the recent past. When the 'context' is from the distant past, they work not as good [39]. In theory, RNNs are capable of handling such 'long-term dependencies' (also by repeating context from the recent past long enough). The right parameters can be chosen to solve this problem. However, in practice, RNNs do not seem to be able to learn them.

6.3 LSTM networks

As we saw in Figure 25, an RNN can be represented in the form of a chain of repeating cells of the RNN. As shown in Figure In standard RNNs, this cell A has a straightforward structure, for example a single *tanh* layer between the input x_t and output h_t. The *tanh* layer possesses the corresponding *tanh* activation function:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{43}$$

where e is a mathematical constant that is the base of the natural logarithm.
The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

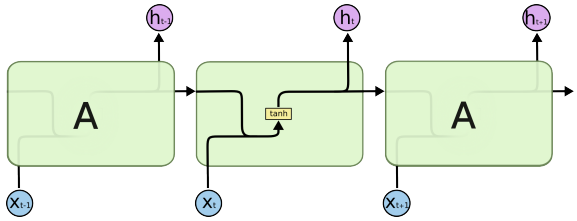

FIGURE 26: The repeating cell A in a standard RNN with a single *tanh* layer.

The chain of LSTMs have the same structure, but the repeating cell is different. Instead of the single neural network layer, LSTMs have four layers, which connect in a complex way as shown in Figure 27. Every connection transports a vector, from the output of one node to the input of another as shown in Figure 28. The pink circles are pointwise operations, such as vector multiplication. The yellow rectangles are learned neural network layers in which activation function are located, such as the *tanh* fucntion. If connections come together, the corresponding vectors concatenate. If a connection branches from a different connection, the corresponding vectors are copied.

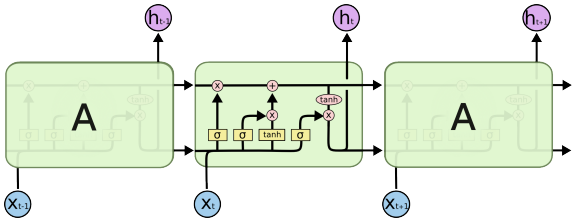

FIGURE 27: The repeating cells A in an LSTM with four layers.

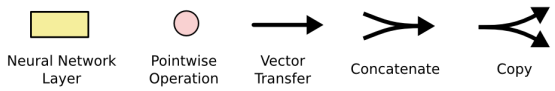

FIGURE 28: Notation in LSTM diagrams.

6.4 The formal working of LSTMs

To understand the working of LSTMs, we take a closer look at one the cells A, shown in Figure 29.

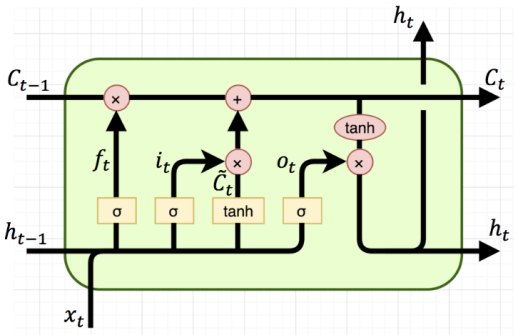

FIGURE 29: Single cell A in an LSTM.

An LSTM unit is build up out of a *cell state*, an *input gate*, an *output gate* and a *forget gate*.

In the equations below, the lowercase variables represent vectors. Matrices W_q and U_q contain, respectively, the weights of the input and recurrent connections, where the subscript q can either be the input gate i, output gate o, the forget gate f or the cell state C, depending on the activation being calculated. In this section, we are thus using a *vector notation*. So, for example, $C_t \in \mathbb{R}^h$ is not just one unit of one LSTM cell, but contains h LSTM cell's units.

Let us define the following variables, where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit.
- $f_t \in (0,1)^h$: forget gate's activation vector.
- $i_t \in (0,1)^h$: input/update gate's activation vector.

- $o_t \in (0,1)^h$: output gate's activation vector.
- $h_t \in (-1,1)^h$: hidden state vector also known as output vector of the LSTM unit.
- $\tilde{C}_t \in (-1,1)^h$: cell input activation vector.
- $C_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training.

The most important elements are the cell states C_{t-1} and C_t, which are connected by the horizontal connection going through the top of the cell in Figure 29. LSTM can add or remove information to this state, which is controlled by *gates*. A gate determines whether information is let through or not. The gates consist of a *sigmoid* layer σ and a pointwise multiplication operation $\times$. The *sigmoid* layer possesses the corresponding σ activation function:

$$\sigma(x) = \frac{1.0}{1.0 + e^{-x}} \tag{44}$$

which takes any real value as input and outputs values in the range 0 to 1.

This describes how much of the information should be let through to eventually the new cell state. A value of 0 means that nothing is going through, while 1 means that everything is let through. An LSTM has three gates, as can be seen in Figure 29.
The first step in the LSTM is to determine what information is removed from the cell state. It consists of the sigmoid layer σ and is named 'forget gate layer'. It takes h_{t-1} and x_t as an input, and outputs a value between 0 and 1 for each value in the cell state C_{t-1}. The subscript t indexes the time step.

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f). \tag{45}$$

In the next step it is determined what new information is stored in the cell state. This step consist of two parts. First, we have a sigmoid layer σ with the name 'input gate layer'. This layer decides which state values are updated. Afterwards, a *tanh* layer generates a vector of new candidate values, $\tilde{C}_t$, that is added to the cell state C_{t-1}. Formally:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \tag{46}$$

$$\tilde{C}_t = tanh(W_C \cdot x_t + U_C \cdot h_{t-1} + b_C) \tag{47}$$

The next step is to combine i_t and $\tilde{C}_t$ to form an update to the cell state C_{t-1}. So, the old cell state C_{t-1} is updated into the new cell state C_t.
The old state C_{t-1} is multiplied by f_t and then add the product of i_t and $\tilde{C}_t$, where $i_t \cdot \tilde{C}_t$ is the new vector of candidate values, scaled by i_t which decides how much each state value is updated. The initial value is $C_0 = 0$ and the operator $\odot$ denotes the Hadamard product (element-wise product).

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t. \tag{48}$$

As a last step, the output h_t has to be determined, which is a filtered version of the cell state C_t. First, we again have a sigmoid layer σ which decides what part of the cell state C_t go into the output h_t. Secondly, the cell state is led through a *tanh* layer and it is

multiplied by the output of the sigmoid layer, so that we only have an output of the parts we decided to. The initial value is $h_0 = 0$.

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{49}$$

$$h_t = o_t \odot tanh(C_t) \tag{50}$$

7 Performance metrics to compare classifiers

In the previous chapters we discussed two different classifiers, namely the Markov model and the neural network. Now we want to compare the two classifiers. In other words, we want to draw conclusions on the performance of both models to evaluate which one performs best. For this we need to use performance metrics, which we discuss in this chapter.

7.1 Choosing the right performance metrics

A confusion matrix can be used to visualize the performance of a model. We simply count True Positive (TP), the correctly predicted positive class data points of the model, False Positive (FP), the incorrectly predicted positive class data points, False Negative (FN), the incorrectly predicted negative class data points and True Negative (TN), the correctly predicted negative class data points. Since in this research we consider binary classification, the confusion matrix is two-dimensional [40]:

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

TABLE 1: Set-up of a confusion matrix.

Based on the values in this confusion matrix, one can calculate different performance metrics of the classification models. The most common used metrics are the accuracy, precision, recall and F1-score [41]. The accuracy is the fraction of the number of correctly predicted data points over the total number of predicted data points. Mathematically,

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}. \tag{51}$$

The precision is the fraction of the number of correct positive predictions made over the total number of positive predictions made by the model. Therefore, it is also called the positive predictive value (PPV). In other words, it calculates the accuracy of the TP:

$$Precision = \frac{TP}{TP + FP}. \tag{52}$$

We also have a negative variant of this metric, namely the specificity. It measures the fraction of the number of correct negative predictions over the total number of negative predictions. This is also known as the true negative rate (TNR):

$$Specificity = \frac{TN}{TN + FP}. \tag{53}$$

The recall (also called sensitivity) also uses the number of correct positive predictions (TP), but now it is divided by the total number of positive predictions that are made by the model. It is also referred to as the true positive rate (TPR):

$$Recall = \frac{TP}{TP + FN}. \tag{54}$$

7.2 Dealing with imbalanced data

Consider a binary classification problem to classify data points (in this thesis the data points are users) into 'positive' and 'negative' class labels. In real-world problems, there is often an imbalance of data across the different classes in a training set. In the context of a binary classification problem this means that the data points are unevenly distributed over the classes, i.e. one class label (positive or negative one) has a very high number of data points and the other has a very low number of data points. Accuracy, as defined in Equation (51), is the most used performance metric for classification models. However, when the positive class constitutes 90% of the data, and the negative class 10%, we can make a simple model that classifies all new data points in the test set (and thus not used for training the classifier) as being positive. This results in an accuracy of 90%, suggesting the model performs very good. However, this classifier is not good, since it only correctly predicts data points belonging to the positive class and incorrectly predicts all the data points belonging to the negative class. Depending on the purpose of the classifier, it can be just as important to correctly predict the negative data points, which is also the case in this thesis. The explanation for this will follow later in this section when we introduce different metrics. So, we have to find ways to deal with imbalance that are in line with the research goals.

There are different ways to deal with imbalanced data. One common way of solving this is to *restore* the balance in the data by oversampling or undersampling [42]. Restoring the number of data points from both classes means obtaining an equal number of data points from both classes. Oversampling means duplicating data points from the minority class, undersampling removing data points from the majority class. Instead of just copying existing data points in the minority class, it is maybe better to create new realistic data points to add to the minority class. This approach is called the Synthetic Minority Oversampling Technique (SMOTE), where a new data point is made based on the k nearest neighbours of an existing data point [43].

Another way of dealing with class imbalance, is to use different performance metrics than the accuracy. Ultimately, we choose one metric, which we use in evaluating both models. First of all, using both the precision and recall, we can determine the so-called F1-score, which equals the harmonic mean of both. This metric is often used when the data is imbalanced, that is the class distribution is uneven:

$$F1 - \text{score} = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{55}$$

Another metric is the balanced accuracy, which is the mean of the recall and the specificity:

$$Balanced\ Accuracy = \frac{Recall + Specificity}{2} = \frac{TP}{2 \cdot (TP + FN)} + \frac{TN}{2 \cdot (TN + FP)}. \tag{56}$$

The choice of the metric depends on the context of the research. We now compare some of the metrics and mention which metric can be used best in which situations.
First of all, we compare the balanced accuracy versus the normal accuracy. In case of imbalanced data, the balanced accuracy is a better metric, since it accounts for the class imbalance by giving the same weight to both classes [44]. In case of a balanced data-set, it is not better to use either of the two.
Another interesting comparison is between balanced accuracy and F1-score. The F1-score

does not take the number of classified TNs into account. Thus, in cases where the negative predictions are at least as important as the positives, balanced accuracy is a better metric than the F1-score.
Since we have imbalanced data and it is just as important to correctly predict a user with a job, than a user without a job, we use the balanced accuracy in this thesis. Another reason is that it is easy interpretable. Note that there are many more metrics (which maybe are even better), but analyzing this has no priority in this thesis.

8 Computational results based on locations data

In this section we mention how we build both the Markov model and the LSTM. Moreover, we show the parameter tuning we perform for both models.

8.1 Model building

We use *stratified group k-fold cross validation* [5], where the set of users with their corresponding location sequences are divided into k groups, or *folds* of approximately equal size. Thus, folds are subsets of the whole data set. The first fold is treated as a test set, and the method is fit on the remaining k - 1 folds. The working of stratified group k-fold cross validation with $k = 3$ is illustrated in Figure 30. It returns *stratified* folds with non-overlapping groups, which means that the distribution of classes in each split is preserved and each group is kept within a single split. In Figure 30 it is shown that the percentages of samples for each class are similar between the folds. Moreover, it does not happen that a group, in a single CV iteration, is both in the testing set and the training set. The fact that we use stratified folds is useful for our dataset, since it is unbalanced as explained in Chapter 7.2. Using just a group k-fold might produce skewed splits, that is splits with different percentages of samples for each class. Each group will appear exactly once in the test set across all folds. For this thesis, the classes represent users with or without a job and one group is one user with corresponding location sequence.

8.1.1 Choice of k in stratified group k-fold cross validation

We use $k = 3$. In other words, approximately 66.67% of the users are in the training set and 33.33% in the test set. There is no correct value of k. Clearly the fact that we use the low value of $k = 3$ has as a disadvantage that the model is presented with few data to train on, which could lead to underfitting of the model. But the choice was made because running the model would require way more time for higher values of k, since it has to train and validate k separate times. On the other hand, when using a higher k it could also be the case the we overfit the model, occurring when the model has been overtrained. We will show in Chapter 8.1.2 that when using $k = 3$, the performances in different iterations (thus for different compositions of the folds) do not differ much, thus justifying the choice for $k = 3$.

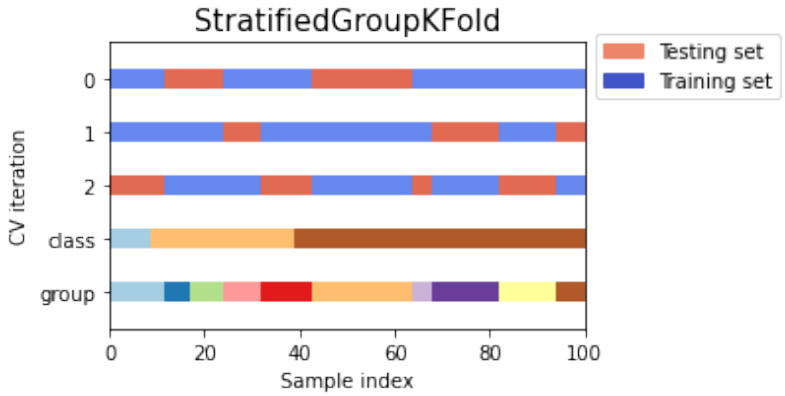

FIGURE 30: Visualization of stratified group 3-fold cross validation behavior for uneven groups [5].

The exact number of users (of in total 171 users) for each iteration in each fold is shown in the table below:

Number of users	**Users with job**	**Users without job**
Training set 1	85	28
Test set 1	43	15
Total set 1	128	43
Training set 2	85	29
Test set 2	43	14
Total set 2	128	43
Training set 3	86	29
Test set 3	42	14
Total set 3	128	43

TABLE 2: The number of users for each iteration in each of the three folds.

In this thesis the performance of the model is determined by taking the mean of the sum of the separate performances in each of the k iterations. However, there are also other ways to measure the performance, depending on the corresponding metric, computing the 'global' performance on the concatenation of the predictions for all k folds being an example. Since the balanced accuracy is used as performance metric as discussed in Chapter 7, the mean balanced accuracy will not differ much from the 'global' balanced accuracy. Moreover, as mentioned before, we do not only want to measure the performance but also get an idea of the differences between the performances across the folds, in order to detect instability and justifying the choice of $k = 3$. This cannot be done from the concatenation of the predictions, so in our case it is more convenient to keep the folds results separated.

8.1.2 Performances of the Markov model

In Table 3 we show the performances of the Markov model as introduced in Algorithms 1 and 2 for the different above mentioned modeling choices:

Modeling choices	**Temporal resolution**	**Hourly location**	**Location types + Weekday/end**	**Accuracy**	**Balanced Accuracy (BA)**
1	1 hour	1st location	All	0.620	0.716
2	1 hour	1st location	{Home, non-home} + weekdays	0.737	0.786
3	1 hour	Longest duration	All	0.619	0.723
4	1 hour	Longest duration	{Home, non-home} + weekdays	0.719	0.767
5	15 min	1st location	All	0.614	0.720
6	15 min	1st location	{Home, non-home} + weekdays	0.684	0.728
7	15 min	Longest duration	All	0.614	0.712
8	15 min	Longest duration	{Home, non-home} + weekdays	0.684	0.728

TABLE 3: Performances of the Markov model in the form of the accuracy and the balanced accuracy BA.

The confusion matrix for the default and the best option, that is modeling choices 1 and 2, is:

		Predicted	
		Job	No job
Actual	Job	67	61
	No job	4	39

		Predicted	
		Job	No job
Actual	Job	88	40
	No job	5	38

TABLE 4: Confusion matrix of modeling choices 1 and 2.

Below we summarize the results of the results of the Markov modeling choices:

Hyperparameter	**Default value**	**Examined options**	**Best option**
Temporal resolution	1 hour	[15 min, 1 hour]	1 hour
Hourly location	1st	[1st, Longest duration]	1st
Location types + Weekday/weekend	All locations + days	[All locations+{weekdays, weekend}, {Home, non-home}+{weekdays}]	{Home, non-home}+{weekdays}

Observation 1: *The balanced accuracy (BA) is always higher than the accuracy.*
This is also the case for modeling choice 2 ($0.786 > 0.737$). An intuition for this can be given with the help of the formulas for the accuracy and BA in Equations (51) and (56), respectively. As we explained, the BA is the mean of the recall in Equation (54) and specificity in Equation (53). Mainly the high specificity of 0.884 has a high contribution in the high value of BA, which implies that we have a high true negative rate. This rate is not incorporated into the accuracy.

Observation 2: *As can be seen in Table 3, choosing only a set of location(s) in combination with a selection of days that distinguish the users in the positive and negative classes well (here we take* $\{Home, non-home\}+\{weekdays\}$*) does improve the balanced accuracy for all relevant modeling choices 2 (0.786>0.716), 4 (0.767>0.723), 6 (0.728>0.720) and 8 (0.728>0.712).*
This shows that the algorithm that potentially gives an indication for the most discriminating set of locations as described in Chapter 4.2.6 has the desired effect.

Observation 3: *All performances (in the form of the BA) are quite similar (difference between highest and lowest BA is only* 0.074*), so there is no clear winner.*
There is no clear winner, since on a different data-set with the same classification question the result could be the other way around.

Observation 4: *The performances between modeling choices for a temporal resolution of 1 hour with different hourly locations and location types of home and non-home locations (choices 2 and 4 with BA values* 0.786 *and* 0.767*), differ more than the performances between the same modeling choices but for a resolution of 15 minutes (choices 6 and 8 with the same values* 0.728*).*
From this we can conclude that a higher resolution has the desired effect of having more similar performances (that is, more robust) for different choices of the hourly location, while at the same time we can not conclude from this that the performances are better as well.

Observation 5: *The number of 'False negatives (FN)' in Table 4 (61 out of 128 and 40 out of 128) is relatively high compared to the number of 'False positives' (4 out of 43 and 5 out of 43). There are many cases in which the model predicts a user to be jobless, while in reality he does have a job.*
To explain this we have a look at the Markov model of such a user and compare it to the

trained job model and trained jobless model. The case where we only consider the locations 'Home' and 'Non-home' during weekdays resulted in the following transition matrices P_1 of the trained job model, P_2 of the trained jobless model and stationary distributions π_1 of the trained job model and π_2 of the trained jobless model:

		X_{t+1}	
		Non-home	Home
X_t	Non-home	0.839	0.161
	Home	0.088	0.912

FIGURE 31: Transition matrix $P_1(X_{t+1}|X_t)$ for user with job.

X_t	$\pi_1(X_t)$
Non-home	0.355
Home	0.645

FIGURE 32: Stationary distribution π_1 for user with job.

		X_{t+1}	
		Non-home	Home
X_t	Non-home	0.720	0.280
	Home	0.078	0.922

FIGURE 33: Transition matrix $P_2(X_{t+1}|X_t)$ for user without job.

X_t	$\pi_2(X_t)$
Non-home	0.218
Home	0.782

FIGURE 34: Stationary distribution π_2 for user without job.

In line with what one would expect, a user without a job is in the long run more at home (π_2(Home) = 0.782) than out of home (π_2(Non-home) = 0.218). There are a few differences between both models. When in a non-home location at time t, the chances are relatively higher to be out of home at time $t+1$ as well for job users compared to jobless users. Mathematically,

$$P_1(X_{t+1} = \text{Non-home}|X_t = \text{Non-home}) > P_2(X_{t+1} = \text{Non-home}|X_t = \text{Non-home}). \quad (57)$$

The transitions with 'home' as start-location are similar between both groups of users. There is also a clear difference in the stationary distributions of both groups, which reveals that there is a higher chance being out of home for job users compared to jobless users. Mathematically,

$$\pi_1(X_t = \text{Non-home}) > \pi_2(X_t = \text{Non-home}). \quad (58)$$

We now look at the model of a user which is predicted to be jobless, while in reality she does have a job.

		X_{t+1}	
		Non-home	Home
X_t	Non-home	0.633	0.367
	Home	0.086	0.914

FIGURE 35: Transition matrix $P_3(X_{t+1}|X_t)$.

X_t	$\pi_3(X_t)$
Non-home	0.190
Home	0.810

FIGURE 36: Stationary distribution π_3.

The Jensen-Shannon distance as in Equation (34) implies that, roughly speaking, two models are seen as more similar if the differences between similar matrix-entries are smaller. In this case model 3 as in Figures 35 and 36 is most similar to model 2 in Figures 33 and 34. So test user 3 will be predicted to have the classification in line with model 2 (jobless), while in reality she does have a job. For this user the main reason for this misclassification is that she works a lot from home, possibly due to the ongoing Covid situation during November 2021 - January 2022.

It can also happen that a user is predicted to have a job, while in reality she is jobless. However we see in Table 4, that this occurs less often than a user which is predicted to be jobless, while having a job. As already mentioned in Chapter 3.2, this is most likely due to the fact that they perform voluntary work. As an example, they drive the bus voluntarily 5 days in the week, which means visiting many non-home locations leading to the incorrect prediction of having a job.

8.2 LSTM building and parameter tuning

Just as for the Markov model, in this model we have for each test user a semantic location sequence over 3 months as input, from which we gain a classification for this user with or without a job. We obtain a classification performance in the form of the balanced accuracy. We again use k-fold cross validation with $k = 3$, as explained in Section 4. We use the class *sklearn.model_selection.GridSearchCV* in the scikit-learn toolkit for this. Note that all locations in a sequence are replaced by integers that represent the ordered frequency of each location in the dataset. So, for example the locations sequence $AABA$ would result in the corresponding integer sequence 1121. The sequences of locations are therefore replaced by a sequence of integers. The ordered frequency is calculated by the number of times each location occurs in the dataset, where we still have the restriction that we can only choose a single location in each time period (as time period we choose 1 hour). Just like for the Markov chain model, we choose to use the 1st location in each period of an hour.

8.2.1 Word embedding

We use a word embedding for each index in the sequence of positive integers, before we let the sequence go through the LSTM network. Positive integer representations of words (or locations in this thesis) are encoded as real-valued vectors. Word embeddings are a type of word representation that allows words with similar meaning to have a similar vector representation. Embedding (in the context of locations) can be interpreted as follows: the

more often certain location patterns occur for the same type of users, the closer these locations are in the vector space. *Keras* has an embedding layer class *Embedding* in their library [45]. Keras is an open-source neural-network library written in Python. This function converts the positive integer representations of words into a word embedding. We map each word onto a real valued vector with length v, where we use $v = 32$. Again, the model performance potentially depends on the value of v. However, we only evaluated the model with $v = 32$, due to the high running time of the model. As explained before, all sequences of individual users are of the same length. In this way it is not needed to constrain each location sequence to consist of a certain amount of words.

8.2.2 LSTM building

We have the following simplified plot of the LSTM network:

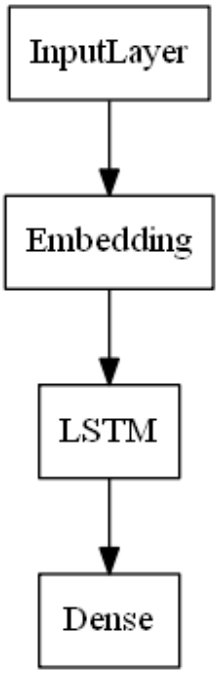

FIGURE 37: Visualisation of the used LSTM network with four different layers: the input layer, embedding layer, LSTM layer and dense layer for the output.

In our model, the embeddings are part of the LSTM network model, as can be seen in the second layer in Figure 37. As in the Markov model, we use an input length of the locations in the sequences of $\mathbf{n}$, which is equal for all individual users. The embedding vectors with length $v = 32$ are initialized randomly and are trained to maximize the model performance, that is minimizing the loss function during the training phase. As a loss function, we use the *binary cross-entropy loss*, which means that we obtain probabilities for both labels as our output.
The binary cross-entropy of the distribution Q relative to a distribution P over a given set is defined as follows:

$$H(P, Q) = -\sum_{x \in X} P(x) log Q(x), \tag{59}$$

which is very similar to the Kullback-Leibler divergence as defined in Equation (33).
Then, the label with the highest probability is chosen. After creating all the individual

embedding vectors, we concatenate (that is, joined together) them to form a vector of length $n \times v$. As a next step, the vector arrives at the LSTM layer(s) and a dense layer, used to obtain the binary classifications as output. In this way, the embeddings are learned while the LSTM network is optimized, leading to optimal embeddings for our specific problem with corresponding dataset.

8.2.3 Tuning settings: batch size and number of epochs

Compared to the Markov model, the LSTM neural network is more complicated and there are many hyper-parameters that can be tuned. In this section we focus on the so-called batch size and the number of epochs. The batch size is the number of data points processed before the model is updated. The number of epochs is the number of times that the training data-set is presented to the network during the training phase. An LSTM neural network is sensitive to both the batch size and number of epochs, which means that there is a combination that 'optimizes' the training of the network. The default for the batch size used in Keras is 32. There is no default value for the number of epochs. However, since a high number of epochs can lead to what is known as overfitting, we use 20 as default. We evaluate the model for different batch sizes: [16, 32, 64]. We use as the number of epochs: [20, 40, 80].

8.2.4 Tuning settings: other tuning settings (including number of neurons in the hidden layer)

Keras has many different state-of-the-art optimization algorithms. We do not test the network for different algorithms. The stochastic gradient descent optimizer 'ADAM' is based on adaptive estimation of first-order and second-order moments and is the best method to use for big data-sets [46]. We use a sigmoid neuron activation function in the dense output layer, because we have a binary classification problem. Note that we used a 'baseline' LSTM network as represented in Figure 37. It is possible to extend this baseline model by for example a Dropout layer between the Embedding and LSTM layer and/or the LSTM and Dense output layer, but in this thesis we did not do this.

Lastly, the number of hidden layers and the number of neurons in the hidden LSTM layer are important to tune. In general, a large enough single layer network can approximate any other neural network [47]. There is no default number of hidden layers in Keras. However, for most problems, one is sufficient [48]. We try one and two LSTM layers in our network: [1, 2].

There is also no default number of neurons in the hidden layer. In the literature, opinions vary about what is the best number. What is known, is that the number should be between the dimension of the input and output layer. The dimension of the input layer is the total number of different locations in all sequences (=274), while we only have one output feature, whether or not a user has a job. In order to get to a default value, we can make use of certain rules of thumb. One gets a decent performance if the number of neurons in the hidden layer equals the mean of the neurons in the input and output layers [48], which equals 137 in our case. Another rule of thumb is a size of 2/3 the size of the input layer, plus the size of the output layer, equalling 184. We take this as the default value. However, we risk that we overfit if we take too many neurons, so we test a smaller number of 47 as well, resulting in the following number of neurons in the hidden layer: [47, 137, 184].

8.2.5 Performances of the LSTM model

In this section we discuss the performances of the LSTM model for 1 and 2 hidden layers, respectively. We ran this separately since the running time of the model would be too large when combining both options, that is running the model for all combinations of examined options of the hyperparameters as in Table 6 including the options of 1 and 2 hidden layers at the same time.

	Performance before tuning		**Performance after tuning**	
	Accuracy	Balanced Accuracy	Accuracy	Balanced Accuracy
LSTM	0.737	0.493	0.608	0.583

TABLE 5: Performance of the LSTM network with 1 hidden layer before and after the hyper-parameter tuning.

Hyperparameter	**Default value**	**Examined options**	**Best option**
Batch size	32	[16,32,64]	32
Number of epochs	20	[20,40,80]	80
Neurons in hidden layer	137	[47,137,184]	184

TABLE 6: Grid search for best combination of hyper-parameters of the LSTM network with 1 hidden layer.

	Performance before tuning		**Performance after tuning**	
	Accuracy	Balanced Accuracy	Accuracy	Balanced Accuracy
LSTM	0.702	0.476	0.673	0.575

TABLE 7: Performance of the LSTM network with 2 hidden layers before and after the hyper-parameter tuning.

Hyperparameter	**Default value**	**Examined options**	**Best option**
Batch size	32	[16,32,64]	64
Number of epochs	20	[20,40,80]	80
Neurons in both hidden layer	137	[47,137,184]	137

TABLE 8: Grid search for best combination of hyper-parameters of the LSTM network with 2 hidden layers.

Observation 6: *As can be seen in Tables 5 and 7, the balanced accuracy (BA) is now always lower than the accuracy, which is contrary to Observation 1.*
This suggests that it is not true that in any case for any entries of the confusion matrix BA (in Equation (56)) is greater than the accuracy (in Equation (51)), which also follows from their definitions.

Observation 7: *As can be seen in Tables 5 and 7, tuning does not improve the accuracy (*$0.608 < 0.737$ *and* $0.673 < 0.702$*), but does improve the balanced accuracy for both the network with 1 hidden layer and 2 hidden layers (*$0.583 > 0.493$ *and* $0.575 > 0.476$*).*

Since we chose the balanced accuracy as being the best performance metric for our dataset, the tuning has the desired effect.

Observation 8: *As can be seen in Tables 5 and 7, having 2 hidden layers instead of 1 does not improve the performance (expressed in balanced accuracy). This is true before tuning (*$0.476 < 0.493$*) as well as after tuning (*$0.575 < 0.583$*).*
As mentioned in Section 8.2.4, for most problems, 1 hidden layer is sufficient. The same goes for our LSTM network.

Observation 9: *As can be seen in Tables 6 and 8, the best combination of hyper-parameters is different for an LSTM with 1 hidden layer compared to an LSTM with 2 hidden layers.*
Only the best option for the number of epochs is equal between the two LSTMs (both 80). The best batch size is higher for the LSTM with 2 hidden layers (64>32) as well as the best number of neurons in the hidden layer(s) (184>137). This tells that each network with a different architecture has different best options for the hyperparameters.

8.2.6 Comparing the performances of the Markov and LSTM model.

By comparing the balanced accuracy's of the different types of models, namely the Markov and LSTM model, we come to the following observation:
Observation 10: *The Markov model performs better in classifying users with or without a job than the baseline LSTM neural network.*
This conclusion is based on the balanced accuracy (BA) of the best modeling choice 2 of the Markov model (0.786) which is significantly higher than the highest BA achieved in the LSTM network for 1 hidden layer (0.583).

9 Conclusions and recommendations

In this section we come back to our research goal and give some recommendations.

9.1 Conclusions

In Section 1 we posed the following research goal: *show that a DTMC performs well in classifying users with or without a job by comparing the performance in the form of the balanced accuracy of this DTMC with the performance of a baseline LSTM neural network.* We can say that we achieved this goal, since after selecting the right metric in the context of the research in the form of the balanced accuracy (BA), the BA of the best modeling choice of the Markov model (0.786) is higher than the highest BA achieved in the LSTM network for 1 hidden layer (0.583). Of course, for other classification questions apart from the question if a user has a job or not the results could differ, yet the Markov chain model works well and is simple to interpret. Thus, it is a promising approach to classification. If we perform the same procedure for other questions, it could well be that the LSTM neural network performs better than the Markov model. Moreover, it could also be that the LSTM starts performing better when we add even more layers to the LSTM network as in Figure 37. This could be in the form of a Dropout layer as mentioned in Section 8.2.4, but also by adding even more hidden LSTM layers (that is, even more than two), to obtain a deep learning problem.

9.2 Recommendations

9.2.1 Experimental and computational recommendations

There is room for improvement regarding the Markov model. First of all, it is best to try as much different combinations of parameters as possible. In this research, we only investigated two different values for the parameters: 'time resolution', 'hourly location' and 'location types and days of the week'. Moreover, there exist also Markov models different than the discrete-time Markov model. There are Markov chains of a higher order than 1, as seen in Equation 31, a variable-order Markov chain in which the order may vary for each location based on its context and a continuous-time Markov chain.
Also the LSTM network can be expanded, with even more dropout and/or hidden layers, as mentioned in the Conclusion 9.1. This is all with the goal of obtaining higher (and robust) performances of the models.
Instead of using different classification models, it is also possible to directly cluster users into profiles using an unsupervised clustering model. This could be a faster approach, but it most likely not as accurate and reliable.
As an alternative to the Jensen-Shannon divergence (as presented in Section 5) to classify users, the use of the MLSE as introduced in Section 3.3 is also promising. If we remain at the example of classifying whether a user has a job or not, there is potential we could use the differences in the MLSE between both types of users to classify them. We could expect a low MLSE for users without a job throughout the day, while users with a job would probably present two peaks around times that these users commute to work.

9.2.2 Business recommendations

There are some business recommendations. First of all, the dataset can be improved. It would be good to only include those users in the dataset from which we know for sure that they have the correct (and thus the latest) demographic characteristics which was

discussed in Section 3.2. This can for example be done by performing surveys about these characteristics more frequently, not only once every year. Secondly, in order to get more reliable results, it is better to include trip-data over a longer period than 3 months, but more important is to include data of more users. In this research we only had 171 total users, while only having 43 users without a job, leaving very few users in the test set to perform predictions on. Another interesting research direction is to use the classification models on digital data that represent other aspects of human behavior (other than the location sequences); for example, classifying users by their web surfing histories or by their actions on their mobile phones. Of course, this is only an option if the relevant data is available for Mobidot.

9.2.3 Towards non-binary classification

The next step in the research is to perform the same procedure for different questions than the job question and afterwards combine these questions into user profiles. This is not straightforward however.
The last and most important step is to test if and how user classification can be used to improve activity recognition. This was not the scope of this project, yet is the ultimate goal of the company Mobidot. The improved activity recognition would provide valuable answers to Mobidot's customers for market research, panel surveys, urban policy development, and impact evaluation. As an example, the data can be used as an input for government policies to achieve a smoother and better distributed traffic flow.

References

[1] Barbara Furletti, Paolo Cintia, Chiara Renso, and Laura Spinsanti. "Inferring human activities from GPS tracks". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* August (2013). DOI: 10.1145/2505821.2505830.

[2] Juhong Liu, Ouri Wolfson, and Huabei Yin. "Extracting semantic location from outdoor positioning systems". In: *Proceedings - IEEE International Conference on Mobile Data Management* 2006.June (2006), pp. 73–80. DOI: 10.1109/MDM.2006.87.

[3] Zhixian Yan, Dipanjan Chakraborty, Christine Parent, Stefano Spaccapietra, and Karl Aberer. "Semantic trajectories: Mobility data computation and annotation". In: *ACM Transactions on Intelligent Systems and Technology* 4.3 (2013). DOI: 10.1145/2483669.2483682.

[4] Ming Yan, Shuijing Li, Chien Aun Chan, Yinghua Shen, and Ying Yu. "Mobility prediction using a weighted markov model based on mobile user classification". In: *Sensors* 21.5 (2021), pp. 1–20. DOI: 10.3390/s21051740.

[5] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.May 2014 (2011), pp. 2825–2830.

[6] Shreya Ghosh and Soumya K. Ghosh. "Modeling of human movement behavioral knowledge from GPS traces for categorizing mobile users". In: *26th International World Wide Web Conference 2017, WWW 2017 Companion* (2017), pp. 51–58. DOI: 10.1145/3041021.3054150.

[7] Harry Zhang. "The optimality of Naive Bayes". In: *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004* (2004), pp. 562–567.

[8] Shivam Agarwal. *Data mining: Data mining concepts and techniques*. 2014, pp. 203–207. DOI: 10.1109/ICMIRA.2013.45.

[9] Felipe Rocha de Araújo, Denis Lima Rosário, Kassio Machado, Eduardo Coelho Cerqueira, and Leandro Villas. "TEMMUS: A Mobility Predictor based on Temporal Markov Model with User Similarity". In: *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. 2019, pp. 594–607. DOI: 10.5753/sbrc.2019.7389.

[10] Daniel Ashbrook and Thad Starner. "Using GPS to learn significant locations and predict movement across multiple users". In: *Personal and Ubiquitous Computing* 7.5 (2003), pp. 275–286. DOI: 10.1007/s00779-003-0240-0.

[11] Chen Cheng, Haiqin Yang, Irwin King, and Michael R. Lyu. "Fused matrix factorization with geographical and social influence in location-based social networks". In: *Proceedings of the National Conference on Artificial Intelligence* 1 (2012), pp. 17–23.

[12] Haiping Ma, Huanhuan Cao, Qiang Yang, Enhong Chen, and Jilei Tian. "A habit mining approach for discovering similar mobile users". In: *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web* April (2012), pp. 231–240. DOI: 10.1145/2187836.2187868.

[13] Nathan Eagle and Alex Sandy Pentland. "Eigenbehaviors: Identifying structure in routine". In: *Behavioral Ecology and Sociobiology* 63.7 (2009), pp. 1057–1066. DOI: 10.1007/s00265-009-0739-0.

[14] Sepp Hochreiter and Jürgen Schmidhuber. "LSTM can solve hard long time lag problems". In: *Advances in Neural Information Processing Systems* (1997), pp. 473–479.

[15] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems* 6.2 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[16] J ¨ Urgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. "Training Recurrent Networks by Evolino". In: *Neural Computation* 19.3 (2007), pp. 757–779.

[17] Luca Pasa and Alessandro Sperduti. "Pre-training of recurrent neural networks via linear autoencoders". In: *Advances in Neural Information Processing Systems* 4.January (2014), pp. 3572–3580.

[18] Jinmiao Chen and Narendra S Chaudhari. "Segmented-Memory Recurrent Neural Networks". In: *IEEE Transactions on Neural Networks* 20.8 (2009), pp. 1267–1280.

[19] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[20] Eduardo F. Morales and Julio H. Zaragoza. "An introduction to reinforcement learning". In: *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*. 2011, pp. 63–80. DOI: 10.4018/978-1-60960-165-2.ch004.

[21] Shan Jiang, Joseph Ferreira, and Marta C. González. "Clustering daily patterns of human activities in the city". In: *Data Mining and Knowledge Discovery* 25.3 (2012), pp. 478–510. DOI: 10.1007/s10618-012-0264-z.

[22] Irad Ben-Gal, Shahar Weinstock, Gonen Singer, and Nicholas Bambos. "Clustering Users by Their Mobility Behavioral Patterns". In: *ACM Transactions on Knowledge Discovery from Data* 13.4 (2019), pp. 0–28. DOI: 10.1145/3322126.

[23] Mohamed Kafsi, Matthias Grossglauser, and Patrick Thiran. "The entropy of conditional Markov trajectories". In: *IEEE Transactions on Information Theory* 59.9 (2013), pp. 5577–5583. DOI: 10.1109/TIT.2013.2262497.

[24] Richard Serfozo. *Basics of Applied Stochastic Processes*. Springer Science & Business Media, 2009.

[25] Jingjing Wang and Bhaskar Prabhala. "Periodicity Based Next Place Prediction". In: *Procedings of the Mobile Data Challenge by Nokia Workshop in Conjunction with International Conference on Pervasive Computing (Pervasive '12)* (2012), pp. 1–5.

[26] Irad Ben-Gal, Gail Morag, and Armin Shmilovici. "CSPC : A Monitoring Procedure for State Dependent Processes". In: *Technometrics* 45.4 (2003), pp. 293–311.

[27] I. Ben-Gal, A. Shani, A. Gohr, J. Grau, S. Arviv, A. Shmilovici, S. Posch, and I. Grosse. "Identification of transcription factor binding sites with variable-order Bayesian networks". In: *Bioinformatics* 21.11 (2005), pp. 2657–2666. DOI: 10.1093/bioinformatics/bti410.

[28] S. Kullback. *Information theory and statistics*. 1968.

[29] Yifan Li, Jiawei Han, and Jiong Yang. "Clustering moving objects". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 617–622. DOI: 10.1145/1014052.1014129.

[30] Josh Jia Ching Ying, Eric Hsueh Chan Lu, Wang Chien Lee, Tz Chiao Weng, and Vincent S. Tseng. "Mining user similarity from semantic trajectories". In: *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks, LBSN 2010 - Held in Conjunction with ACM SIGSPATIAL GIS 2010* January (2010), pp. 19–26. DOI: 10.1145/1867699.1867703.

[31] Kenneth Wai Ting Leung, Dik Lun Lee, and Wang Chien Lee. "CLR: A collaborative location recommendation framework based on co-clustering". In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 305–314. DOI: 10.1145/2009916.2009960.

[32] Eric Hsueh Chan Lu and Vincent S. Tseng. "Mining cluster-based mobile sequential patterns in location-based service environments". In: *Proceedings - IEEE International Conference on Mobile Data Management* (2009), pp. 273–278. DOI: 10.1109/MDM.2009.40.

[33] Mingqi Lv, Ling Chen, and Gencai Chen. "Mining user similarity based on routine activities". In: *Information Sciences* 236 (2013), pp. 17–32. DOI: 10.1016/j.ins.2013.02.050.

[34] Vincent S.M. Tseng and Kawuu W.C. Lin. "Mining sequential mobile access patterns efficiently in mobile web systems". In: *Proceedings - International Conference on Advanced Information Networking and Applications, AINA* 2 (2005), pp. 762–767. DOI: 10.1109/AINA.2005.248.

[35] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. "Finding similar users using category-based location history". In: *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems* 49 (2010), pp. 442–445. DOI: 10.1145/1869790.1869857.

[36] Prashant Krishnamurthy and Konstantinos Pelechrinis. "Location-Based Social Networks". In: *Advanced Location-Based Technologies and Services* (2013), pp. 127–144. DOI: 10.1201/b14940-7.

[37] Warren S Sarle. "Neural Networks and Statistical Models". In: *SAS USers Group International Conference* (1994), p. 13.

[38] Christopher Olah. *Understanding LSTM Networks*. Tech. rep. 2019, pp. 1–8.

[39] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[40] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning*. 2010. DOI: 10.1007/978-0-387-30164-8.

[41] Daniela Xhemali, Christopher J. Hinde, and Roger G. Stone. "Naive Bayes vs. Decision Trees vs. Neural Networks in the Classification of Training Web Pages". In: *International Journal of Computer Science* 4.1 (2009), pp. 16–23.

[42] Foster Provost. "Machine learning from imbalanced data sets 101". In: *Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets* (2000), p. 3.

[43] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic minority over-sampling technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357. DOI: 10.1613/jair.953.

[44] Qiong Wei and Roland L. Dunbrack. "The Role of Balanced Training and Testing Data Sets for Binary Classifiers in Bioinformatics". In: *PLoS ONE* 8.7 (2013), e67863. DOI: 10.1371/journal.pone.0067863.

[45] Francois Chollet et al. *Keras*. 2015.

[46] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15.

[47] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: 10.1016/0893-6080(91)90009-T.

[48] Heaton T Jeff. *Introduction to Neural Networks with Java*. Heaton Research, Inc., 2005.

10 Appendix

10.1 List of locations

FIGURE 38: List of all locations.