JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Katharina Prinz

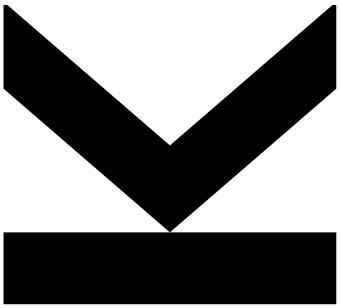
Submitted at
**Institute of
Networks and Security**

Supervisor
**Univ.-Prof. Priv.-Doz.
DI Dr. Rene Mayrhofer**

Co-Supervisor
Dr. Muhammad Muaaz

February 2019

Next Place Prediction with Hidden Markov Models



Master Thesis
to obtain the academic degree of
Diplom-Ingenieurin
in the Master's Program
Computer Science

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Abstract

The prediction of future locations can be useful in various settings, one being the authentication process of a person. In this thesis, we perform the prediction of next places with the help of a HMM. We focus on models with a discrete state space and thus need to discretise the data. This is done by pre-processing the raw, continuous location data in two steps. The first step is the extraction of stay-points, i.e. regions in which a person spends a given time period at. In the second step, multiple stay-points are grouped with the clustering algorithm DBSCAN to form significant places. After pre-processing, we train a HMM with a state and observation space that correspond to the extracted significant places. Based on the previously observed location, our model predicts the next place for a person. In order to find good models for next place prediction, we did experiments with two datasets. The first one is the Geolife GPS trajectory dataset from Microsoft, which consists of GPS traces. The second dataset was self-collected and contains additional data obtained from WiFi and cell towers. Our final model achieves a validation accuracy higher than 0.95 on both datasets. However, a prediction accuracy reaching from 0.8 to 0.99 of a model that solely predicts *noise* as its future location, leads us to the conclusion that the datasets, as well as the pre-processing step need further refinements for our HMM to encapsulate more valuable information.

Zusammenfassung

Die Vorhersage von zukünftigen Aufenthaltsorten eines Menschen ist in vielen Anwendungen hilfreich. Einer davon ist der Authentifizierungsprozess von Menschen. In dieser Thesis wird der wahrscheinlichste zukünftige Standort mithilfe eines HMM vorhergesagt. Um kontinuierliche Positionsdaten zu diskretisieren, und sie dadurch auf eine Verarbeitung durch ein Modell welches einen diskreten Zustandsraum aufweist vorzubereiten, führen wir zuerst eine Vorverarbeitung in zwei Schritten durch. Der erste Schritt ist das Eruieren von so genannten stay-points, die Regionen, in welchen Personen eine längere Zeit verweilen, darstellen. Im zweiten Schritt werden mithilfe des Cluster-Algorithmus DBSCAN mehrere dieser Punkte gruppiert, um bedeutungsvolle Standorte zu formen. Nach dem Durchführen der Vorverarbeitung wird ein HMM trainiert. Der Zustands- sowie Beobachtungsraum dieses Modells entspricht den zuvor bestimmten bedeutungsvollen Standorten. Basierend auf einem vergangenen Standort, sagt unser Modell die nächste, meist wahrscheinliche Position einer Person vorher. Um die Genauigkeit dieser Vorhersage zu testen, wurden Experimente mit zwei verschiedenen Datensätzen durchgeführt. Das Geolife GPS trajectory dataset von Microsoft besteht ausschließlich aus GPS Folgen. Unser eigener Datensatz beinhaltet zusätzlich Daten, welche mithilfe von WiFi und Mobilfunkmasten gesammelt werden können. Das daraus resultierende endgültige Modell erreicht eine Validierungsgenauigkeit von über 0.95 für beide Datensätze. Ein Modell, dass nur *noise* als nächsten Standort vorhersagt und damit eine Genauigkeit von ungefähr 0.8 bis zu 0.99 erzielt, lässt uns jedoch darauf schließen, dass beiderlei Datensätze sowie der Vorverarbeitungsprozess weitere Verbesserungen benötigen, damit unser HMM hochwertigere Informationen darstellen kann.

Acknowledgements

Writing this thesis would not have been possible without the help of many different people. First I want to thank Prof. Mayrhofer for finding a topic suited for my interests, and the supportive and patient supervision. Also Dr. Muaaz, for all ideas, discussions, and support within the course of working on my thesis. Thanks also to everyone else at the INS institute who helped with ideas, questions or suggestions of any kind. Thanks to Dr. Roland, who helped me find my way with NFC tags.

Another big thank you to all users within our collected dataset, who spent a lot of time recording and annotating the data for this project. Thanks to everybody who simplified the implementation of experiments by providing their work openly available for everyone.

Finally, I also want to thank my friends, and in particular my family, for all their support and help. Thanks to my parents. Thanks for proof-reading, particularly Pieter-Jan, and for the patience in doing so. Thanks for always listening and believing in me all the way throughout this work.

Contents

List of Figures

1 Introduction

The increasing prevalence of smart phones and other Global Positioning System (GPS)-enabled devices have brought various new possibilities to the task of location data acquisition [31]. Knowing where a person is located at a certain point in time can be considered highly sensitive information, however, as a series of locations could be used to infer their identity (cf. [13]). Nevertheless, finding these traces - or *tracking* - allows, among others, improved content for context-aware applications [12] or location-aware recommender systems [4].

Location history has not only been used for the purpose of providing a better user experience so far. Due to their sensitive nature, location traces are suited for determining the geographic position of a person up to a certain precision. This degree of confidence in a current or future location can in turn assist in tasks such as authentication, cf. [19].

In this thesis we want to look at the usage of location history from a similar, security related angle. A model that attempts to predict human trajectories based on their often high degree of spatial and temporal regularity is learned [25]. The model, describing the series of locations a user visits regularly, is used to predict the next geographic position of that person. If we obtain a second location directly from the environment, originating from sensors such as mobile phones, our prediction can in turn be compared to this location in question. Later on, the distance between these two locations can be used as a base to compute a level of confidence, denoting our belief in the person actually being at the location in question. This could then serve as a base for authentication decisions.

When working in an indoor environment, computing the location of a user can be complicated by poor satellite signals [35]. These cases require alternatives to the usage of GPS data. Technologies often used for this are WiFi, Bluetooth or infrared ray [35]. Also ultrasound sensors and vision systems have been applied to track human locations [12] before, as well as Call Detail Records (CDRs) and data traffic of mobile networks [25]. The work in this thesis will, however, mainly focus on GPS data as its input.

1.1 Motivation

Despite the sensitive nature of the data, plenty different factors motivate the application of location trajectories for next place prediction. Often, these originate in the desire of improving user experience. Using human trajectories to predict locations can, for example, allow service providers location-based advertising, early warning systems or traffic planning [25], as well as efficient network resource management schemes [1].

Dedicating a separate section specifically to the topic, Bobadilla et al. state that also a number of recommender systems incorporates location data [4]. This way, recommendations can be given with respect to a current, estimated location of a user.

Computing user similarities based on location histories can also be used to improve the user experience. In general, the similarity of users can be used to find e.g. potential friends on one hand or improve marketing strategies with reasonable recommendations on the other [16].

A second factor motivating the usage of location trajectories focuses on security, more specifically on authentication. Extracting significant location and recognising daily activity patterns with the help of human trace histories can support continuous authentication of users [19]. Also Fridman et al. use location data as an additional source to verify an identity [9].

1.2 Outline

The rest of this thesis is organised as follows. Chapter 2 contains a theoretical background and foundation of the work presented later on, followed by current developments in the field in chapter 3. Thereafter, chapter 4 introduces a theoretical view of the model we used to approach the issue of location tracking, followed by a closer description of the according implementation. Experiments and their results are presented in chapter 5, before chapter 6 contains a summary of our conclusions.

2 Background

2.1 The Spherical Coordinate System

Spherical coordinates are a common tool for describing the geographic location of a person. The 3D-coordinates are defined as the triplet radius r, *longitude* ϕ and polar distance or *latitude* θ [24]. Figure 2.1 visualises the three factors describing a point P on a sphere. Radius r denotes the length of radius vector $\vec{OP}$, i.e. the distance of point P to the origin O. The longitude ϕ describes the angle measured from reference vector x to an orthogonal projection of $\vec{OP}$ to the reference plane, and latitude θ is defined as the angle between the upwards direction and vector $\vec{OP}$.

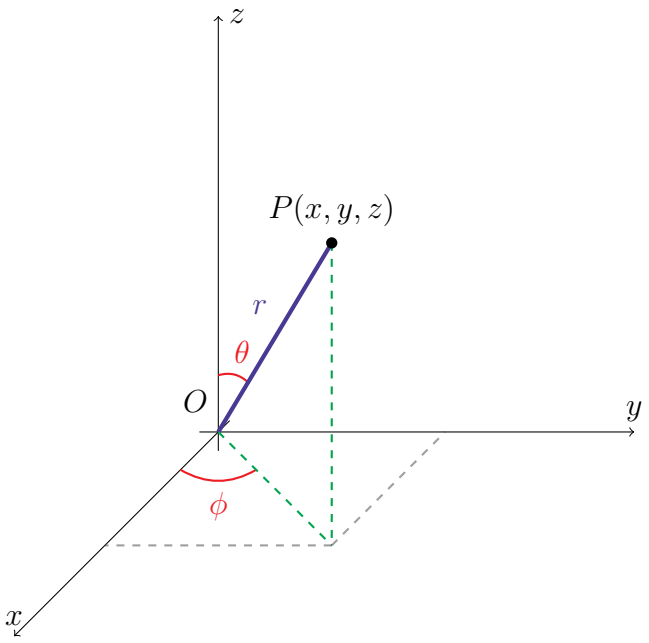


Figure 2.1: Point P in spherical coordinates

The elements x, y and z depicted in Figure 2.1 denote the Cartesian coordinates of M. Given the triplet (x, y, z), according spherical coordinates can be obtained by [24]

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \phi &= \arctan \frac{y}{x} \\ \theta &= \arccos \frac{z}{r}. \end{aligned} \tag{2.1}$$

Oppositely, Cartesian coordinates can also be expressed by means of (r, ϕ, θ) [24], i.e.

$$\begin{aligned} x &= r \sin\theta \cos\phi \\ y &= r \sin\theta \sin\phi \\ z &= r \cos\theta. \end{aligned} \tag{2.2}$$

As any number of added or subtracted full turns to an angular measure do not change the angle and therefore the point itself, any point can be described in infinitely many ways using spherical coordinates. In order to ensure uniqueness, angles ϕ and θ are therefore often restricted [24]. Polyanin and Manzhirov suggest to assume $0 < \phi \leq 2\pi$ and $0 \leq \theta \leq \pi$, or $-\pi < \phi \leq \pi$ and $0 \leq \theta \leq \pi$.

2.1.1 The Global Positioning System

The *Global Positioning System* is a way to obtain geographic coordinates of a location on the earth. For this, satellites broadcast radio signals at the speed of light containing their location, status and precise time [22]. GPS devices on the other hand receive satellite signals, and compute the respective distance between them based on the time of arrival. The location on earth in all its three dimensions can be computed as soon as a device knows the distance to at least four satellites.

2.1.2 Processing Geographic Coordinates

Most machine learning algorithms are not able to find interesting information in a raw signal. Therefore, a common step that comes before learning consists of finding useful features that represent the information necessary for the algorithm to find patterns. Whenever we want to apply machine learning algorithms to raw geographic coordinates, i.e. longitudes and latitudes, a common pre-processing routine is to find places that are significant to a user. Before these *significant places* can be found with methods such as clustering, the raw locations are typically grouped into stay-points [16].

A *stay-point* describes a geographic area a user stayed in for a certain amount of time [16]. It is often defined by a radius to define the size of the area, as well as a time span defining the minimal time frame that has to be spent there. If a sequence of locations is within the radius and spans the minimal time span, we call it a stay-point. In addition to a mean coordinate, the time of entering and leaving the assigned region of a stay-point is frequently used as a characterisation thereof.

As an example, we can think of a shopping mall representing a possible significant location [30]. Different shops within this shopping mall would be denoted by stay-points. In other words, a significant location consists of multiple stay-points, or, multiple stay-points need to be clustered in order to make up one significant location. A visual explanation of the connection between initial location data, stay-points and two exemplary, final significant places is given in Figure 2.2.

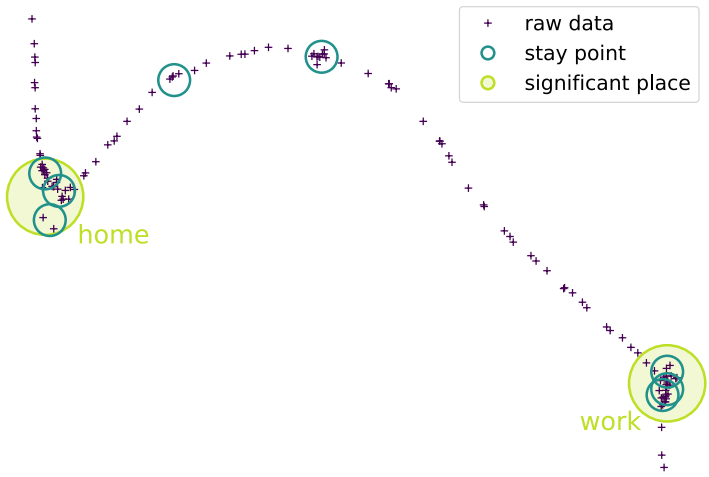


Figure 2.2: Relation of GPS trajectories, stay-points and significant places

2.2 Clustering

Putting different objects in groups that are similar to each other is a way of finding and expressing regularities in data [18]. Within the field of unsupervised machine learning, this grouping or segmentation of data is often called *clustering*. The predictive powers of good clustering often motivate the usage of this technique. For us, however, the lossy compression that can be achieved via clustering will aid in pre-processing and discretising the locational input data.

More precisely, with clustering we try to group data so that observations within a cluster are related more closely to each other than to objects from different clusters [11]. To achieve this, a variety of distance or dissimilarity measures can be used. Apart from the distance measure, different types of clustering are differentiated by other factors like e.g. assumptions about underlying probability models of data.

2.2.1 Combinatorial Clustering

Clustering algorithms that do not make use of probabilistic models to describe the data are called *combinatorial* [11]. This prominent set of algorithms assigns each object or observation to one of a predefined number of clusters, typically labelled by an integer $k \in \{1, \ldots, K\}$.

The most popular representative of combinatorial algorithms is *K-means* [11]. This algorithm assigns every element to the cluster with the closest centre iteratively. Based on this assignment, new cluster-centres are computed until the assignments do not further change. K-means employs squared Euclidean distance as dissimilarity measure, and often initialises cluster-centres randomly. How clusters and their according centres can change in the course of iterations is illustrated in Figure 2.3.

Various algorithms which are strongly related to the base k-means clustering approach

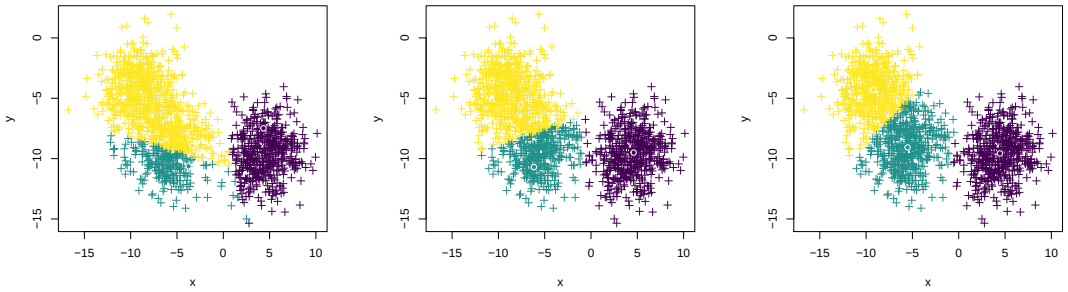


Figure 2.3: Changes during iterative k-means clustering

exist. Popular examples of such extensions are *K-medoids* and *CLARA* [29].

2.2.2 Distribution-Based Clustering

In contrast to the previous set of clustering algorithms, *distribution-based clustering* relies on an assumption on the distribution underlying the data [29]. Typical algorithms of this category are DBCLASD as well as *Mixture Models*.

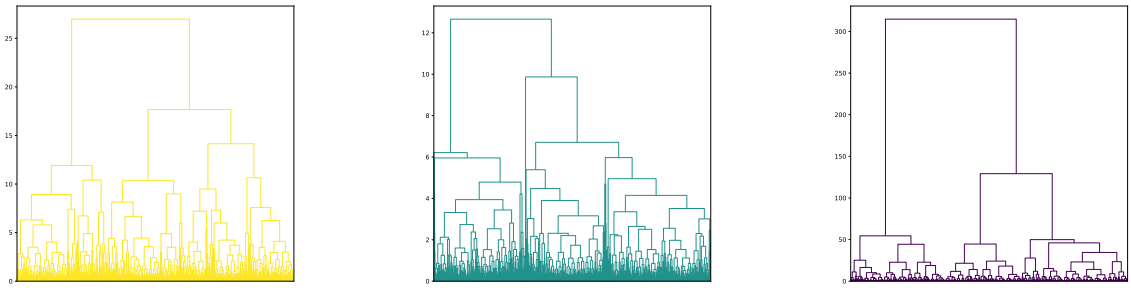
Mixture models assume that observations are independently sampled from probability density functions that are a composition of different, well-understood component distributions [11]. The goal of the algorithm is to find parametrisations for these simple distributions, so that the distribution of the available data matches that of the composition or *mixture* as well as possible. The clustering then follows from finding the component to which each point most likely belongs. A prominent mixture model that can be used for clustering is the GMM.

2.2.3 Hierarchical Clustering

Given a specific dissimilarity measure, *Hierarchical Clustering* looks at data segmentation on different levels [11]. Every cluster in the lowest level has exactly one observation as its element, while at the highest level a single cluster contains the entire data. Clusters at arbitrary hierarchical levels can be obtained by merging clusters at the next lower level (*agglomerative* paradigm) or by splitting clusters at the next higher level (*divisive* paradigm). Independent of the used paradigm, every level in the hierarchy corresponds to a possible clustering result.

Hierarchical clustering allows the usage of various *linkages*, which describe the way dissimilarities between different clusters are computed [11]. Figure 2.4 illustrates that the application of different linkages can indeed lead to divergent results. The depicted dendrograms visualise how data is partitioned by hierarchical clustering algorithms. By cutting a dendrogram horizontally, branches fall apart and form the expected clusters.

Agglomerative hierarchical clustering has been studied extensively in literature, and exhibits wide-ranging applicability [11]. The divisive paradigm on the other hand is



(a) Complete linkage (b) Average linkage (c) Ward linkage

Figure 2.4: Hierarchical clustering results based on different linkage methods provided by the Python library *scikit learn* [23]

often preferred when looking for a *small* number of clusters. To perform this kind of hierarchical clustering, one can use combinatorial clustering methods such as K-means with $K = 2$ to split clusters iteratively.

2.2.4 Density-Based Clustering

In a lot of applications, domain knowledge necessary for choosing the predefined number of clusters is not available [7]. Approaches using density-based clustering try to bypass this issue by taking into account that the density within a cluster tends to be a lot higher than outside of a cluster. A prominent representative of this kind of algorithms is DBSCAN.

The key idea of DBSCAN is to extract clusters of observations, which contain at least a minimum number of points within a specific radius [7]. In other words, the density of a neighbourhood has to be higher as a defined threshold. For this purpose, DBSCAN supports the usage of various distance functions, which simultaneously control the shape of a neighbourhood. Two parameters regulate previously stated radius and minimum number of points. The former one is often denoted as *epsilon* or ϵ.

Further examples of this kind of clustering are *DJ-Cluster*, a density- and join-based algorithm which especially tackles potential memory issues of DBSCAN [34], and OPTICS, overcoming the sensitivity of DBSCAN to its two parameters [29].

2.2.5 Further Developments in Cluster Analysis

Prior approaches were only an excerpt of all possibilities that allow the clustering of data. There exist a lot of algorithms tackling the issue from entirely different angles, some of which have been developed more recently. Examples include *Affinity Propagation*, which was proposed in 2007 and performs clustering via message passing between data points [8].

Kernel-based clustering methods map input data via a non-linear kernel functions to a higher dimensional feature space, in which original clustering algorithms are applied [29]. Clustering based on *density and distance* was proposed in 2014. These kind of algorithms choose cluster centres which exhibit a high local density and a high distance to other potential cluster centre points simultaneously. After the selection of centre points, the remaining data is grouped based on the nearest cluster. Other clustering algorithms can be found in [29].

2.2.6 Hard vs. Soft Clustering

The majority of clustering algorithms discussed thus far, e.g. the prominent K-means algorithm, assign data-points deterministically to clusters [11]. In literature, this is often being referred to as *hard* clustering. This can also be done in a probabilistic fashion, however, such that every data-point belongs to a cluster with a certain probability, making it a *soft* assignment.

Figure 2.5 illustrates the connection of hard and soft clustering based on the close relation of the K-means algorithm and the process of finding GMM parameters [11]. The left hand side shows two images, each with two Gaussian densities $g_0(x)$ and $g_1(x)$. The standard deviation σ differs from top to bottom images; while the upper Gaussian densities are generated with $\sigma = 1.0$, the lower plots feature densities with $\sigma = 0.1$.

The images on the right hand side of Figure 2.5 illustrate the relative densities or *responsibilities*, which represent a *soft* cluster assignment [11]. Responsibilities can be close to 0.5 whenever the standard deviation is relatively large. Oppositely, if the standard deviation, σ, converges to zero, the responsibilities approach 1.0 for the closest cluster, and 0.0 for the other. For example, the highlighted data-point x in the upper image is assigned 0.73 by $g_0(x)$ and 0.27 by g_1(x). In the lower image, on the other hand, $g_0(x)$ and $g_1(x)$ assign 1.0 and 0.0 to x respectively. This means that x is assigned to former cluster with a probability of 1.0.

When the probabilities assigned to a data-point become 0.0 and 1.0, as illustrated in the lower part of Figure 2.5, we reach a *hard* assignment, causing K-means and GMMs coincide.

2.3 Hidden Markov Models

2.3.1 Notation

The subsequent notation largely follows the work of Koller and Friedman [15]. Capital letters, e.g. X, denote *random* variables. $Val(X)$ is the corresponding domain of a random variable X. Bold, italic capital letters represent sets of random variables, e.g. $\boldsymbol{X} = \{X_1, X_2, \dots\}$. The set of all random variables a model is defined over, is denoted by $\mathcal{X}$.

In a dynamic setting, we use superscripts to incorporate time. Concretely, $X^{(t)}$ denotes the random variable representing a value of X at time t. As X, without superscript t,

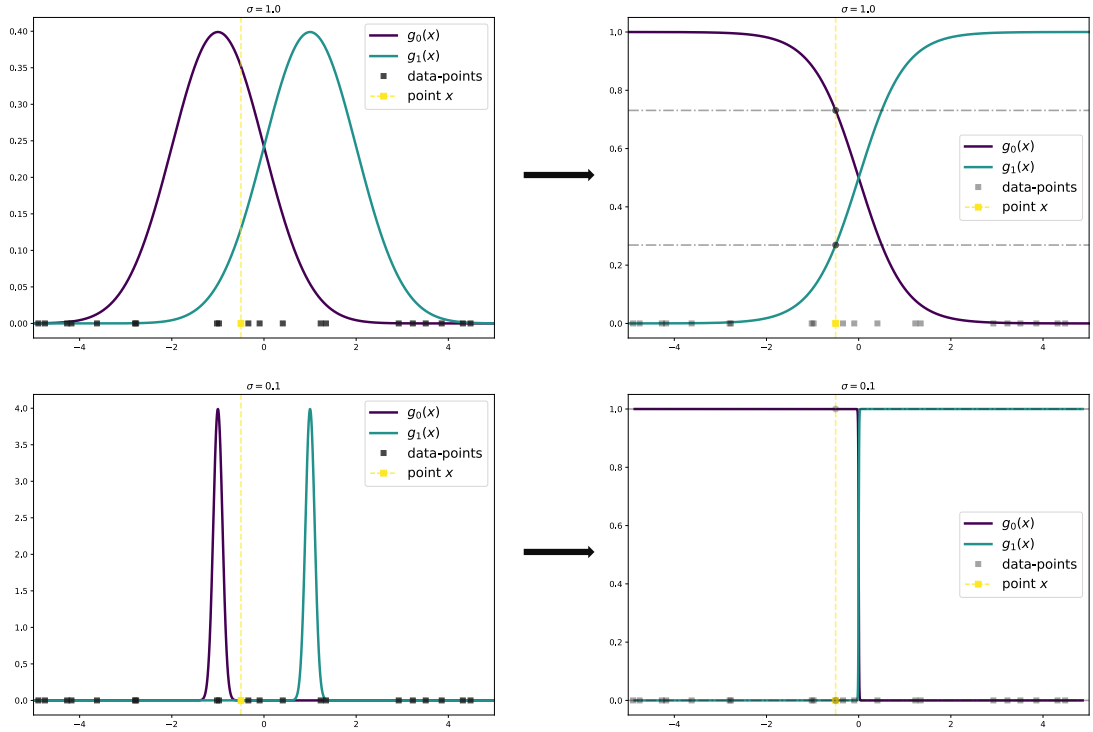


Figure 2.5: Left-hand side: Two illustrations of two Gaussian densities with $\sigma = 1.0$ and $\sigma = 0.1$, with various data-points to be clustered at the bottom of the images.
Right-hand side: Relative densities or responsibilities, used to make a *soft* assignment to said data-points.

does not take a single value in this case, but rather has a value for every time-step t, it is no longer a random variable. Instead, X denotes a *template* variable when we operate in a dynamic setting. The domain $Val(X)$ once more describes all values a template variable X can take at time t.

Furthermore, $P(X)$ is a probability distribution, and $P(X = x)$ is a probability. To abbreviate this kind of notation, we will write $P(x)$ instead of $P(X = x)$ or $P(x, y)$ as a shorthand variant for $P(X = x, Y = y)$. $P(X, Y)$ denotes the probability distribution over the discrete variables X and Y. $P(X \mid Y)$ will be used for the conditional distribution of X given Y.

Subsequent sections also contain matrices described by bold capital letters, e.g. $\mathbf{A}$, and vectors represented by a bold, lower-case letter such as $\boldsymbol{\pi}$. For the definition of used terms and further information about the basics of probability theory, we refer to [15].

2.3.2 Probabilities

Another term established in probability theory is the notion of *events*. An event denotes the assignment of values to some or all random variables describing an environment [15]. These values originate from the domains of respective random variables. *Atomic events* assign a value to each random variable in $\mathcal{X}$, therefore corresponding to a specific state the environment can be in.

Given a set $\mathcal{S}$ of atomic events, we have that a probability distribution P is a function $P : \mathcal{S} \mapsto \mathbb{R}$, satisfying the following constraints [15]:

1. $P(\alpha) \geq 0$ for all $\alpha \in \mathcal{S}$
2. $P(\Omega) = 1$ where $\Omega = \bigcup_{\alpha \in \mathcal{S}} \alpha$, and finally
3. if $\alpha, \beta \in \mathcal{S}$ and $\alpha \cap \beta = \emptyset$, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$

A probability distribution P assigns some value $P(\alpha)$ between zero and one to a specific event α. This value is called the *probability* of the event. $P(\alpha) = 1$ suggests that we are certain that the event α will occur. $P(\alpha) = 0$, on the other hand, indicates certainty that the event will not occur.

There exist two main philosophical stances on the interpretation of these probability values [15]. On one side, there is the *frequentist* interpretation, where probabilities indicate frequencies of events. Concretely, a probability is viewed as the relative count of event occurrences after infinitely many experiments. In the *subjectivist* or Bayesian setting, on the other side, probabilities resemble a subjective degree of belief for an event to occur.

2.3.3 Bayesian Models

A Bayesian Network (BN) is a graphical model based on the concepts of probability theory [15]. It allows a compact representation of a system, by taking advantage of conditional independences between variables describing its environment.

The core of a BN, network structure $\mathcal{G}$, is a directed acyclic graph where each node corresponds to one of the random variables in $\mathcal{X}$ [15]. This acyclic graph is structured so that it models local independencies of the random variables. This gives rise to an encoding of local independences

$$\mathcal{I}(\mathcal{G}) = \{(X \perp NonDesc(X) \mid Pa(X)\} \quad \forall X \in \mathcal{X}. \tag{2.3}$$

Here, $NonDesc(X)$ denotes non-descendent nodes of X in $\mathcal{G}$, and $Pa(X)$ represents all parents of X. More intuitively, Equation 2.3 conveys that every node in $\mathcal{G}$ is independent of its non-descendants, given its parents.

Figure 2.6 depicts a directed, acyclic graph with multiple nodes. If we try to find local independences for node X, we first need to determine parents $Pa(X) = \{P_1, P_2\}$ and

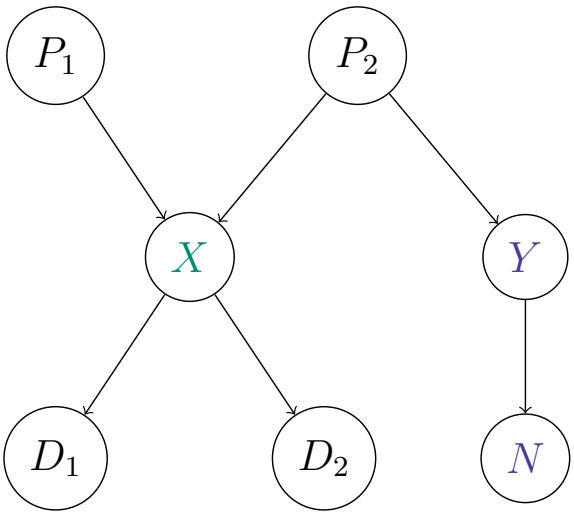


Figure 2.6: Graph $\mathcal{G}$ with node X, its parents P_1 and P_2, descendents D_1 and D_2, and non-descendent nodes Y and N

non-descendents $NonDesc(X) = \{Y, N\}$. Equation 2.3 then tells us, that for this graph $(X \perp Y, N \mid P_1, P_2)$.

The BN does not only model the independencies of $\mathcal{X}$, but it also represents the full joint distribution. To retrieve this joint distribution over $\mathcal{X}$ from the model, we can make use of the *chain rule* for Bayesian networks [15], i.e.

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i)). \tag{2.4}$$

The factors $P(X \mid Pa(X))$ are Conditional Probability Distributions (CPDs), and are associated with individual random variables in $\mathcal{G}$ [15].

2.3.4 Temporal Models

Often, systems act in an environment that evolves over time, resulting in a dynamic, as opposed to a static setting. A snapshot of relevant system attributes at time t, is called the *system state* [15]. It is an assignment of values to template variables $\mathcal{X}$, and therefore corresponds to a probabilistic event.

In addition to previous notions, $\boldsymbol{X}^{(t_1:t_2)}$ denotes the set $\{\boldsymbol{X}^{(t)} \mid t \in [t_1, t_2]\}$ for $\boldsymbol{X} \subseteq \mathcal{X}$ [15]. The assignment of values to this set of variables, i.e. an event, will accordingly be denoted as $\boldsymbol{x}^{(t_1:t_2)}$.

Atomic events in the dynamic setting are now called *trajectories*, and are assignments of values to all variables $\mathcal{X}^{(1:T)}$ for some duration T [15]. Temporal models therefore represent a joint distribution over these trajectories. Because arbitrarily long trajectories can result in a huge probability space, a lot of temporal Bayesian models work with simplifying assumptions to assure feasibility.

2.3.5 Dynamic Bayesian Networks

Dynamic Bayesian Networks (DBNs) are the basic temporal extension of Bayesian models. In order to compactly represent distributions over infinite trajectories, they are built upon three simplifying assumptions [15].

The first simplification is to assume discrete time. This can be achieved by splitting a continuous time-line into a set of time-steps regularly spaced with granularity Δ. Secondly, a DBN is assumed to be time invariant or *stationary*. Dynamic systems are stationary when $P(\mathcal{X}^{(t+1)} \mid \mathcal{X}^{(t)}) = P(\mathcal{X}^{(s+1)} \mid \mathcal{X}^{(s)})$ for any s and t, i.e. when system-state transitions stay the same for every time-step. The assumption of stationarity allows us to represent state transitions with a transition model $P(\mathcal{X}' \mid \mathcal{X})$, so that

$$P(\mathcal{X}^{(t+1)} = \xi' \mid \mathcal{X}^{(t)} = \xi) = P(\mathcal{X}' = \xi' \mid \mathcal{X} = \xi), \tag{2.5}$$

for any $t \geq 0$. Assignments to all variables in $\mathcal{X}$ are here denoted by ξ.

The third and final simplification of DBNs is the Markov assumption, which is explained in the next subsection.

2.3.6 The Markov Property

Over all template variables $\mathcal{X}$, a dynamic system satisfies the Markov assumption if [15]

$$(\mathcal{X}^{(t+1)} \perp \mathcal{X}^{(0:t-1)} \mid \mathcal{X}^{(t)}) \quad \forall t \geq 0, \tag{2.6}$$

i.e. the future only depends on the present, and not on the past.

Using the *chain rule* in combination with all simplifying assumptions of a DBN, we can express the joint distribution over trajectories in a dynamic setting compactly [15],

$$P(\mathcal{X}^{(0:T)}) = P(\mathcal{X}^{(0)}) \prod_{t=0}^{T-1} P(\mathcal{X}^{(t+1)} \mid \mathcal{X}^{(t)}). \tag{2.7}$$

In some situations, random variables at a time-step t, i.e. $\mathcal{X}^{(t)}$, do not solely depend on the variables of previous time-step $t-1$ [15]. Instead, they are influenced by variables of multiple past time-steps, i.e. $t-1, \ldots, t-k$. In these circumstances, dynamic systems do not strictly satisfy the Markov assumption. These systems are often being referred to as *semi-Markov* of order k, or shorter, k^{th} order Markovian.

2.3.7 The Hidden Markov Model

A Hidden Markov Model (HMM) is a state-observation model and a special case of a DBN [15]. Dynamic systems that can be modelled with a HMM, are described via a hidden, or *latent*, state variable $S^{(t)}$. Factors that can be observed in the environment of the system on the other hand, are expressed with a separate, visible observation variable $O^{(t)}$. As opposed to the discrete domain of the state variable, the observation variable can either be discrete or continuous [3]. In what follows, we will focus on HMMs

with a discrete observation variable, as is common in engineering and machine learning communities.

A HMM is characterised by the initial state probabilities $P(S^{(0)})$, typically denoted by vector $\boldsymbol{\pi}$, a state transition model $P(S' \mid S)$ encoded in matrix $\mathbf{A}$, and matrix $\mathbf{B}$ describes the observation model $P(O \mid S)$ [26]. This compact matrix representation is possible due to the discrete character of state and observation variables. In case latter variable is continuous on the other hand, the observation model is often modelled via continuous probability density functions, such as GMMs [3]. Figure 2.7 shows a HMM, for which the first observation is obtained at time $t = 1$.

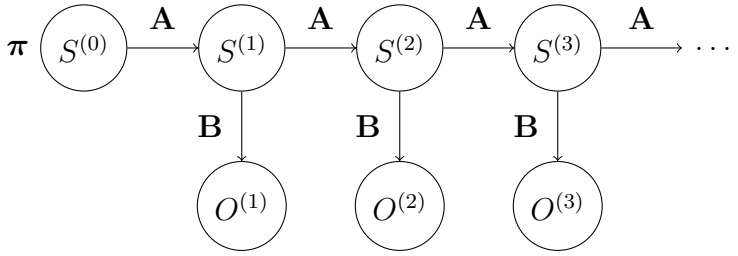


Figure 2.7: Graphical representation of an unrolled HMM with its parameters $\boldsymbol{\pi}$, $\mathbf{A}$ and $\mathbf{B}$

Probability measures $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$ with the two factors N and M, denoting the number of states and distinct observation symbols respectively, fully specify a HMM [26]. Subsequently, the complete set of model parameters will be abbreviated with $\theta = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$.

Next to being able to model how observations were emitted by dynamic systems, HMMs also allow us to directly generate observation sequences according to its parameters [26]. To make use of this generative function of the HMM, one can choose an initial state according to distribution $\boldsymbol{\pi}$. Thereafter, the state transition model can be applied to obtain a state for the next time-step, followed by using the observation model to retrieve the first observation of the sequence. Applying the state transition and observation model alternately can then be repeated until a desired time-step is reached.

2.3.8 Common Inference Tasks

Three of the most common reasoning tasks related to HMMs are *filtering*, *smoothing* and *prediction* [15]. Also the *decoding* problem is a frequent topic in literature, which is why it will subsequently be covered as well.

Filtering is a common reasoning task that is also termed *tracking* [15]. When performing the task of filtering, one computes $P(S^{(t+1)} \mid \boldsymbol{o}^{(1:t+1)})$ recursively with the *forward pass* [15], [26],

$$\begin{aligned} f^{(1:t+1)} &= P(S^{(t+1)} \mid \boldsymbol{o}^{(1:t+1)}) \\ &= \frac{1}{Z} \times P(o^{(t+1)} \mid S^{(t+1)}) \sum_{s^{(t)}} P(S^{(t+1)} \mid s^{(t)}) f_s^{(1:t)}. \end{aligned} \tag{2.8}$$

When subscripting f with a particular state-assignment s, we refer to a probability as opposed to the entire distribution, i.e. $f_s^{(1:t)} = P(s^{(t)} \mid \boldsymbol{o}^{(1:t)})$. Intuitively, the filtering task is the computation of a distribution over all hidden states, given everything we observed thus far. Z represents the factor that normalises this distribution.

Smoothing is the task to compute $P(S^{(t)} \mid \boldsymbol{o}^{(1:T)})$ for $t < T$, which is done with the *forward-backward algorithm* [26], i.e.

$$\begin{aligned} P(S^{(t)} \mid \boldsymbol{o}^{(1:T)}) &= \frac{1}{Z} \times P(S^{(t)} \mid \boldsymbol{o}^{(1:t)}) P(\boldsymbol{o}^{(t+1:T)} \mid S^{(t)}) \\ &= \frac{1}{Z} \times f^{(1:t)} \times b^{(t+1:T)}. \end{aligned} \tag{2.9}$$

Here Z is again a normalisation factor, $f^{(1:t)}$ is is the previously introduced forward pass with $f^{(1:0)} = \boldsymbol{\pi}$, and $b^{(t+1:T)}$ denotes the *backward pass*

$$\begin{aligned} b^{(t+1:T)} &= P(\boldsymbol{o}^{(t+1:T)} \mid S^{(t)}) \\ &= \sum_{s^{(t+1)}} P(s^{(t+1)} \mid S^{(t)}) P(\boldsymbol{o}^{(t+1)} \mid s^{(t+1)}) b^{(t+2:T)}, \end{aligned} \tag{2.10}$$

where $b^{(T+1:T)} = \mathbf{1}$. Note here, that a backward pass does not express a probability distribution, as is indicated with the initialisation that does not sum to one.

Smoothing is related to the filtering task. When applying the forward-backward algorithm to perform smoothing, we want to compute a distribution over all hidden states for a point t laying in the past, given observations up to a later point T in time.

Prediction is a task which is, to some degree, an extension of the filtering task. We can compute $P(S^{(t+k)} \mid \boldsymbol{o}^{(1:t)})$ with $t + k > t$ by initialising $p^{(t+0)} = f^{(1:t)}$, and then propagating forward [15]

$$\begin{aligned} p^{(t+k+1)} &= P(S^{(t+k+1)} \mid \boldsymbol{o}^{(1:t)}) \\ &= \sum_{s^{(t+k)}} P(S^{(t+k+1)} \mid s^{(t+k)}) p^{(t+k)}. \end{aligned} \tag{2.11}$$

Decoding computes the most likely trajectory of the system based on given evidence, i.e. $\arg\max_{s^{(0:T)}} P(s^{(0:T)} \mid \boldsymbol{o}^{(1:T)})$ [15]. *Viterbi* addressed the problem of finding this trajectory by using [27]

$$v^{(1:0)} = \boldsymbol{\pi},$$

$$v_i^{(1:t+1)} = \max_j v_j^{(1:t)} P(s_i^{(t+1)} \mid s_j^{(t)}) P(\boldsymbol{o}^{(t+1)} \mid s_i^{(t+1)}), \tag{2.12}$$

$$bp_i^{(t+1)} = \arg\max_j v_j^{(1:t)} P(s_i^{(t+1)} \mid s_j^{(t)}). \tag{2.13}$$

Here, s_i denotes the i^{th} value in the domain of state variable S. In order to terminate, one can then use the back-pointers $bp^{(t)}$ in order to trace the path back from the best final state, i.e.

$$s^{(T)*} = \arg\max_i v_i^{(T)}. \tag{2.14}$$

2.3.9 Learning HMM Parameters with Expectation-Maximisation

Often, the parameters $\theta = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ characterising a HMM are not directly available. If we want to create a model that describes real phenomena well, we need to estimate and optimize these parameters [26]. In other words, for $\mathcal{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots\}$, we want to maximise $P(\mathcal{O} \mid \theta)$, i.e. the probability that a sequence of observations was produced by a HMM with parameters θ [27]. Here, $\mathcal{O}$ denotes a sequence of real observations. By adapting θ so that we maximise $P(\mathcal{O} \mid \theta)$, we simultaneously maximise the probability of our model producing sequence $\mathcal{O}$.

Estimating HMM parameters is often being referred to as *training* the model [27]. As there is no optimal way to optimise parameters θ, θ is chosen so that $P(\mathcal{O} \mid \theta)$ reaches a local maximum [26]. This local optimum is usually found via expectation-maximisation methods, in particular with the incremental *Baum-Welch* algorithm. Another option would be to use gradient-based techniques. When applying Baum-Welch, one computes expected values based on the current model, and takes these to re-estimate a set of new parameters. This is done iteratively, until convergence.

2.3.10 Evaluating the Learning Performance

Learning HMM parameters can also be seen as choosing one particular model out of a set of candidate HMMs [15]. This set of possible models is the *hypothesis space*. By learning from real data $\mathcal{D}$, we want to find a model, specified by its parameters, that perfectly describes a distribution P^*. P^* here is the true distribution the data is sampled from. A lack of sufficient data often prevents us from finding the real, underlying distribution P^* in practice, which is why we look for a model that approximates P^* best. To be able to say what is a reasonable approximation, one has to choose a goal for learning, in the form of some function that needs to be minimised.

In section 2.3.9, we tried to approximate the true distribution P^* with a distribution $\tilde{P}$ associated with our model. More generally however, we can also view learning as an optimisation problem. Given an *objective function*, we want to find the model with the best score within our hypothesis space. The objective or loss function here provides us with a numeric preference for different models, and would ideally be a function of the true distribution P^*. As P^* is not available in general, it is often estimated by the empirical distribution $\hat{P}_{\mathcal{D}}$. $\hat{P}_{\mathcal{D}}$ is based on the real data $\mathcal{D}$, and approaches the real distribution P^* as the number of samples in $\mathcal{D}$ grows. If we optimise the loss function relative to the empirical distribution $\hat{P}_{\mathcal{D}}$ as opposed to the true distribution P^*, therefore minimising the expected loss or *risk*, we talk about empirical risk minimisation.

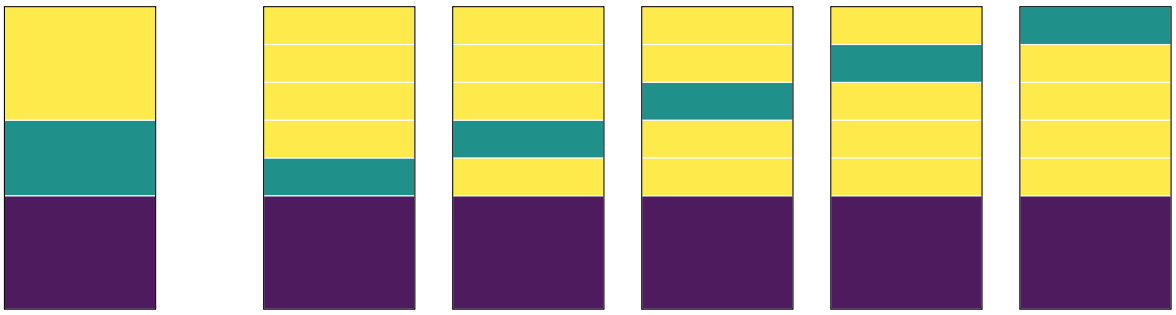
Disregarding the true risk completely when minimising the empirical risk can result in difficulties. The available dataset $\mathcal{D}$ is often too small to contain relevant samples of all events in an environment, leading to a poor estimate of the true underlying distribution P^*. Nevertheless, after learning model parameters, we want to reason about new events that were not necessarily part of the seen data $\mathcal{D}$. If we train our model so that it perfectly captures $\hat{P}_{\mathcal{D}}$, we can minimise the empirical loss. However, reasoning about an event unseen in the training data $\mathcal{D}$ can then result in poor estimates with respect to the true underlying probability of the event. We therefore want a model that *generalises*, such that it can also perform well on data unseen in the training dataset. Whenever our model is trained so that it can explain training data $\mathcal{D}$ without it being able to generalise for unseen data, it is said to *overfit*.

To determine e.g. whether or not our model overfits, various techniques of evaluating the performance on unseen data exist. One possibility is applying *holdout testing*, where only a part of the data, the training-set, is used for updating model parameters. The unused part, also denoted as test-set, can then be used to test how well the learned model performs on unseen data. By holding out this test data, we are able to get an unbiased empirical estimate of the risk.

Choosing the size of training- and test-set for a performance evaluation often proves difficult. Increasing the number of samples in the training-set results in a better trained model, but in a less accurate estimate of the performance. On the other hand, increasing the size of the test-set gives us better performance estimates, but decreases the quality of our learned model. A small number of overall training samples in dataset $\mathcal{D}$ therefore makes it even more difficult, to determine a fitting size of training- and test-set. An approach that should help to overcome this issue, is *k-fold cross validation*.

Cross validation is an approach that uses all available data $\mathcal{D}$ for training and testing our model. As we do not want to bias our performance measure by testing on samples also used for training, training- and test-sets have to be kept disjoint. We can do so by applying holdout testing multiple times, and average over resulting estimates. More precisely, when applying k-fold cross validation, we first split the data into k disjoint and equally sized sets. In each iteration of this method we use one partition as test-set, and all remaining partitions as training-set. The obtained estimates can then be combined. If we set $k = |\mathcal{D}|$, i.e. every sample corresponds to a separate partition, we talk about *leave-one-out cross validation*.

Figure 2.8 shows possible partitions of holdout testing compared to 5-fold cross validation. The images additionally show a third type of dataset. Test-sets $\mathcal{D}_{test}$ as we discussed so far are typically used to determine the performance of a final model. However, we might also want to compare the performance of different techniques. One way to do this is to learn various models with the help of training-set $\mathcal{D}_{train}$, and compare the performance estimates obtained with $\mathcal{D}_{test}$. The problem with this approach is that we tend to optimise the final model based on the test set. Therefore, the obtained performance measures are too optimistic. To this end, we would need yet another set of data that has not been used to make any decisions whatsoever. Therefore, the actual test set is locked away until a final model is available and a part of the left-over training data is again kept apart for comparing different techniques. This training-test data is



(a) Holdout testing (b) 5-fold cross validation

Figure 2.8: Possible partitions of available data $\mathcal{D}$. The darkest, purple, area corresponds to the test-set, necessary for achieving an unbiased performance estimate. The lighter, turquoise areas denote the validation-sets, which can be used to determine the performance of different parameters. The lightest, yellow, regions are training-sets used to train the model.

often called the validation data.

3 Related Work

3.1 Extraction of Stay-Points and Significant Locations

Yang et al. extract so-called stay-locations from cell phone records [31]. The temporal order of records as well as small distances between them, determine stay-points. They are characterised by a location, start-time, and expected, maximum and minimum duration.

Zhou et al. talk in their location tracking approach via Received Signal Strength (RSS) about so-called location points [35]. A location point consists of similar RSS samples, recorded with small RSS distance within a small time interval. Location points at which a person stays for a long time are then referred to as *personal common locations*, denoting e.g. restaurants or labs and therefore corresponding to our notion of significant places.

Instead of extracting stay-points as a first step, Mahbub and Chellappa perform clustering on the raw location data [19]. DBSCAN based on geographical distances, is applied to find clusters characterised by the latitude and longitude of the centre point, as well as the radius in which all remaining points lie within. To capture data-points that were obtained while the user was travelling or just at some insignificant place, two types of dummy clusters were added.

Yang et al. propose a variant of the DBSCAN algorithm for an extraction of significant places [30]. In order to find stay-points, speed and density conditions are used as opposed to common distance and time limits. After raw locations are condensed to stay-points, significant locations are obtained as in this thesis via DBSCAN.

The extraction of *key locations* from anonymised CDRs is a topic addressed by Isaacman et al. [13]. In a first stage, cell towers in a location trace are spatially clustered. Here, a location trace consists of multiple CDR entries, or so-called mobile network events. In a second stage, the importance of each cluster is determined. This is done using a logistic regression model. Isaacman et al. additionally propose a method based on logistic regression that estimates the key locations *Home* and *Work*, given a list of important cell tower clusters.

Gambs, Killijian and Prado Cortez discuss *Points of Interest* as their fundamental building block in next place prediction [10]. The DJ-Cluster algorithm (cf. Section 2.2.4) is applied to discover the points of interest by first removing non-static or redundant location traces. Subsequently, clusters are formed according to a radius and a minimum number of data-points, and in a final stage clusters with common traces are merged.

Using the term semantic, rather than significant location, Ying et al. emphasise the benefits of using semantic labels as an addition to geographic trajectories [32]. Similar or equal semantic trajectories can have profoundly different geographic location traces.

On the other hand, similar geographic trajectories can have various possible semantic location sequences. After applying various algorithms for stay-point extraction and density-based clustering for finding locations, every geographic trajectory is translated into a sequence of significant locations. Finally, semantic labels obtained from a spatial database are assigned to each of these locations.

To reduce the computational effort of processing raw location data, this thesis follows the proposition of Li et al. for the extraction of stay-points based on a distance and time threshold [16]. Their approach is not only based on data with similar structure, but also very intuitive. When these condensed location points are computed, we take the work of Yang et al. as an example and apply DBSCAN in order to find significant locations of a user [30]. The design of DBSCAN for spatial data is well suited for our task, and provides benefits such as an arbitrary amount of clusters. In addition to that, implementations of DBSCAN are available in common clustering libraries.

3.2 Next Place Prediction

Mining and processing human trajectories is nothing new. Together with a multitude of purposes, different algorithms have been applied for predicting where people might go next. Some of the most notable machine learning models that have been used for next place prediction are Bayesian models, neural networks and clustering methods [25]. However, also pattern matching algorithms and state-based techniques have been applied. The latter contain, among others, hidden or regular Markov models. In what follows, we summarise different approaches to next place prediction for the most important groups of models.

3.2.1 Markov Models

Markov models in all their variations are used frequently to find possible future locations of a user. Mathew, Raposo and Martins first cluster location histories based on time [20]. Thereafter, one HMM is trained for every cluster, such that one model corresponds to a specific type of location history. In order to obtain a prediction for the next location, the cluster most likely associated with the trajectory thus far is chosen, and the corresponding HMM is used to find the most probable following location.

Gambs, Killijian and Prado Cortez propose a Mobility Markov Chain (MMC) for next place prediction [10]. More precisely, they work with an $n-$MMC, in which the states of the model correspond to n significant locations. In addition to a set of states, a MMC includes transitions obtained from labelled mobility traces, describing the probability of moving from one state to another. The predicted next place is derived by taking the n previously visited significant locations, and finding the most probable transition to a next state.

After extracting significant locations from historical trajectories, future places are predicted via a statistical model in the work of Yang et al. [30]. They propose a *variable order Markov model*, and use partial matching for the prediction task.

Qiao et al. propose a hybrid Markov-based model for mobility prediction [25]. The input of the prediction algorithm includes data obtained from a cellular network, consisting of a user-identifier, a time-stamp and city-hotspot identifiers. Before the next location is predicted, typical mobility sequences of a user are used to determine the order of the Markov predictor. Furthermore, Qiao et al. adjust the predictions of their approach based on other users with similar mobility patterns.

Also in the approach presented in this thesis a Markov model will be applied to predict next locations. More precisely, we use a HMM with states based on previously extracted significant locations.

3.2.2 Artificial Neural Networks

Despite the fact the Artificial Neural Network (ANN) has been gaining popularity the last decade, its application for next place prediction is rather limited. Akoush and Sameh predict future locations via a hybrid Bayesian Neural Network (NN) in order to manage network resources more efficiently [1].

The proposed model allows Bayesian inference as its output consists of a probability distribution, describing the uncertainty of a prediction [1]. It has one single hidden layer with 15-25 hidden neurons, and its input comprises data about current and previous cell towers, as well as temporal information. Integrations which are necessary for Bayesian learning are approximated with the help of Markov Chain Monte Carlo (MCMC) methods.

De Brébisson et al. predict future locations of taxis based on a sequence of initial, partial GPS trajectories and meta-data [6]. The ANN model that is proposed has a single hidden layer, consisting of 500 hidden Rectifier Linear Units. Its output is a tuple of latitude and longitude, computed as a weighted average of predefined location cluster centres. In addition to this, De Brébisson et al. conducted experiments with several, more recent, models. The results obtained by applying a standard Recurrent Neural Network (RNN), a Bidirectional RNN or a so-called Memory Network however, could not match those achieved by the simpler ANN model.

3.2.3 Other Methods

There are plenty of alternative methods for next-place prediction. In addition to that, various approaches exist which compare or even combine different techniques. Ying et al. use geographic as well as semantic information for predicting the next location of a user [32]. After computing candidate trajectories based on geographic scores, predictions are adjusted with the help of semantic scores and the most probable path is chosen. This, in turn, leads to a prediction of the next location of a user. To realise their approach of next location prediction, Ying et al. propose adapted prefix-trees that represent semantic trajectory patterns.

Providing improved services for users of context-aware mobile applications is the main motivation behind the proposed work of Anagnostopoulos, Anagnostopoulos and Hadjiefthymiades [2]. They approach next place prediction as a supervised classification

task, by first applying K-Nearest Neighbours (kNN) and *decision trees* to classify trajectories of a user, and providing a knowledge base for the *location predictor*. The most likely future symbolic location is either found by predicting a mobile cell directly, or a direction in which the user is moving.

Chon et al. look at different mobility models and their suitability for future location prediction [5]. In particular, a location-dependent Markov model is compared to a location-independent *NextPlace* model, which applies non-linear time series analysis for predicting locations. Furthermore, various schemes that determine the information extraction from data, among others, are put to the test.

Lv et al. propose a slot-based next place prediction model, and divide a day into 288 slots of 5 minutes [17]. Given a triplet containing the location of a base station, the time-slot of arrival and the number of slots describing a stay, not only the next place is predicted, but also the departure slot, i.e. when the transition to the next place will occur. The temporal behaviour of a user is once more predicted with a variant of the *NextPlace* model or a Markov-based model. For the prediction of the next location, two variations of Markov models are applied.

4 Location Prediction

Reasoning about current or future locations of a person is often done with the help of Markov models (cf. section 3.2). A class of relatively simple models that nevertheless perform well for a multitude of tasks, is the class of HMMs [15]. Particularly interesting is therefore the application of HMM to perform next place prediction. The simplicity behind a HMM, however, also comes with restrictions and limitations. The discrete state-space of HMMs (cf. section 2.3.7), as an example, prevents us from being able to predict continuous coordinates as our next places. As we still want to look at next place prediction with the class of HMMs however, we do not circumvent this issue by using models with a continuous state-space. Instead, we discretise the raw coordinates, about which we want to reason, before we can use HMMs.

Preparing raw data for its usage is done in two steps, namely the extraction of stay-points and significant places respectively. This form of pre-processing provides us with a discrete representation of the raw, continuous, location data we want to work with. Thereafter, we can use the pre-processed data to train a HMM that allows us to predict the next, most probable place a user will be at. A possible structure of this model for next place prediction is the third and major topic.

All illustrations of clustering algorithms, applied during the pre-processing of data in what follows, originate from results obtained with the help of the *scikit-learn* library for Python [23], unless stated otherwise.

4.1 Extracting Stay-Points

To obtain a discrete representation of continuous location data, we want to determine locations that are significant to a user. Before significant places can be extracted, however, it is useful to group raw location data in stay-points. A stay-point contains a temporal factor next to location information, ensuring we later on solely consider locations a users stays at for a certain amount of time. The extraction of stay-points as opposed to clustering raw location data avoids issues in finding significant places where sampling is more sparse, e.g. in buildings [16]. After all, most cluster algorithms require a high density of points around a centre point to detect the presence of clusters. Regions where a user often passes by, on the other hand, could be falsely considered meaningful for the clustering algorithm without a stay-point extraction. In addition to this, the smaller input space after extracting stay-points reduces the computational effort of clustering noticeably.

Li et al. propose a way to extract stay-points from GPS data [16]. Figure 4.1 illustrates their notion of a stay-point, obtained from a list of logs containing coordinates, described

Figure 4.1: GPS entries and types of stay-points

by a time-stamp, latitude and longitude.

A stay-point can be thought of as a geographic region where a user remains for a while [16]. Therefore, it can be argued that stay-points carry semantic meaning. Figure 4.1 depicts two different kinds of stay-points. The first kind, stay-point 1, corresponds to a particular log-entry. The user here stayed at the respective location for a time-span longer than a certain timing threshold. The second type, i.e. stay-point 2, describes the case in which a user did not remain stationary, but rather moved within a relatively small radius for a certain period. While the first kind of stay-point is located at the respective location of the log-entry, the coordinates of the second are computed by averaging over latitude and longitude of all contributing GPS points.

In our approach we apply the algorithm proposed by Li et al. to extract stay-points. As later experiments will show, however, we do not restrict the data to solely originate from GPS devices. In order to compute stay-points from the raw location data, the algorithm requires two threshold parameters. The first parameter is a spatial threshold that specifies the radius in which points need to lie to be considered as one location. The second parameter is the minimal time-span the user needs to be in the same location.

4.1.1 The Implementation

Algorithm 1 depicts pseudo-code, describing the core idea of the stay-point extraction method proposed by Li et al. Input data P is the base of our stay-points, namely a sequence of temporal locations, i.e. locations with their time of recording. As stated previously, the algorithm additionally requires a spatial and temporal threshold. These thresholds control the number, as well as the specific location of a stay-point we extract.

The procedure in algorithm 1 begins with computing spatial, as well as temporal distances between a point in the input data P, and its temporal successors. As long as we remain below the distance and timing thresholds, the user stayed within close proximity for a period of time, i.e. the points contribute to a stay-point. If we surpass the thresholds however, we reach the border of a stay-point. In this case, we compute the mean coordinate of previously discovered stay-point, as well as arrival and departure time, and we store the stay-point in a list of stay-points. Thereafter, the first point across the border is used in the search for subsequent stay-points.

The log-files of different users tend to have largely varying scales, which might require

Algorithm 1 Stay-Point Extraction; Adapted from [16]

```
1:  procedure STAYPOINT_DETECTION(P, dist_thres, time_thres)
2:  # Input: Input data P, distance threshold dist_thres
3:    and time span threshold time_thres
4:  # Output: set of stay-points SP
5:
6:      i ← 0, pointnum ←| P |
7:      while i < pointnum do
8:          j ← i + 1
9:          while j < pointnum do
10:             dist ← DISTANCE(p_i, p_j)          # Compute distance between two points
11:             if dist > dist_thres then
12:                 delta ← p_j.t − p_i.t          # Compute time span between two points
13:                 if delta > time_thres then
14:                     s.coord ← MEANCOORDINATE({p_k | i ≤ k < j})
15:                     s.arrival_t ← p_i.t, s.depart_t ← p_j.t
16:                     SP.INSERT(s)
17:                 break
18:             j ← j + 1
19:         i ← j
20:     return SP
```

different parameters for a good stay-point extraction. Therefore, we chose to normalise the coordinates within a log-file optionally before applying the original algorithm (cf. algorithm 1).

Figure 4.2 visualises a trajectory with non-normalised coordinates on the left-hand side. The coordinate of a single point in the trajectory is a latitude-longitude pair in radians. The same trajectory with normalised coordinates is given to the right. As the image illustrates, the only visible change is the labelling of the two axes. In order to achieve this, latitude and longitude are not normalised separately; instead, the normalisation makes sure to modify latitude and longitude to lie between zero and one, while simultaneously maintaining the ratios of different data-points.

4.2 Finding Significant Locations

As HMMs require a discrete state-space (cf. section 2.3), continuous geographic coordinates cannot be used directly for next place prediction. Extracting significant locations from raw location data provides us with a discretisation of the input data. In section 4.1, we discussed the advantages of performing this extraction on previously determined stay-points, as opposed to raw coordinates.

As stated before (cf. section 3.1), multiple stay-points make up one significant place. Intuitively, this means that we tend to spend a lot of time at restricted areas that are

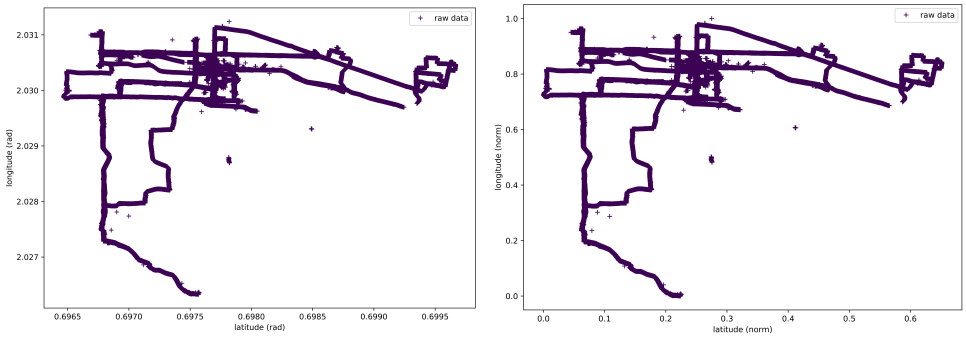


Figure 4.2: Left-hand side: Coordinates before normalisation. Right-hand side: Coordinates after normalisation. Trajectory originates from the Geolife dataset (cf. section 5.1)

meaningful to us, such as our home. When extracting stay-points, we take the factor time into account. This enables us to find locations where a person spends at least a certain amount of time. In other words, the extraction of stay-points is a process that filters out places a user does not spend a lot of time at. Therefore we can discard the temporal factor and focus on finding the geographical location of significant places.

In order to find locations that are meaningful to a user, we subsequently group or *cluster* stay-points to retrieve significant places. Section 2.2 provides us with the necessary tools for this task. Before the stay-points can be clustered, however, we need to find the best clustering algorithm for this task.

4.2.1 Choosing a Clustering Algorithm

To describe significant places, we will use discrete labels, e.g. home, office, etc. As opposed to their underlying, true, locations, a lot of these labels might overlap for a vast number of users. Nevertheless, significant places can differ in type and number from one user to another, as illustrated in Figure 4.3. Meaningful locations are therefore determined for every user separately, instead of using data of multiple users simultaneously for the extraction of significant places.

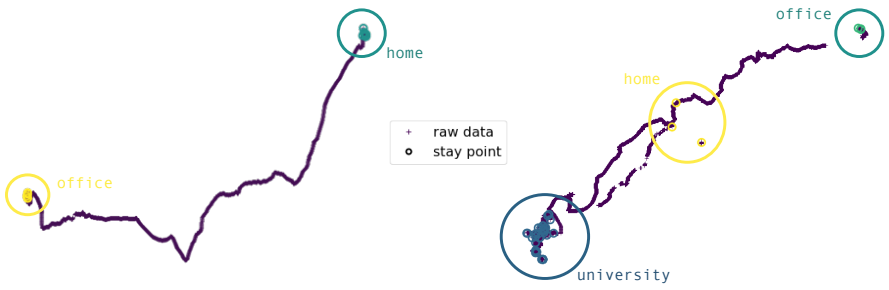
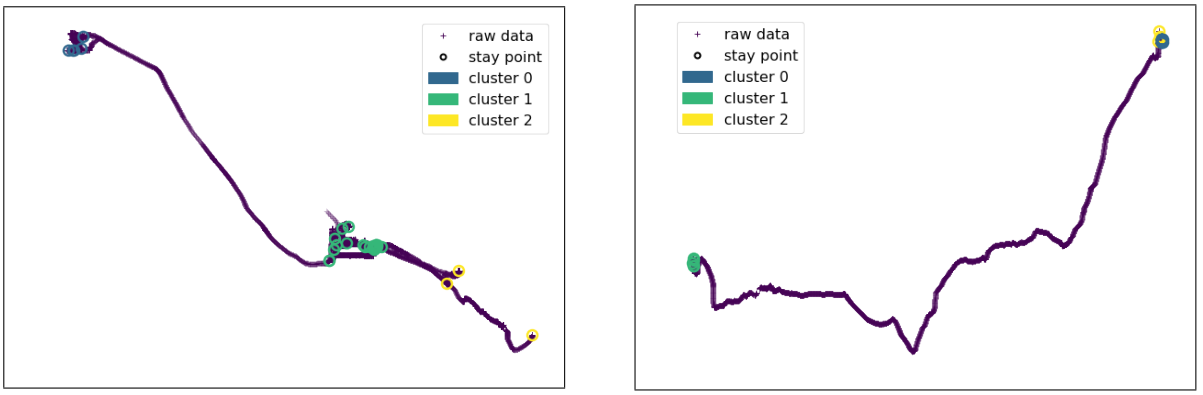


Figure 4.3: Two users with different numbers of possible significant places. Chosen labels are examples; Trajectories originate from the Geolife dataset (cf. section 5.1)

Keeping the previous considerations, about the varying number of significant places and underlying stay-points, in mind, we look for a clustering algorithm suited for the task of finding meaningful locations. If we examine the basic methods presented in section 2.2, two main issues emerge.

The first issue is that most common clustering algorithms are only able to find a predefined number of clusters. The name-defining characteristic of the K-means clustering algorithm is the predefined number of clusters k. Figure 4.4a shows an example for which a reasonable k has been chosen. The overall grouping of stay-points appears reasonable to the human eye. Only for the lower right cluster, it could be argued that the data points lie rather far away.



(a) Suited number of clusters

(b) Unsuited number of clusters

Figure 4.4: Extraction of significant places with K-means and $k = 3$. Exemplary trajectories originate from Geolife dataset (cf. section 5.1)

Oppositely, Figure 4.4b shows a second trajectory, where it looks very much like if there would be two significant places. However, K-means is forced to find three point-groups, since k has been set to three. Because k is bigger than the actual number of clusters, some of these clusters will be split up.

Adjusting the predefined number of clusters k could avoid this kind of issue for the second trajectory. This, however, would require finding a suitable number of significant locations and therefore distinct clustering parameters for every single user, thus introducing a new issue. Even for relatively small datasets, a manual estimation of meaningful places would demand extensive additional data pre-processing. What we aim for instead, is a clustering algorithm that is able to find a suitable number of clusters by itself.

The second issue with many of the commonly used clustering algorithms, is that each point must be assigned to a cluster. Since not all stay-points correspond to a significant location, this is actually problematic. Whenever a person remains, for a longer period of time, within a small region that does not carry any significant meaning, a corresponding stay-point will be extracted as soon as the spatio-temporal conditions are satisfied. A specific example could be stopping during a walk, in order to talk to a person. Nevertheless, stay-points that result from events like this, do not contribute to a significant location, and should be treated as noise.

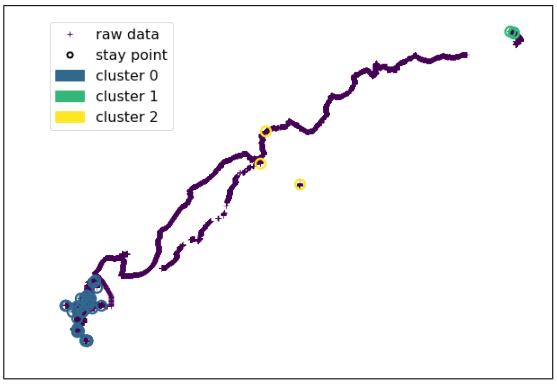
Figure 4.5 shows results obtained with various types of clustering algorithms. In their traditional implementation, they can come across issues with stay-points that represent noise. The images on the left-hand side once again illustrate clustering results that correspond to our intuition. The trajectories on the left in Figure 4.5 consist of three well-separated sets of stay-points. When clustering with a three-component GMM, the three corresponding significant locations can be detected clearly. Agglomerative hierarchical clustering and affinity propagation (cf. section 2.2.5) lead to equivalent assignments of clusters. Note that a permutation of the cluster numbers does not affect the results in any way. After all, the purpose of clustering is to find groups of points that belong together, not to assign labels to the data.

The right-hand side of Figure 4.5, on the other hand, demonstrates how these algorithms struggle to handle isolated stay-points that do not belong to significant places. Both the three-component GMM and affinity propagation are unable to find different clusters. The outlier stay-points disable the capability of these algorithms to distinguish between any significant location in this case. Agglomerative hierarchical clustering, on the other hand, is capable of distinguishing multiple clusters. Different hierarchical levels, each corresponding to a possible clustering result, allow us to choose a cluster assignment with e.g. three significant places. Nevertheless, the three clusters seem to be larger than necessary, since isolated points are also assigned to the nearest cluster.

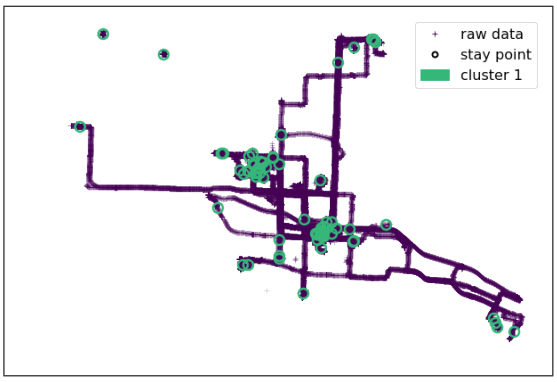
In all three clusters obtained from agglomerative clustering, the majority of stay-points is located relatively close to each other, i.e. with high spatial density. Nevertheless, two of these three clusters also contain stay-points that lie isolated from the remaining points. These stay-points, which occur by themselves, indicate that the user does not frequent these positions. Before labelling a stay-point as noise however, it should be assured that the geographic distance to other fixed points is sufficiently large.

To circumvent both of these issues in the detection of significant places, we need a more sophisticated approach. Concretely, our clustering algorithm should be able to detect the number of clusters from the data and support noisy data points. We want to look for a better suited clustering algorithm, that ideally allows us to circumvent these issues. As shown in previous section 3.1, the density-based DBSCAN method has been applied for the task before (cf. [19]). As the algorithm was developed for clustering spatial data [7], it offers solutions to our difficulties by design. Next to the consideration of noise, DBSCAN does not need a predefined number of clusters, and is built to find clusters with arbitrary shape.

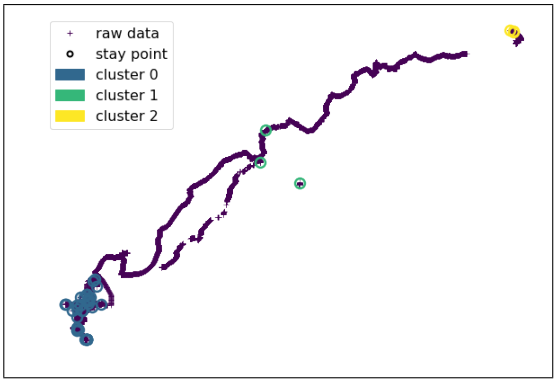
Figure 4.6 illustrates the result of applying DBSCAN to the trajectory with noisy stay-points that presents the GMM, agglomerative clustering as well as affinity propagation with a challenge. If we cluster stay-points of this trajectory with DBSCAN, we do not need to define the number of clusters to be extracted on beforehand. Instead, we have to provide a minimum number of points and a radius ϵ for the clusters that are to be found by the clustering algorithm. With parameters $\epsilon = 0.05$ and $min\_points = 5$, DBSCAN discovers three different clusters and therefore significant locations. Compared to the results obtained previously by the agglomerative hierarchical clustering approach, some deviations become clear. Most notably, the additional set of stay-points which now is not assigned to any of the clusters, and thus represents noise.



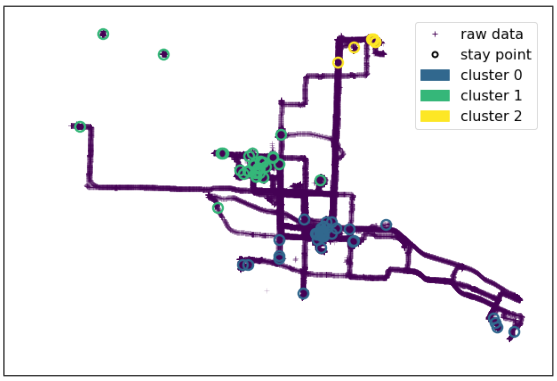
(a) Method: GMM with *components* = 3; little noticeable noise in stay-points



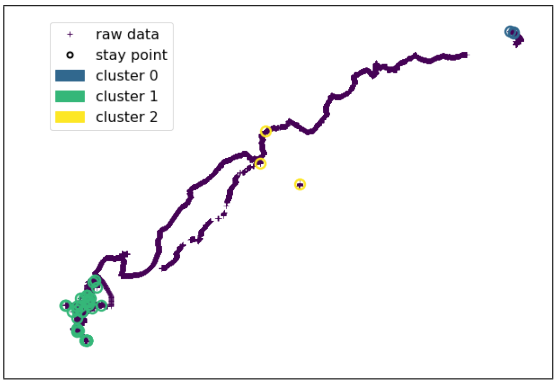
(b) Method: GMM with *components* = 3; noticeable noise in stay-points



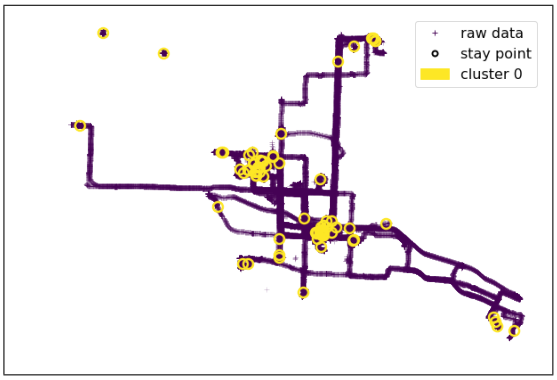
(c) Method: Agglomerative clustering, *clusters* = 3, *linkage* = *ward*; little noticeable noise in stay-points



(d) Method: Agglomerative clustering, *clusters* = 3, *linkage* = *ward*; noticeable noise in stay-points



(e) Method: Affinity propagation, *affinity* = *euclidean*; little noticeable noise in stay-points



(f) Method: Affinity propagation, *affinity* = *euclidean*; noticeable noise in stay-points

Figure 4.5: Extraction of significant places with various clustering algorithms. Exemplary trajectories originate from the Geolife dataset (cf. section 5.1)

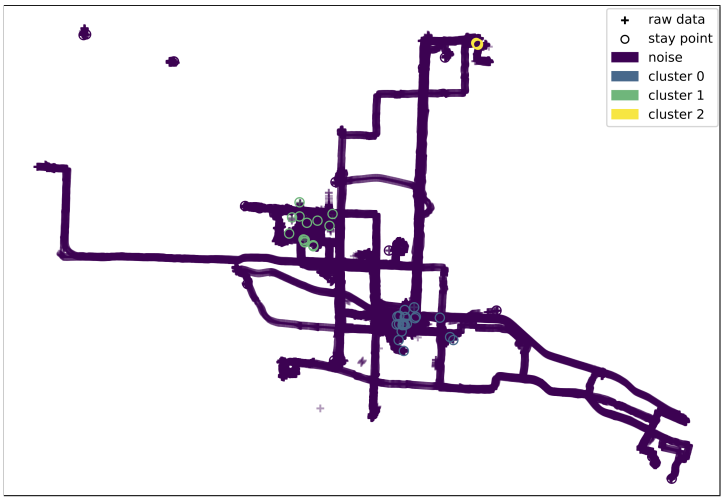


Figure 4.6: Extraction of significant places with DBSCAN, where $\epsilon = 0.05$ and $min\_points = 5$; Trajectory originates from the Geolife dataset (cf. section 5.1)

4.2.2 Applying DBSCAN for Significant Location Detection

Previous considerations indicate the suitability of applying DBSCAN for the detection of significant places in this thesis. The design of this algorithm manages to circumvent main discussed issues of common clustering algorithms with spatial data. Whenever we try to extract significant locations based on previously determined stay-points in the remainder of this thesis, we will apply the implementation of DBSCAN provided by *scikit-learn* [23]. As opposed to other, more recent density-based clustering approaches (cf. section 2.2.4), DBSCAN is thus already part of commonly used libraries. The two parameters of DBSCAN, i.e. a radius and minimum number of points to form a cluster, are chosen based on subsequent experiments presented in section 5.3.

4.3 Predicting Next Locations

Predicting the next location of a user is done with the help of a HMM. The application of these and other models, which assume the Markov property to hold, is well represented in next place prediction literature (cf. section 3.2). The main goal of the next place prediction is to learn a model that encodes the whereabouts of a person. This model can then be used as a basis for determining how likely it is for that person to be detected at some position. In the future, similarities or discrepancies between predicted and sensed locations of a person could be used to make security-based decisions, e.g. in the course of authentication processes.

4.3.1 The HMM Structure

In section 2.3, we found that three components fully specify a HMM. These are the initial state probabilities, as well as the state transition and observation model. All three components contain likelihoods associated with the state and observation variables, which are the structural building blocks of a HMM. The state variables are latent, which means that the current state of a dynamic system cannot be directly observed. In order to reason about the true state of a system, observable values have to be used instead. For our task of next location prediction, the state of the system corresponds to the true, but not directly observable location of a person. Sensor output, consisting e.g. of measured coordinates, are possible observations of the system.

As stated previously, the state space of a HMM is discrete. This means that the next place prediction of our model must be one of a discrete set of location labels. The set of possible locations here corresponds to the extracted significant places, which we determined during pre-processing (cf. section 4.2). Therefore, we chose the number of states to be equal to the number of extracted significant places for our model. We also added one additional state for data-samples that were not assigned to a particular significant place, i.e. noise.

In contrast to the discrete nature of states in a HMM, observations can originate from discrete, as well as continuous distributions (cf. 2.3.7). In the dynamic environment our system is acting in, observations correspond, as the name suggests, to everything we can observe. In particular, everything that can be observed with the help of sensors that are able to capture location information. Naturally, these sensors are heterogeneous, and differ e.g. in the sampling rates in which new observations are sensed. Another example would be the heterogeneity between stationary biometric sensors, which do not sense continuous location traces, but offer a high confidence in detecting a person at a specific place; and GPS sensors, which observe location traces, but might not exhibit as high of a confidence in its records. For now, we want to neglect this heterogeneity, and take a specific look at what all these sensors can provide.

Continuous observations obtained from a sensor that captures location information, could be a coordinate consisting of a latitude-longitude pair, equipped with the time-stamp describing the exact point in time at which the location was recorded. Also the altitude could be a possible continuous observation that would be easily retrievable by a lot of sensors. Since the stay-point extraction and thus the detection process of significant places does not differentiate between the height of different locations, however, this factor is neglected in this thesis.

Applying a HMM with discrete observations for the task of next place prediction on the other hand, requires a discretisation of continuous sensor measurements. One possible way to discretise coordinates is the detection of significant locations, as shown in our pre-processing step (cf. section 4.2).

In order to discretise time-stamps, which encapsulate a date as well as the time of day, we can think of different granularities. As people tend to re-visit places with a temporal periodicity (cf. [5]), extracting the day within a week could be one possibility. With coarser granularity, we could also determine whether the time-stamp belongs to a

day on the week-end, or not. Considering the time, we could divide the day in different time slots with various granularities, e.g. night vs. day, hours, etc. Table 4.1 shows a summary of continuous, as well as discrete observations, that have been considered for the task of training a model for next place prediction.

continuous	discrete
time-stamp	day of the week
latitude	weekday/weekend
longitude	hour of day
altitude	hour slots
...	location tags
	...

Table 4.1: Possible continuous and discrete observations for our HMM

As stated in section 2.3.7, the observation model of a HMM is typically a continuous probability density function in the case of continuous observation variables. How these functions look like, can strongly depend on the kind of continuous observation we work with. Revisiting the possible continuous observations in table 4.1, it seems reasonable to assume that the latitude and longitude of the raw data points that make up a significant location are distributed according to a Gaussian or normal distribution. One single state, corresponding to a particular significant place, is then described by observing the mean latitude or longitude on average, with a certain variance. This does not hold for every other sort of observation, however. Time stamps, for example, do not stay close to their mean value as measurements do not occur at the same time. The probability density function, which maps continuous observations to discrete states, therefore has to be chosen strongly depending on how the continuous observations look like.

One such continuous probability density function that has already been discussed, is a mixture of Gaussians, which is also used in GMM for clustering. More precisely, GMM-HMMs use GMMs to model observations and their probabilities. Since we are able to cluster, and hence discretise, the data better than with GMMs, however, we can focus on using discrete observation variables in our HMMs. The observation model of a HMM with a discrete observation variable can be thought of as a matrix that contains the probability of observing a certain discrete observation value in a state, for each state of our model. In the dynamic environment our system is acting in, we can choose the observation variable to be the day of the week, as an example. The observation model then describes how likely it is that we observe that it is e.g. Monday, while we are currently at a certain location, for every significant place of a user.

A schematic HMM for next place prediction with three states, i.e. two significant places and one state for noise, is illustrated in Figure 4.7. The arrows between states visualise state transitions, which are assigned probabilities by means of the state transition model. On the other hand, arrows pointing from states to observations are described in the observation model, which encodes the probability of making an observation when the system is in a specific state. With this basic structure for our HMM in place, we can

focus on the granularity of the observation variables and the unknown probabilities in the model. Concretely, we still need to learn probabilities for the observation and state transition models as well as find the correct granularity for discretising time.

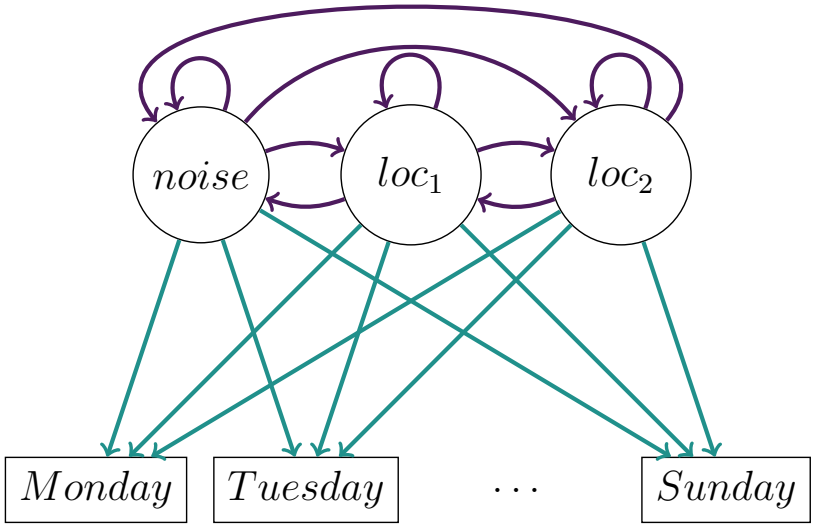


Figure 4.7: Exemplary 3-state HMM. Discrete observations correspond to days in the week. Upper, purple, arrows describe state transitions; lower, green, arrows possible observations in a state.

4.3.2 Learning HMM Parameters and Next Place Prediction

Apart from the structure of our HMM, as discussed in section 4.3.1, we also need probabilities that are encoded in a state transition and observation model to do next place prediction. These probabilities are the parameters of a HMM and define its behaviour, allowing us to shape HMMs with a desired behaviour by adapting the probabilities accordingly. Because it is impossible to specify HMM parameters by hand, we need to rely on available data and a training procedure to have our HMM exhibit the expected functionality.

The basic approach to learn HMM parameters was introduced in section 2.3. The main idea is to change the parameters so that it becomes more likely for the data to originate from our model. For the HMM we are training, this sequence consists of discrete observations, such as location labels, the day of the week, or combinations thereof. For each observation, we also have the correct location, which will be our target label, available. This target is the significant place we want our model to predict as a next place, corresponding to the location label of the next time-step. Due to the availability of target labels, the HMM is trained in a supervised fashion. The main difference, as opposed to learning approaches such as the Baum-Welch algorithm, lies in the computation of the probability that we try to maximise. The defined target labels

influence the probability of our model in a positive way if we manage to perform correct predictions, and in a negative way otherwise.

After the HMM is trained, i.e. the model parameters have been learned, we can apply the HMM for next place prediction. Section 2.3.8 provides us with common inference tasks, which can be used to find the most likely state that our model is in, given observations thus far. In particular, we use the forward-backward algorithm to obtain a sequence of most likely states, based on every observation we give to the model.

4.3.3 Implementation and Choosing the Final Model

While previous sections both covered the main structure of our HMM, how HMM parameters can be learned, and finally how the model can be used to predict next places, there are still some points that need to be cleared. In particular, in this thesis we trained multiple models, which exhibit the same state space, but differ in their observations. Building the structure of the HMM, as well as the process of learning its parameters has been done with the help of the *pomegranate* library [28]. Its source-code is openly available on Github[1].

The library is also used to perform inference with the trained HMM, allowing us to predict next places in a straightforward fashion. At the same time, comparing predicted locations to target labels helps us to determine the performance of our model. More precisely, in this work we apply 4-fold cross validation to compare multiple models (cf. section 2.3.10). On one hand, we trained a HMM for next place prediction that does not use the time component of a sensor measurement in its observations, and focuses solely on the discrete location description. On the other hand, models were trained with observations that are the day of the week, whether it is a weekday or the weekend, and the specific hour or hour slot within a day. In addition to this, also combinations of these kinds of observations have been considered. The final HMM for next place prediction has then been chosen based on the results of the experiments in the subsequent section.

When performing cross validation, the data is typically shuffled before being split into k different sets, as we work with the assumption of the data being independent and identically distributed. Shuffling our data arbitrarily however, would mean that we lose all temporal order. To maintain some temporal dependencies in our data, we chose to first split the data according to their day of recording. These sequences of days are subsequently shuffled and split into $k = 4$ sets for training and testing respectively, as opposed to shuffling data samples individually.

Algorithm 2 shows the main work-flow of our cross validation implementation in pseudo-code. For a specific dataset, we retrieve the data for one particular user, split it into daily sequences to maintain temporal information, and shuffle the days randomly. After this, the main process of training and evaluation is performed. Here, $k = 4$ folds of the data are extracted and for each fold, the model is trained on the data in the other folds, and evaluated against the samples in the extracted fold. By averaging over the accuracy of all users, we retrieve an overall performance estimate of our model on the

[1] `https://github.com/jmschrei/pomegranate`

data.

Algorithm 2 Basic cross validation work-flow for one specific dataset

```
1: procedure DO_CV(users)
2: # Input: List of users users, could e.g. be directory
3: # Output: Mean accuracy of HMM on current dataset
4:
5:     accuracies ← []
6:     for user ∈ users do
7:         data ← GET_DATA(user)
8:         data ← SPLIT_INTO_DAYS(data)
9:         data ← SHUFFLE(data)
10:        user_acc ← []
11:        for train_set, test_set ∈ GET_SPLITS_FOR_K_FOLDS(data, k ← 4) do
12:            model ← BUILD_MODEL()
13:            model.TRAIN(train_set)
14:            res ← model.EVALUATE(test_set)
15:            user_acc.APPEND(res)
16:        accuracies.APPEND(user_acc.MEAN())
       return accuracies.MEAN()
```

When learning HMM parameters and using the trained model for next place prediction as described above, the limitations of our previous choice to use discrete observations quickly become clear. As soon as an observation is encountered that has not been part of the training sequence, the HMM fails to do a prediction. The discrete nature of the observations implies that only locations or time-descriptors that we have previously seen in the process of parameter learning have a probability higher than zero of occurring. In other words, by default, the model assumes that new observations cannot occur in our environment. For now, we solve the issue by treating it as a wrong prediction. In the future, this could be improved by e.g. assigning an emission probability larger than zero to unseen observations (cf. [19]). However, approaches that tackle this issue require at least some knowledge about what could be unseen observations in the future.

5 Experiments

5.1 The Datasets

In order to determine the performance of previously introduced approach of next place prediction, real input data has to be considered. This is done with the help of two different datasets. The Geolife GPS Trajectory Dataset has been provided by Microsoft, and will be the first base of our experiments. Secondly, we collected our own set of data for the task, which will be described in greater detail in section 5.1.2.

Figure 5.1 shows two exemplary trajectories of the datasets. An example for a trajectory from the data that was collected for this research, is depicted on the left-hand side. The right-hand side oppositely displays a partial trajectory from the Geolife dataset.

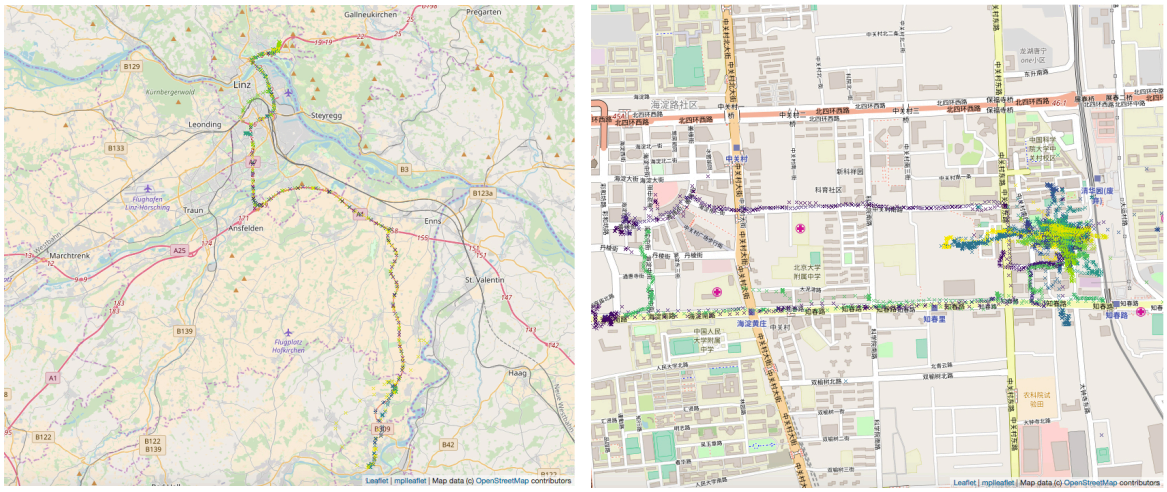


Figure 5.1: Left-hand side: Trajectory collected from a participant of our data collection process over the course of 24 days.
Right-hand side: Partial trajectory consisting of first 10000 location points, collected from the Geolife dataset over the course of 13 days. The underlying maps of both images were provided by OpenStreetMap under the Open Database Licence. The colour gradient illustrates the passage of time.

5.1.1 The Geolife GPS Trajectory Dataset

The *Geolife GPS Trajectory Dataset* was published by Microsoft in 2011 [33]. As the name suggests, it consists of GPS logs that have been recorded through various devices by

178 users. The Geolife dataset was collected over a period of multiple years in total, and provides temporal information as well as geographical data, i.e. latitude, longitude and altitude. The altitude of a geographic location will not be considered in our experiments however.

Participants of the Geolife dataset collection process recorded various different activities in their day-to-day life. Most of the data has been collected from subjects in Chinese cities [33]. Nevertheless, also trajectories from places located in the US and Europe are part of this open GPS dataset. All recorded location traces are available in files with a *.plt* extension.

The sampling rate, although generally high, varies strongly from one participant to another. The majority of trajectories are logged every one to five seconds [33]. The records are not always consecutive, however, and can have breaks of multiple days.

The recorded time-span of different users varies notably in the Geolife dataset. Since our main goal is to learn about day-to-day routines concerning the whereabouts of a person, all subjects with less than seven days of recording have been discarded. The resulting dataset that we use for our experiments is left with trajectories for 111 people.

5.1.2 The Collected Dataset

The Geolife GPS Trajectory Dataset does not provide us with ground-truth that is necessary for determining the performance of our work, in particular, the detection of significant locations. Therefore, we additionally collected a set of data with location annotations. Due to limited resources, this dataset is significantly smaller than the Geolife dataset, but is still useful to assess the quality of proposed algorithms.

The collection of our dataset was done with the help of a mobile application for Android devices. Ten participants recorded data for a period of 13 to 33 days. The profession of the participants at the time of recording diverges; people with flexible or regular working hours are represented as well as students and users without current occupation. The application adapts the *GPS Logger* for Android that is available in the Google Playstore[1] as well as on Github [21]. Location logs retrieved through this mobile application, were stored in CSV files.

When looking at the origin of collected location information, the data can be divided into two categories. The main source of the data is the GPS of a mobile phone, and has been employed by every participant. In addition to that, intermediate location measurements were collected using *network* services, i.e. cell towers and WiFi signals, if available. The logging interval was set to 30, assuring that at least 30 seconds passed before a new log-entry was made.

The collected dataset contains information that is very similar to the Geolife dataset. Most importantly a time-stamp, latitude, longitude and altitude; furthermore, the source of the measurements and a user-annotated label are part of recorded log-entries. Before starting the data collection, every participant was asked to repeatedly annotate places significant to them, i.e. locations they visit regularly for a longer amount of time. The

[1] `https://play.google.com/store/apps/details?id=com.mendhak.gpslogger`

labels serve as ground-truth for the extraction of significant places, as will be discussed in subsequent experiments. In order to simplify the annotation process for the participants, location labels could be added manually or with the help of Near Field Communication (NFC) tags. The Android application that was used for recording was therefore extended so that it can read this kind of tags, and add an appropriate annotation for the current location.

While providing a base for performance evaluations, the user-annotated labels still exhibit limitations. One issue could be a certain degree of faulty annotations, e.g. due to some room for interpretation in the definition of significant locations. On the other hand, labels were recorded infrequently, as an annotation was only made once during a visit of a certain significant location. In other words, not every recorded sample has its true location label available. Nevertheless, the labels allow us an evaluation of the proposed method.

5.2 Pre-Processing: Extracting Stay-Points

As mentioned in section 4.1, the first step in predicting next locations is to extract stay-points from the raw location data. This extraction of stay-points serves as a pre-processing step, which should first and foremost simplify the task of finding buildings that represent significant places. At the same time, using stay-points as opposed to raw data for further processing can result in reduced computational complexity. This is due to the fact that stay-points are based on accumulations of raw points in a trajectory, which naturally results in a smaller amount of points.

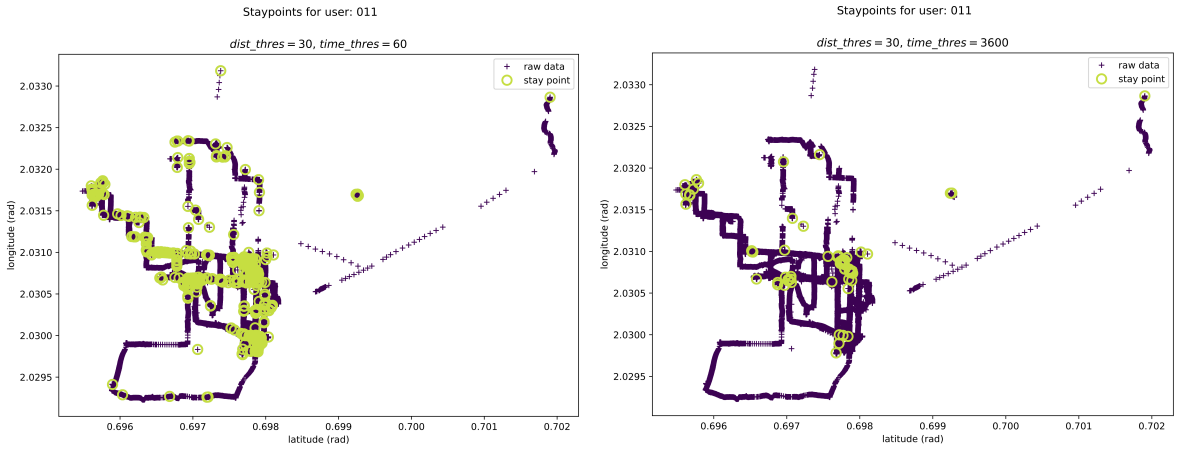
Two main arguments control the extraction of stay-points proposed by Li et al., and previously described in algorithm 1. To tune these arguments, i.e. the thresholds for distance and time, we perform a grid search. The resulting stay-points for some of the considered settings are displayed in Figure 5.2. The thresholds mainly affect the number of stay-points that are extracted, but also the exact positions are different. The images show a trajectory of the Geolife dataset, composed of raw, non-normalised, location points visualised by means of their respective longitude and latitude in radians.

Figure 5.2a visualises the stay-point extraction with $dist\_thres = 30$ and $time\_thres = 60$. Without a prior normalisation of longitude and latitude, this corresponds to detecting a stay-point whenever a person stays for at least 60 seconds within a radius of 30 meters. The short time-span implies a rather high number of extracted stay-points, as this can happen frequently in a daily routine.

On the other hand, Figure 5.2b illustrates the consequence of increasing the timing threshold. With $dist\_thres = 30$ and $time\_thres = 60 * 60$, we reduce the number of detected stay-points significantly. In other words, by requiring people to be at the same place for 60 minutes to create a stay-point, we extract a subset of previous extracted stay-points.

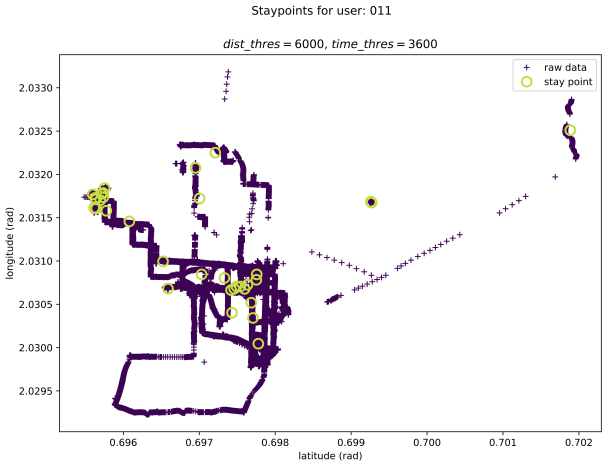
The third image, Figure 5.2c, shows the result of increasing the distance threshold. For non-normalised trajectories, taking $dist\_thres = 6000$ and $time\_thres = 60 * 60$ implies, that we detect a stay-point as soon as a person stays within 6000 meters for at

least 60 minutes. Since an increased area allows more raw measurements to be combined into a single stay-point, this again leads to a reduced number of extracted stay-points compared to the case where $dist\_thres = 30$ and $time\_thres = 60$. In other words, one stay-points captures the information for more of the raw longitude-latitude pairs. As the mean-coordinate of a stay-point strongly depends on contributing location points, this also explains the slight shift in the placement of stay-points in Figure 5.2c, compared to the results in Figures 5.2a and 5.2b.



(a) $dist\_thres = 30$ and $time\_thres = 60$; Small distance and time thresholds lead to a high number of extracted stay-points.

(b) $dist\_thres = 30$ and $time\_thres = 3600$; The increased timing threshold lessens the number of extracted stay-points.



(c) $dist\_thres = 6000$ and $time\_thres = 3600$; Increasing the distance threshold lessens stay-points, and shifts their locations.

Figure 5.2: The influence of different parameters for the stay-point extraction. Trajectory taken from the Geolife dataset. Longitude and latitude values are given in radians.

In their work, Li et al. chose a distance threshold of 200 meters, and a time threshold of

30 minutes for the extraction of stay-points [16]. In our approach however, we conducted a random search over a grid with different distance thresholds ranging from 100 to 1500 units with an average interval of 50, and time thresholds ranging from 1 to 120 minutes with intervals of 5 to 10 minutes. As the process of stay-point extraction has immediate impact on how well the subsequent detection of significant places performs, this performance will be used as a base to judge the eligibility of possible stay-point extraction arguments.

5.3 Pre-Processing: Finding Significant Locations

After extracting stay-points based on trajectories, we determine what regions could be important to each user. This means that we have to group stay-points to form significant places. In section 4.2, we looked at some issues occurring with common clustering algorithms from a more abstract point of view. In this section, we want to see how well these algorithms perform on real-world data. For this, we will primarily focus on the dataset that we collected. The annotations made by the participants give us an idea of places that are significant to them, and in particular, how many of these places there are to detect. As the annotations have been recorded infrequently, i.e. the true significant location is not available for every recorded sample, clustering results cannot be compared and therefore evaluated directly. Instead, we rank the performance of different algorithms primarily by their ability to find the correct number of clusters. In addition to that, we compare the average coordinates of predicted and recorded significant locations in section 5.3.6. The often diverging number of predicted and true locations however, does not allow a straightforward computation of its deviations.

In what follows, the Mean Squared Error (MSE) of different clustering methods describes how close the algorithm can approximate the real, annotated, number of significant locations of a user. It is computed as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2. \tag{5.1}$$

Here, Y denotes the real number of significant places of a user; $\hat{Y}$ on the other hand, describes the predicted number thereof. N is the total amount of users in the dataset.

In order to find the best clustering algorithm, we should consider multiple parameter settings for each of the algorithms. This can be done by conducting a grid search over different parameters, and explore the impact of these arguments on the MSE. First and foremost, we will look at parameters specific to each clustering method. As discussed in section 5.2, however, we also need to find the best arguments for the extraction of stay-points, i.e. a fitting distance and time threshold. Finally, possible consequences of normalising coordinates (cf. section 4.1.1) prior to the extraction of stay-points will be examined.

5.3.1 Finding a Predefined Number of Stay-Point Clusters

The first issue when applying common clustering algorithms for the task of significant place detection is the need to specify the number of resulting clusters (cf. section 4.2). While a vast number of algorithms introduced in section 2.2 exhibits this characteristic, the prime example is the K-means clustering algorithm.

The main argument of the K-means algorithm is the number of clusters, k. In addition to this, we want to find fitting time and distance thresholds for the stay-point extraction itself, as previously mentioned. Finally, we examine the impact of normalising latitude and longitude prior to the extraction of stay-points.

Figure 5.3 shows results obtained by applying the K-means algorithm with $k = \{3, 4, 5\}$ to two different participants of our dataset. The real number of significant places is five for the images on the left-hand side, and three on the right-hand side. For K-means clustering, the predicted amount of significant locations is equal to the predefined number k. This means that normalising or adapting the thresholds for stay-point extraction will not have an impact on the MSE. However, it does change where exactly the stay-points are located.

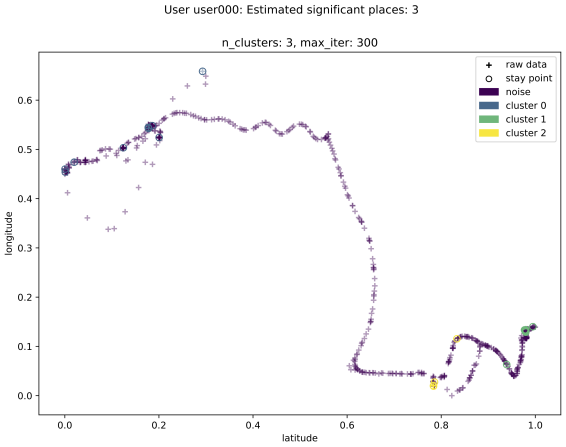
Table 5.1 lists the MSE for different choices for the number of clusters k, as an average over the ten users from our dataset. As other arguments leave the MSE unchanged, they are not considered in the table.

k / *nr_clusters*	MSE
1	8.7
2	4.3
3	1.9
4	**1.5**
5	3.1
6	6.7

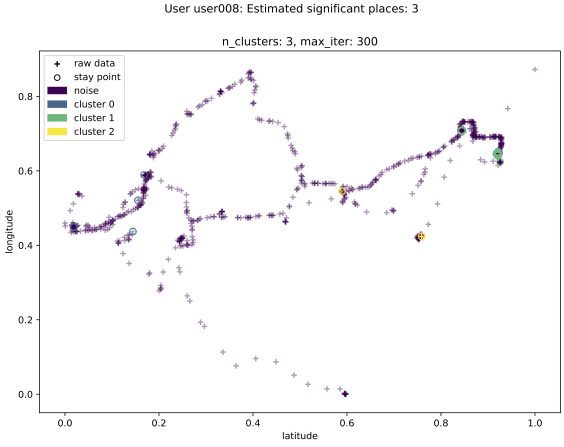
Table 5.1: Average MSE of K-means and agglomerative hierarchical clustering applied to our dataset

Simultaneously, table 5.1 shows the MSE of agglomerative hierarchical clustering. Implementations of this algorithm typically require a predefined number of resulting clusters, corresponding to the predicted amount of significant places. Once again, this makes the prediction itself independent from other arguments such as the linkage (cf. section 2.2.3), which is why both, K-means and agglomerative hierarchical clustering, lead to the same MSE.

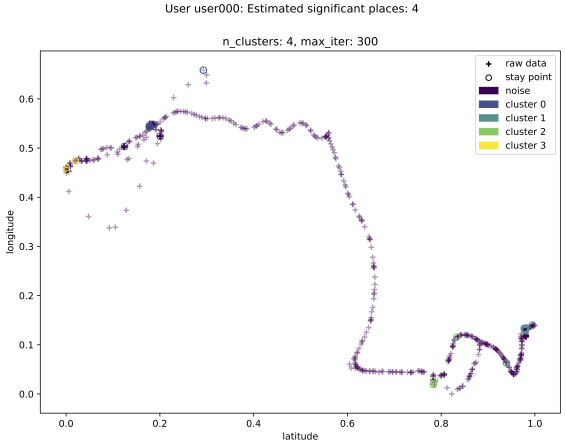
Figure 5.4 illustrates detected significant places, obtained by applying K-means and agglomerative hierarchical clustering. The images show a trajectory, collected by one user from our dataset. Choosing the number of clusters to be four, the amount of real and predicted significant places of this user are equal for both algorithms. Nevertheless, the two images visualise the weakness of solely looking at the MSE. Different algorithms or parameters might result in the same, correct, but predefined, number of significant



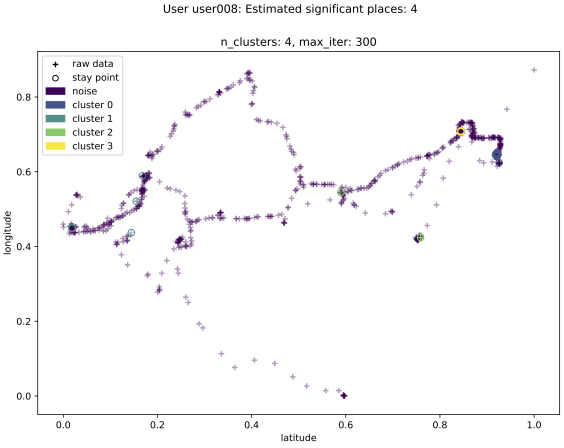
(a) User 000, significant places: extracted = 3, real = 5



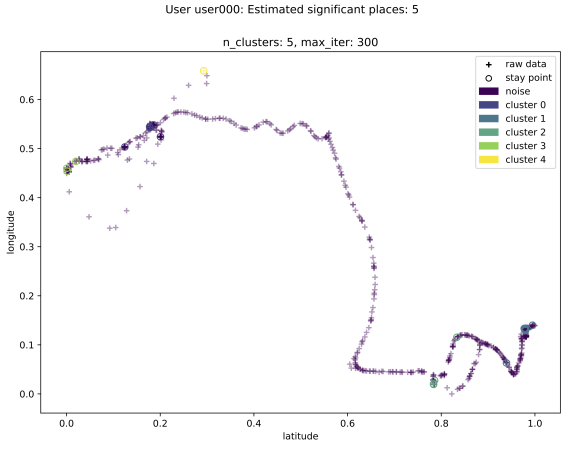
(b) User 008, significant places: extracted = **3** real =**3**



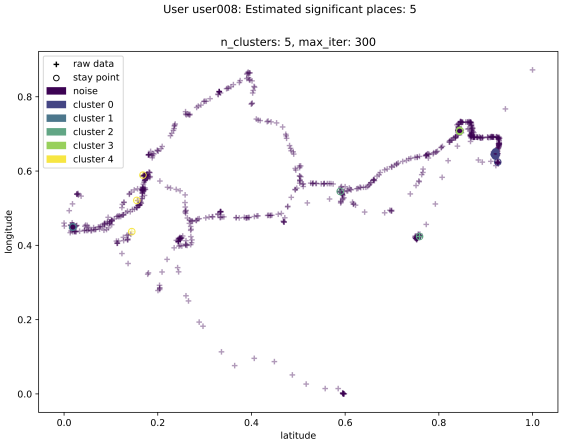
(c) User 000, significant places: extracted = 4, real = 5



(d) User 008, significant places: extracted = 4, real = 3



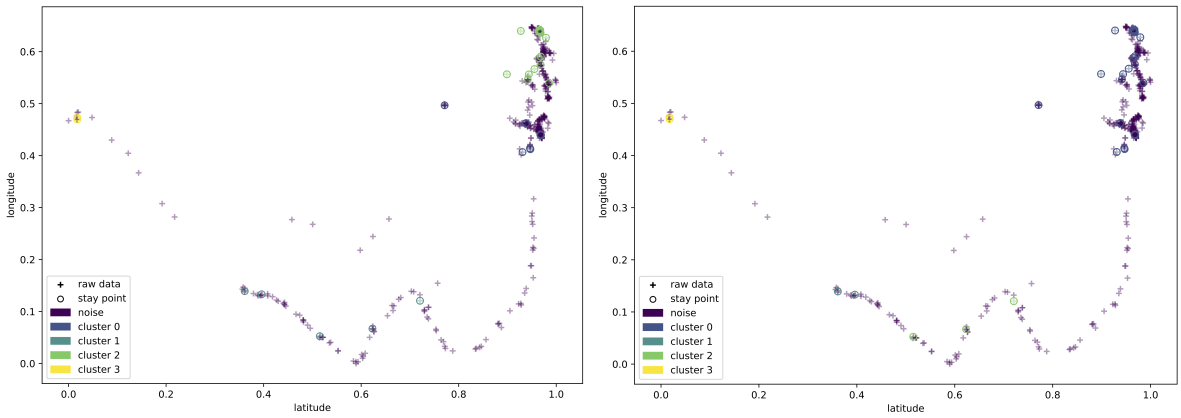
(e) User 000, significant places: extracted = **5**, real = **5**



(f) User 008, significant places: extracted = 5, real = 3

Figure 5.3: Clustering results of K-means when applied to two different users of our dataset. The number of extracted significant places is equal to argument k. Stay-points were extracted with $dist\_thres = 200$ and $time\_thres = 30 * 60$ from normalised data-points.

places. They do, however, lead to considerably different arrangements of data-points within the clusters.



(a) K-means clustering

(b) Agglomerative clustering

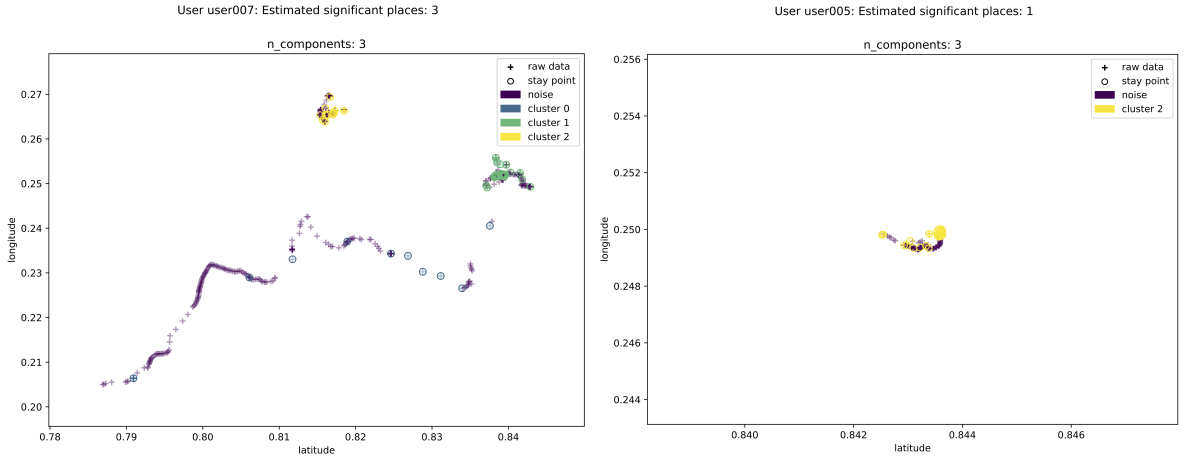
Figure 5.4: K-means and agglomerative hierarchical clustering with linkage=*average* applied to the trajectory of User 005 of our dataset. The algorithms lead to different shapes of the clusters; they do not influence the number of extracted significant places (4) however.

5.3.2 Finding Significant Locations with GMMs

Applying GMMs to the task of significant place detection leads to similar issues as the application of K-means or hierarchical clustering, and requires a predefined number of mixture components. First, we take a look at results obtained if we do not normalise the coordinates of data-points prior to the stay-point extraction. Figure 5.5 illustrates that despite defining three components for a GMM, we can actually get less resulting clusters. Concretely, the image on the right-hand side shows the trajectory of a user, who appears to travel a lot less on a regular day, as opposed to the user on the left-hand side. This results in the points on the right-hand side to be grouped into one single significant place, whereas we find multiple ones on the left-hand side.

Table 5.2 contains the MSE for different parameters, as an average over all users within our non-normalised dataset. We previously mentioned, that the amount of predicted significant places is at most the number of components of a GMM. Also, GMMs do not support noise and therefore assign every point to a cluster, meaning that there is always at least one cluster found, regardless of chosen distance or time thresholds. Therefore, a GMM with $nr\_components = 1$ achieves the same MSE as K-means or agglomerative clustering with $nr\_clusters = 1$.

For our dataset, table 5.2 also suggests that four components work well for GMMs, similar to the performance peak of K-means and hierarchical clustering for four predefined clusters. When extracting stay-points with a time threshold of ten minutes and a distance threshold such as 200 meters, a four-component GMM results in a MSE of



(a) Significant places: extracted = 3, real = 1

(b) Significant places: extracted = 1, real = 4

Figure 5.5: Trajectory of users 007 and 005 in our data-set. Result of clustering with a three-component GMM, applied to stay-points extracted with $dist\_thres = 100$ and $time\_thres = 600$ from non-normalised coordinates, given in radians.

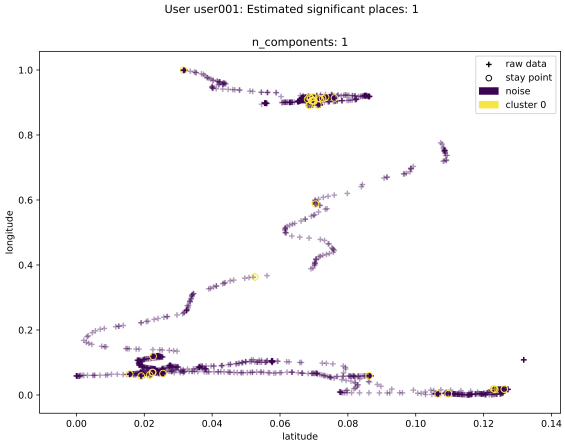
4.2. These two thresholds appear to generally perform well for different numbers of components. The lowest overall error we achieved with a GMM in this scenario, can not yet match with the results obtained with K-means or hierarchical clustering.

Normalising the coordinates of data-points before we extract stay-points, should ensure comparable trajectories. Especially for users, which travel vastly different distances in their daily routine, this is useful. In experiments, based on the stay-point extraction of normalised data-points, we are therefore able to overcome previous issues that were encountered with GMMs. In particular, this means we can find multiple significant places even for users, which do not travel far.

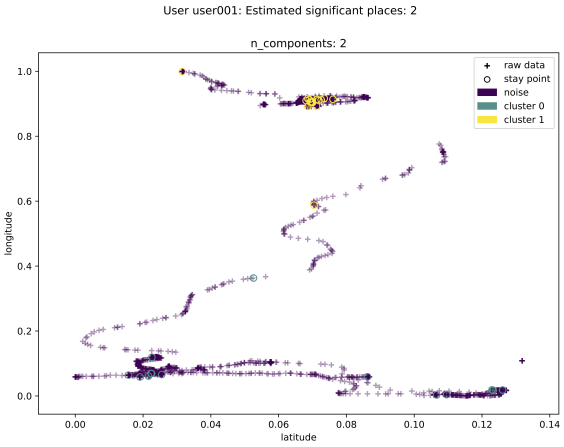
Figure 5.6 shows a trajectory of one user within our dataset. The real amount of significant places for this person is four; the number of components however varies from one to six.

dist_thres	*time_thres*	*nr_components*	MSE
any	any	1	8.7
200	10*60	2	5.4
200	10*60	3	4.5
200	10*60	4	**4.2**
100	5*60	5	4.9
200	40*60	6	5.1

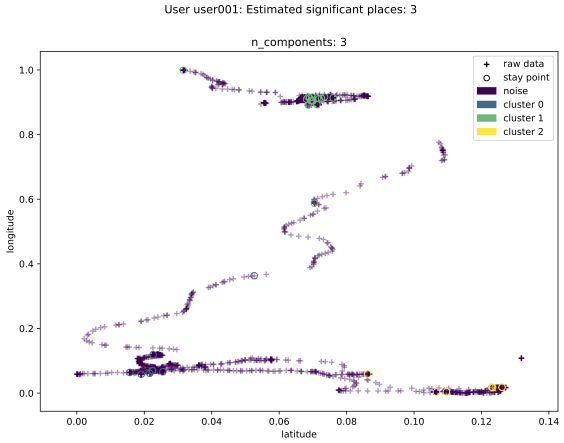
Table 5.2: Average MSE of GMM clustering applied to our dataset, with $normalised = False$ for all entries



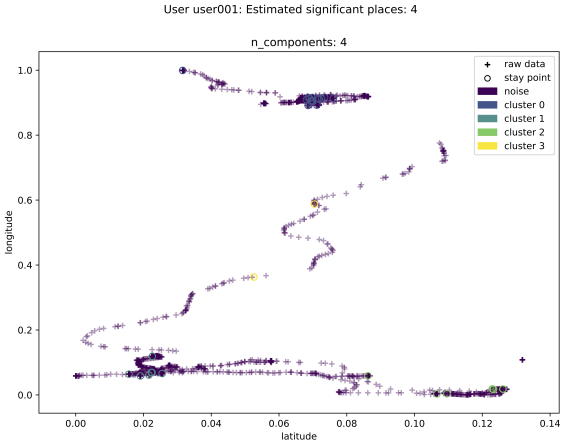
(a) Number of components = 1



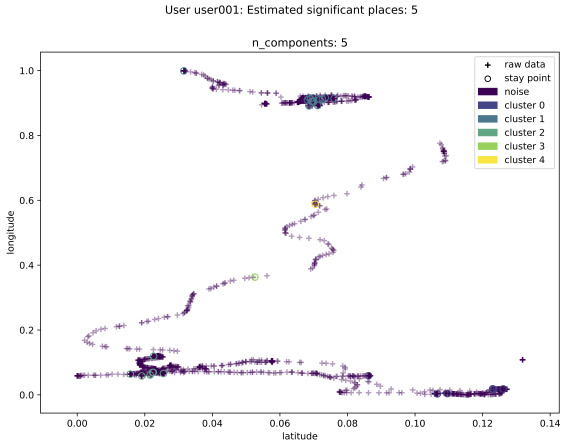
(b) Number of components = 2



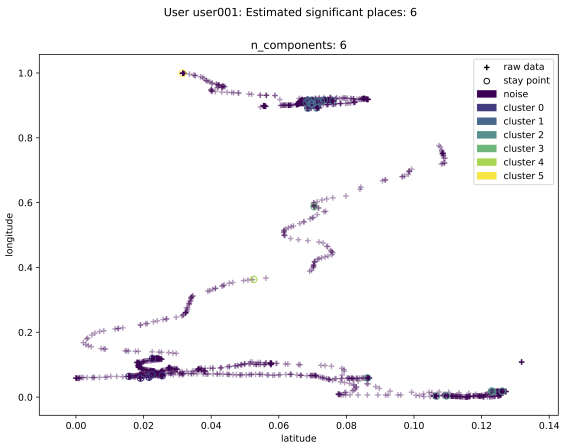
(c) Number of components = 3



(d) Number of components = **4**



(e) Number of components = 5



(f) Number of components = 6

Figure 5.6: Trajectory of user 001 of our data-set, with four real significant places. Result of clustering with a GMM with a varying number of components, applied to stay-points extracted from normalised data.

The images in Figure 5.6 illustrate an example for which the number of GMM components corresponds to the amount of predicted significant places. As soon as we perform a normalisation of the data-points prior to the stay-point extraction, this happens frequently, which is why the resulting MSE is similar to the results of K-means and agglomerative hierarchical clustering in section 5.2. Table 5.3 contains a summary of MSEs after applying GMMs to cluster stay-points based on normalised data. In the case of one, two or three components, the results conform to the MSE achieved by K-means and hierarchical clustering, for various time and distance thresholds. For an increasing number of components, the error slightly deviates. This implies on one hand, that the MSE of clustering with GMMs can be reduced from 4.2 as above, to 1.4, with the best-fitting number of components again to be four. On the other hand, the GMM slightly outperforms K-means and hierarchical clustering, as the number of resulting significant places is not always bound to the amount of defined GMM components.

dist_thres	*time_thres*	*nr_components*	MSE
any	any	1	8.7
various	various	2	4.3
various	various	3	1.9
200	40*60	4	**1.4**
200	40*60	5	2.6
200	40*60	6	5.4

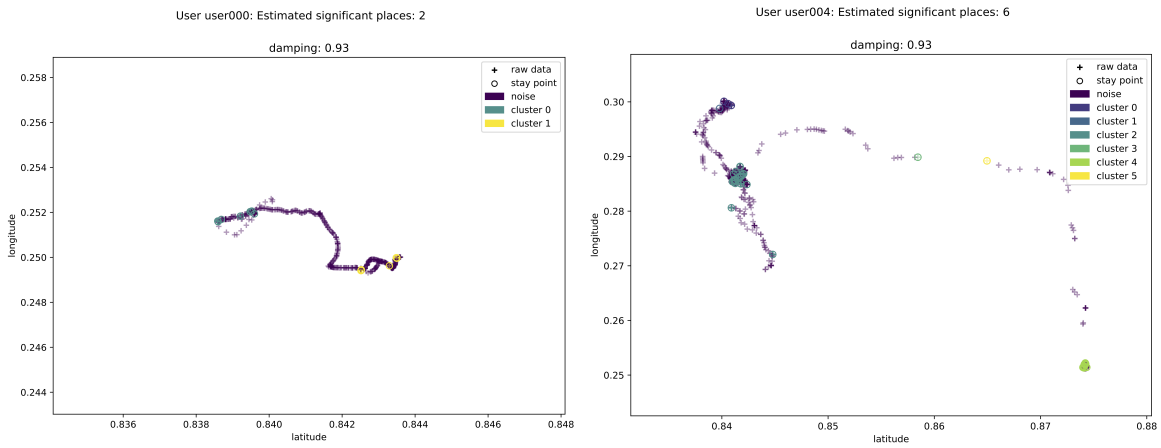
Table 5.3: Average MSE of GMM clustering applied to our dataset, with $normalised = True$ for all entries

5.3.3 Finding Significant Locations with Affinity Propagation

As opposed to previous clustering algorithms, affinity propagation does not require a predefined number of clusters (cf. section 2.2.5). However, as discussed in section 4.2, applying it to stay-points turns out to be challenging. This is due to the often high amount of stay-points, which do not contribute to a significant place, and therefore represent noise.

Let us first take a look at the results of affinity propagation when applying the algorithm on stay-points that were extracted from non-normalised coordinates. Next to the distance and time threshold for the stay-point extraction, we have to tune one parameter specific to affinity propagation. The so-called *damping* factor primarily helps to avoid numerical oscillations during the update of messages, having a value between 0.5 and 1 [23].

Figure 5.7 shows the results from applying affinity propagation to the trajectories of two users from our dataset. The predicted number of significant places does not correspond to the real amount in both of the cases; however, the difference is relatively small. With a distance threshold of 100 meters, and a time threshold of 60 minutes, the



(a) significant places: extracted = 2, real = 5

(b) significant places: extracted = 6, real = 5

Figure 5.7: Trajectories of users 000 and 004 in our dataset. Result of affinity propagation with $damping = 0.93$, applied to stay-points extracted with $dist\_thres = 100$ and $time\_thres = 3600$ from non-normalised data-points, given in radians.

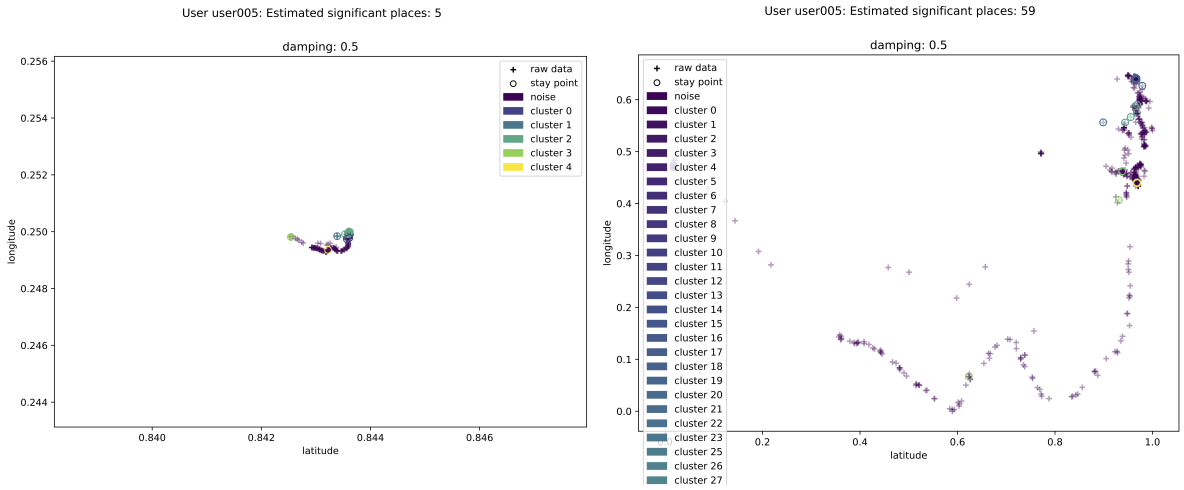
average MSE over all users is 2.8 for a damping factor of 0.93. Nevertheless, the right-hand side image of Figure 5.7 already indicates the issue with noisy stay-points. If we reduce the damping factor, this becomes even more clear. Leaving remaining parameters unchanged, and changing damping to be 0.5, the number of predicted significant places for user 004 rises to 13. For other users, 20 or more significant places are predicted, as noise tends to be assigned to entirely separate clusters.

Next to the impact the damping factor shows on the MSE, also distance and time thresholds change the prediction obtained by affinity propagation. The first lines in table 5.4 show the MSE, depending on different arguments, but with non-normalised data-points. A low distance and time threshold, which result in a high amount of extracted stay-points (cf. section 5.2), presents affinity propagation with more difficulties. Reducing the number of extracted stay-points and therefore the noise, results in a significantly better MSE. This measure of performance does not, however, consider the amount of noisy stay-points that have been put into separate clusters that eventually lead to the high MSE.

The lower part of table 5.4 suggests that normalising coordinates of data-points prior to the extraction of stay-points lowers the MSE. With a distance threshold of 50, a time threshold of 60*60, and a damping factor of 0.93, we can achieve a MSE of 2.5. Nevertheless, the noise remains a difficulty for the algorithm. Reducing the damping factor to 0.5 results, with otherwise unchanged arguments, in a MSE of 451.0. Note here, that for both, the normalised and non-normalised case, the same parameters led to the best results in our grid search. This, however, is not common, as is illustrated in Figure 5.8. For a damping factor of 0.5, the chosen distance threshold of 100 and time threshold of 60*60 work a lot better for stay-points based on non-normalised data.

normalised	*dist_thres*	*time_thres*	*damping*	MSE
False	50	20*60	0.50	2398.9
False	50	20*60	0.93	11.6
False	100	60*60	0.50	342.9
False	100	60*60	0.93	2.8
True	50	20*60	0.50	2032.0
True	50	20*60	0.93	4.7
True	100	60*60	0.50	451.0
True	100	60*60	0.93	**2.5**

Table 5.4: Average MSE of affinity propagation applied to our dataset



(a) *normalised* = *False*, significant places: extracted = 5, real = 4

(b) *normalised* = *True*, significant places: extracted = 59, real = 4

Figure 5.8: Trajectory of user 005 in our dataset. Result of affinity propagation with *damping* = 0.93, applied to stay-points extracted with *dist_thres* = 100 and *time_thres* = 3600.

5.3.4 Finding Significant Locations in a Density-Based Fashion

In section 4.2, we came to the conclusion that density-based clustering techniques might be the best fit for detecting significant locations. As a prominent example, DBSCAN presents multiple characteristics which are suitable for our task. Next to its ability to detect various shapes of clusters as well as noise, the algorithm does not require defining the number of significant places we are looking for. Also, the scikit-learn library of Python provides an implementation of DBSCAN, which is why we chose this particular algorithm within the group of density-based clustering methods.

As in previous sections, we again take a look at the results we obtain from clustering stay-points based on non-normalised data. Next to the arguments controlling the stay-point extraction, we need to tune two additional parameters specific to DBSCAN (cf. section 2.2.4). These arguments are the minimum number of points in a cluster, i.e. min_points, and radius ϵ.

Table 5.5 shows the MSE of DBSCAN for an excerpt of different arguments. The first lines comprise the results obtained without a normalisation of data-points. Revisiting the proposition of Li et al., let us first examine the best ϵ and min_points parameters for $dist\_thres = 200$ and $time\_thres = 30*60$. Taking $\epsilon = 0.0002$ and $min\_points = 3$ leads to a MSE of 1.6. To put this into perspective, the best average error we obtained of affinity propagation for non-normalised data was 2.8.

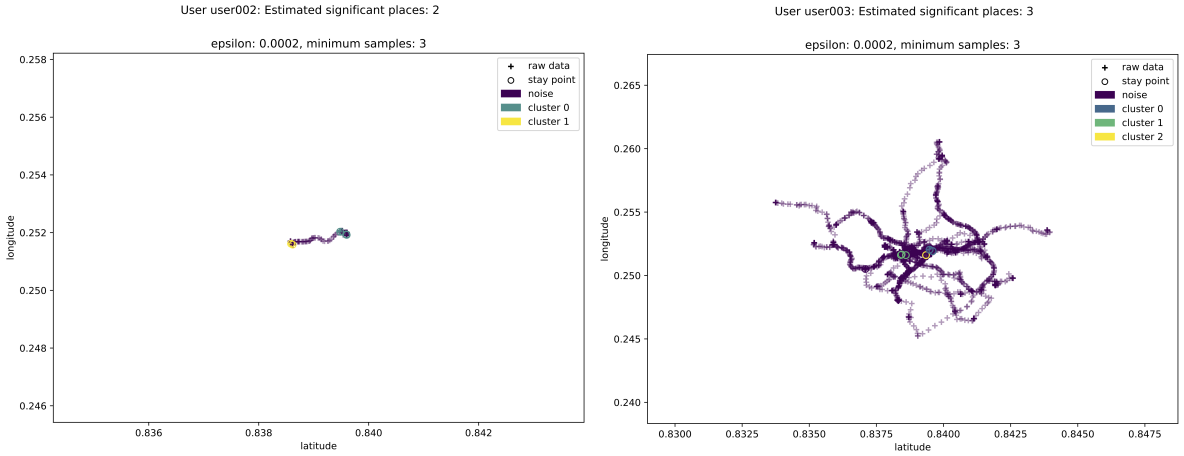
The proposed parameters of Li et al. lead, at the same time, to the lowest MSE within our grid search. Varying the time threshold slightly, as an example, results in a minimally changed average error of 1.7. This is still higher than the best result achieved by algorithms with a predefined number of clusters, which is why we subsequently look at the case of normalised data-points.

normalised	*dist_thres*	*time_thres*	*min_points*	ϵ	MSE
False	200	30*60	3	0.0002	1.6
False	200	40*60	3	0.0002	1.7
True	200	30*60	2	0.04	1.7
True	1000	25*60	2	0.04	**0.9**

Table 5.5: Average MSE of DBSCAN applied to our dataset

Figure 5.9 shows results of applying DBSCAN with parameters from table 5.5 for non-normalised raw data. On the left-hand side, the predicted number of significant places does not correspond to the real amount. The image on the right-hand side, however, shows a correct prediction. When working with non-normalised coordinates, previous clustering algorithms often encountered issues with trajectories spanning a low overall distance. Note that in particular the trajectory on the left hand side covers a relatively small area. Previous algorithms were often unable to find more than one single cluster, but DBSCAN copes with these kind of discrepancies within our dataset better.

The second part of table 5.5 shows average errors of DBSCAN when applied to stay-points extracted from normalised data. Distance thresholds are not given in meters in



(a) User 002, significant places: extracted = 2, real = 3

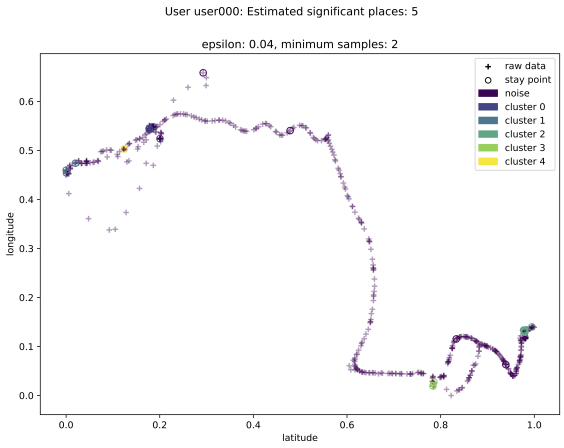
(b) User 003, significant places: extracted = **3**, real = **3**

Figure 5.9: Trajectories of two users in our dataset. Result of applying DBSCAN to stay-points extracted with $dist\_thres = 200$ and $time\_thres = 30 * 60$ from non-normalised data-points; $min\_points = 3$, $\epsilon = 0.0002$

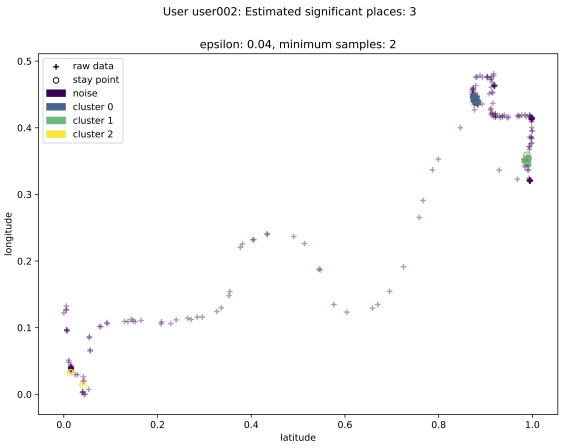
this case. Nevertheless, we can examine how the MSE is influenced by the normalisation of data-points. Taking a distance threshold of 200, and a time threshold of 30*60, we can again achieve an error of 1.7 with $min\_points = 2$ and $\epsilon = 0.04$. Increasing the distance threshold to 1000 and slightly decreasing the time threshold, can push the MSE below 1. More precisely, with $dist\_thres = 1000$, $time\_thres = 25 * 60$, $min\_points = 2$ and $\epsilon = 0.04$ we obtain a MSE of 0.9. This average error of DBSCAN therefore outperforms all of the previously discussed algorithms.

Figure 5.10 illustrates how detected significant places could look like when applying DBSCAN for the task. The shown results are obtained with the parameters leading to the smallest MSE in table 5.5. For the collected dataset, almost half of our predictions match the corresponding real number of significant places. The upper images here show two exemplary trajectories for which this holds. The lower two images illustrate trajectories of users for which we do not obtain the correct prediction; however, except for one case, the difference between real and estimated number of significant locations is exactly one.

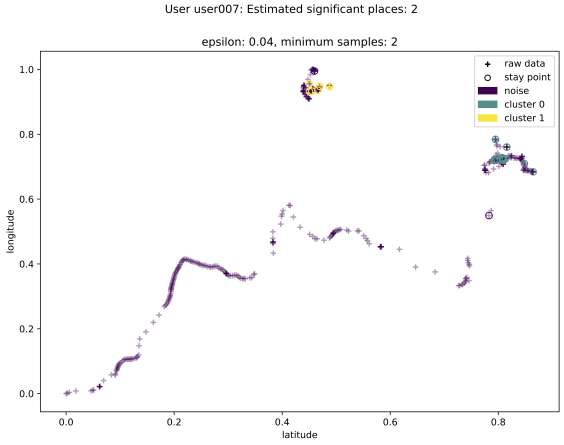
Wrong predictions made by DBSCAN can have a variety of reasons. Parameters might be fitting for a certain number of users within a dataset, while being sub-optimal for others. The normalisation of raw data prior to the extraction of stay-points simplifies the task of finding suitable parameters for multiple users, but does not resolve the issue completely. In addition to that, one also needs to consider the relatively short period of time in which our dataset was collected. Some significant places might not be detected whenever a participant could not visit the location frequently enough during the recording period. This limitation however, gives us room for future work with a more comprehensive dataset (cf. section 6.1).



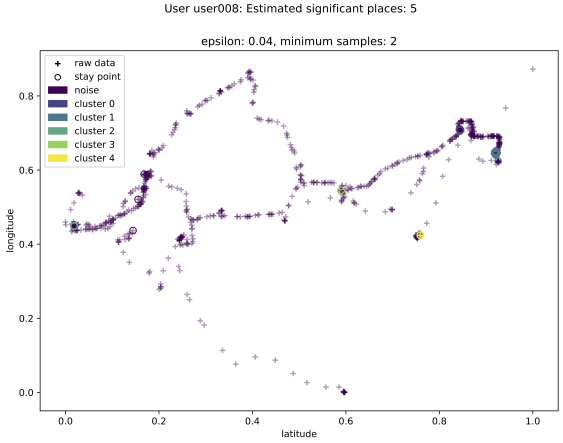
(a) User 000, significant places: extracted = **5**, real = **5**



(b) User 002, significant places: extracted = **3**, real = **3**



(c) User 007, significant places: extracted = 2, real = 1



(d) User 008, significant places: extracted = 5, real = 3

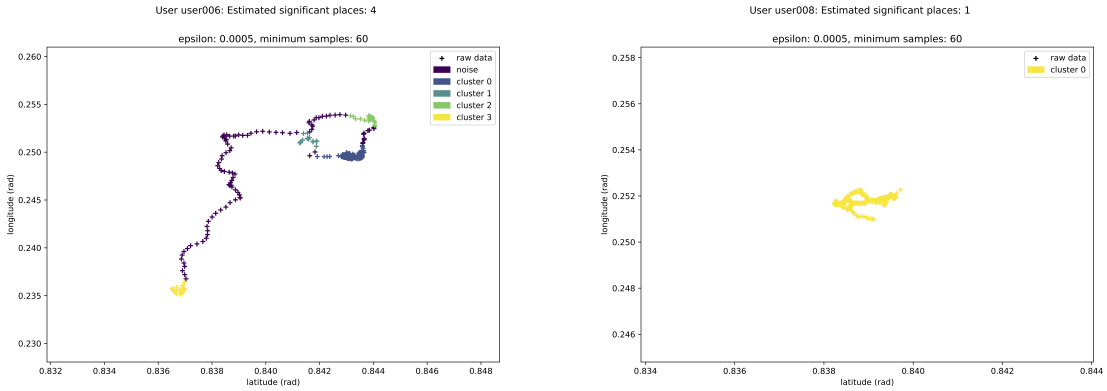
Figure 5.10: Trajectories of various users in our dataset. Result of applying DBSCAN to stay-points extracted with $dist\_thres = 1000$ and $time\_thres = 25 * 60$ from normalised data-points; $min\_points = 2$, $\epsilon = 0.04$

5.3.5 Finding Significant Locations Based on Raw Data

So far, we only performed the detection of significant places based on previously extracted stay-points. This should primarily avoid issues when trying to discover buildings Li et al. In this section however, we want to take a look at the performance of grouping raw location data to form significant places. Due to a high amount of noise within raw data, we apply DBSCAN once more.

The two arguments we subsequently consider, are min_points and ϵ of DBSCAN. Distance and time thresholds do not need to be considered due to the lack of stay-point extraction. However, we still want to examine whether normalising the coordinates of a data-point has an impact on the MSE of DBSCAN in this setting.

Figure 5.11 shows two trajectories within our dataset. The given result is obtained by applying DBSCAN, with $min\_points = 60$ and $\epsilon = 0.0005$, to non-normalised data. The image to the left shows a correct prediction of the amount of significant places. On the other hand, the right image illustrates a prediction where the number of clusters does not align with the ground truth. The trajectory on the right-hand side originates from a user that tends to travel smaller distances in a day-to-day life. This is also indicated by the labelled axes of the two plots. Applying DBSCAN to raw, non-normalised, data, once more encounters difficulties with diverging distances of different users. Radius ϵ is equal for both users represented in the plot, which fits for the left-hand side. As the data-points within the right image are much denser, the radius covers the entire area, and every point is part of the same cluster.



(a) User 006, significant places: extracted = **4**, real = **4**

(b) User 008, significant places: extracted = 1, real = 3

Figure 5.11: Trajectories of two users in our dataset. Result of applying DBSCAN with $min\_points = 2$ and $\epsilon = 0.0005$ to raw, not-normalised data-points.

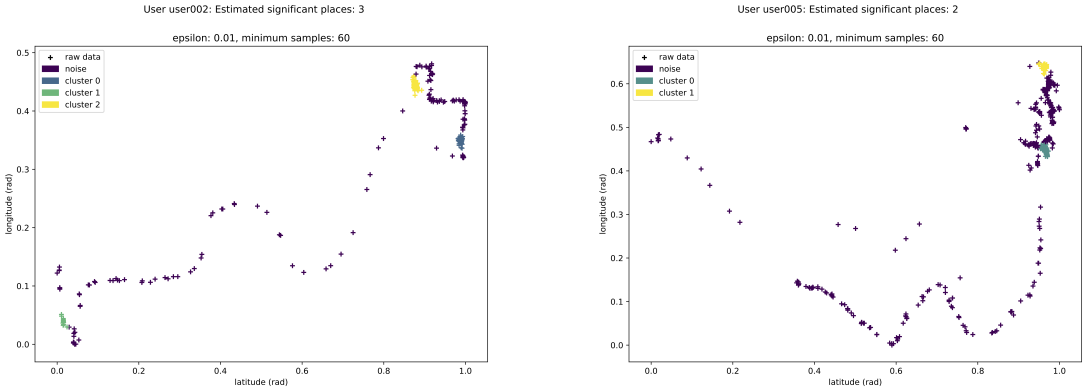
Table 5.6 shows the MSE achieved by the parameters used for the results displayed in Figure 5.11. While it represents the minimum error over all considered parameters, it remains relatively high when compared to other clustering approaches. The second row in the table points out, that normalising the coordinates prior to the detection of

significant places can reduce the MSE significantly. Using a number of minimum points $min\_points = 60$, and $\epsilon = 0.01$, achieves an error, averaged over all users within the dataset, of 2.1. This is higher than the MSE we achieved when applying DBSCAN on stay-points without prior normalisation.

normalised	*min_points*	ϵ	MSE
False	60	0.0005	5.2
True	60	0.01	2.1

Table 5.6: Average MSE of DBSCAN applied to our raw dataset

Despite the relatively low MSE we achieve when clustering normalised data-points, only one number of significant places is predicted correctly by DBSCAN. Figure 5.12 shows results of applying the parameters from table 5.6. The image on the left-hand side illustrates the trajectory, in which the predicted number of significant places corresponds to the ground truth. Predictions concerning the remaining trajectories tend to exhibit a difference of one or two significant locations. One trajectory, for which the predicted number of significant places is lower than the actual amount, is given in the image to the right.



(a) User 002, significant places: extracted = **3**, real = **3**

(b) User 005, significant places: extracted = 2, real = 4

Figure 5.12: Trajectories of two users in our dataset. Result of applying DBSCAN to raw, normalised, data-points; $min\_points = 60$, $\epsilon = 0.01$

5.3.6 Predicting Locations of Significant Places

In previous experiments we compared various approaches by means of how well they predict the real number of significant places. If we solely consider significant locations such as abstract location labels, this might be a sufficient measure. As long as the exact

coordinates of a significant place is not relevant, the main goal is to detect the correct amount, and map an extracted $cluster_x$ to any abstract location label.

Working with abstract significant places is a restriction, in particular when reasoning about the precise coordinates of current whereabouts of a user. As we perform the prediction of next places with the help of a HMM, however, this restriction is reasonable. HMMs exhibit a discrete state-space, meaning that we can only predict a finite set of locations. Nevertheless, every location label corresponds to actual coordinates in the world, differing from user to user. Next to examining the amount of predicted significant places, we take a look at the quality of our predictions, i.e. how close they are to the real coordinates of the significant place.

The often diverging number of real and predicted significant places complicates quantifying the quality of our predictions. Figures 5.13 and 5.14 illustrate the mean coordinate of predicted and real significant locations of all ten participants in our dataset. The first images, i.e. in Figure 5.13, show trajectories with an accurate prediction of the amount of significant locations. The parameters leading to subsequent results are described in section 5.3.4.

Despite a correct number of significant places, the locations do not coincide in all cases. For only one user, we manage to predict the placement of all significant places within close proximity, as is depicted in Figure 5.13. The remaining trajectories reveal some issues with our predictions. Whenever a real significant location has its mean coordinate at a position outside of the trajectory, we fail with our prediction. Possible reasons for this to occur are imprecise sensor measurements, or annotation errors. The mean coordinates of our predicted significant places on the other hand, tend to be placed close to the points that make up a trajectory. A second problem is due to real significant locations that are very close to each other. Whenever they lie within the radius we defined for extracting significant places with DBSCAN, we fail to distinguish between them.

Figure 5.14 on the other hand shows trajectories for which we can not predict the correct number of significant locations. Also for these results, we use the most fitting arguments determined in section 5.3.4. The upper left image illustrates one possible reason, why a different number of significant places is detected. Once more, two locations significant to a user lie within such close range, that they can not be distinguished by DBSCAN. The upper right image shows a similar case, in which all real significant places are located in close proximity. A second notable factor is illustrated in the two images on the second row. All real significant locations overlap with our predictions. However, we estimate additional significant places. A possible scenario for this phenomenon is whenever a user stays at a location for a rather long time, without considering it significant. The short recording period of the dataset requires a rather low minimum number of points, defined with the *min_points* argument of DBSCAN, to form a significant location. Due to this, places visited irregularly for a longer period of time can be easily considered meaningful.

In what follows, we want to base the prediction of next locations on the extracted significant places. For this process, we will use the arguments we determined in section 5.3.4, since they minimise the MSE over all examined clustering algorithms. Even though

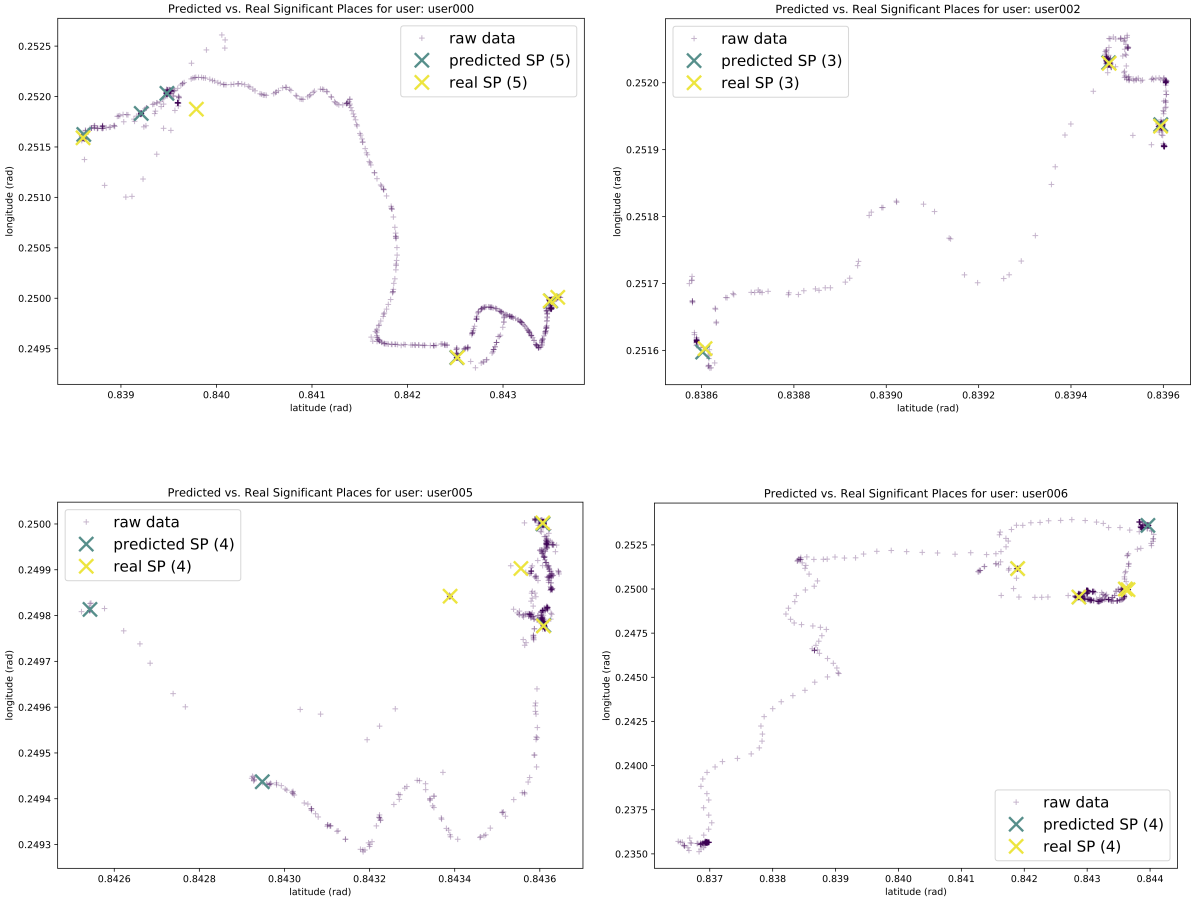


Figure 5.13: Trajectories of users with a correctly predicted number of significant places. Real and estimated significant locations are visualised and confronted

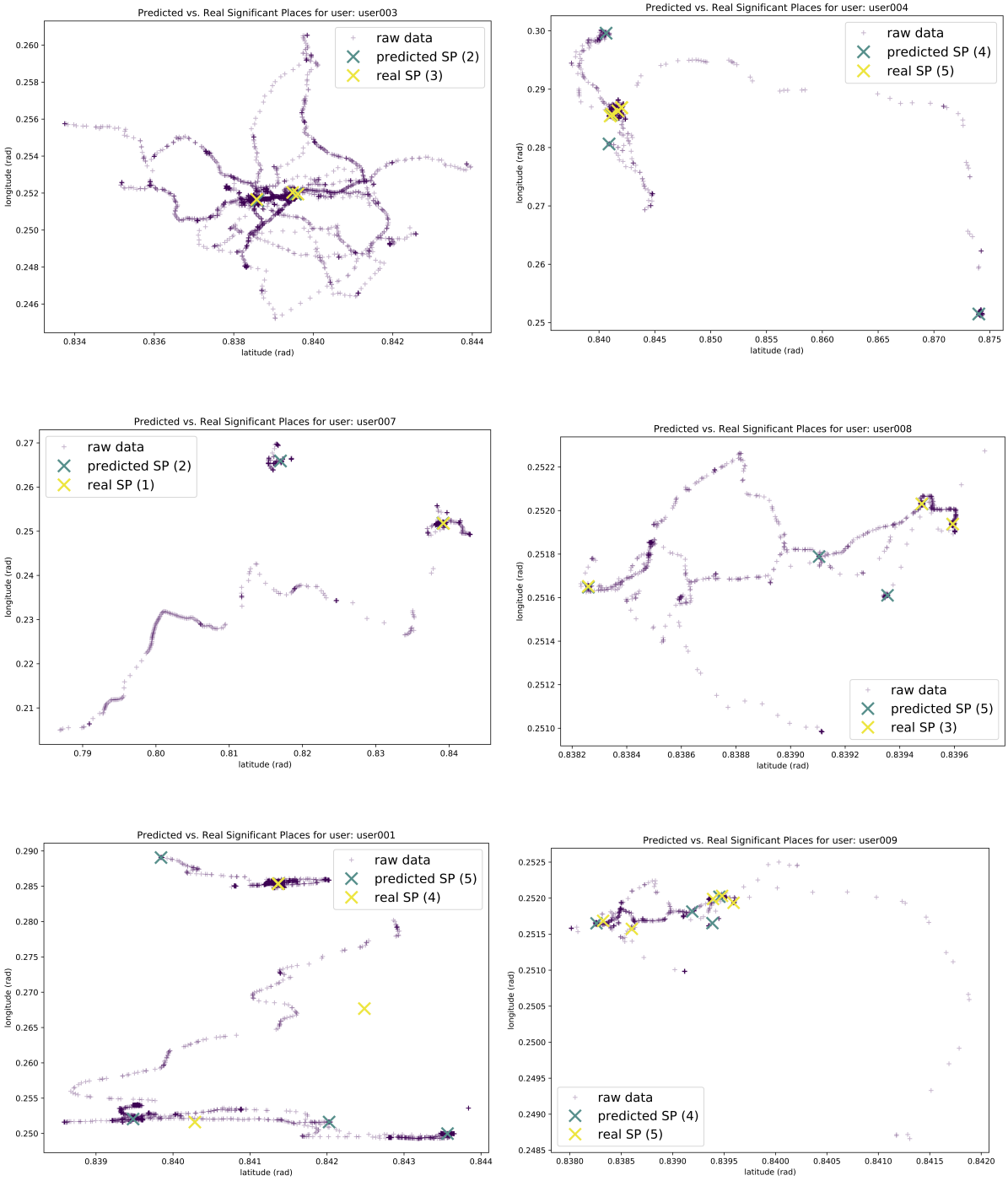


Figure 5.14: Trajectories of users with an in-correctly predicted number of significant places. Real and estimated significant locations are visualised and confronted

the number of detected significant places is not the only factor to be considered, it is crucial for our experiments. Subsequently, we will also take a closer look at the Geolife dataset. As it is differently structured and recorded over a longer period of time, parameters tuned to this dataset would likely look different. However, as we do not have any sort of ground-truth for our specific task, we have no choice but to use the same parameters for both datasets.

5.4 Predicting Next Locations

In section 4.3, we saw that HMMs allow us to predict next places. Due to the discrete state space of a HMM, these predictions originate from a discrete set of possible locations. After pre-processing real data where we extract stay-points and find significant places for a person, we have such a set of location labels available. Using significant places as possible states for our next place prediction model allows us to reason about the future location of a person. In other words, for every person, we first perform pre-processing in order to extract locations that are significant to them, before we train a HMM for each user to be able to reason about their whereabouts.

In this thesis, we look at various kinds of HMMs, as previously stated in section 4.3. Their performance on the task of next place prediction is then used as the deciding factor for choosing a final model. While the state space of every HMM consists of the extracted significant places, including a state for noise, we focus particularly on different observation spaces. In order to be able to train the HMM for next place prediction in a supervised way, subsequent experiments use previous pre-processing results as an approximation of the ground-truth. This also allows us to measure the performance of our models on the Geolife dataset. For the experiments in 5.3, this was not possible due to the lack of ground-truth to compare it with.

5.4.1 Different HMM Structures

Before HMMs are trained and their performance can be determined, we take a look at different HMM structures that will be put to the test. As stated previously, the state-space coincides for all models that we consider. Figure 5.15 illustrates possible states of a HMM for next place prediction, which correspond to previously found significant places and one additional state for noise.

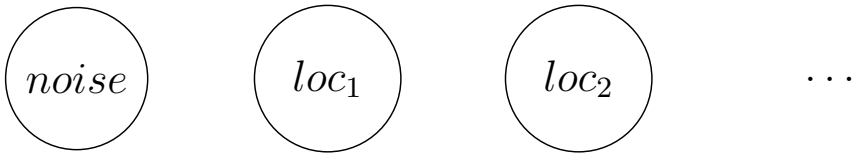


Figure 5.15: State space of our HMM. States correspond to previously detected significant places (cf. section 5.4)

The second factor that determines the structure of a HMM is the space of possible observations (cf. section 4.3). In our experiments we compare various discrete observations, which can be divided into two main topics. The first kind of observations concerns *location* information, and corresponds to discrete location labels, which we obtained from the detection of significant places. Secondly, we can incorporate different temporal factors, namely discrete information about the *date* and *time*.

In our experiments, we considered two different approaches to discretise a continuous date. The first possibility is to extract which day of the week it is, i.e. Monday up to Sunday. Secondly, with an even coarser granularity, we can differentiate between weekday and weekend, reducing the number of possible observations regarding the date to two. Note that people often tend to work within the week, and likely spend more time at home during weekends, which is why this could still carry a lot of useful information for the next place prediction model.

Similarly, we consider different granularities for the factor time. The most obvious choice is to take the exact hour of the day as an observation for our HMM. However, we can also split the hours of a day into a number of slots, so that we only consider every two, three, etc. hours. Figure 5.16 illustrates four different splits of a 24 hour day. Choosing 24 slots is equal to considering every hour of the day by its own; taking two slots, on the other hand, could be seen as feeding the HMM the information of whether it is day or night. One single slot, means we that have the same observation for every hour. In this case, our observations would not carry any information. Whenever 24 is not divisible by the defined number of slots, we obtain slot sizes that slightly differ, as is also illustrated in Figure 5.16.

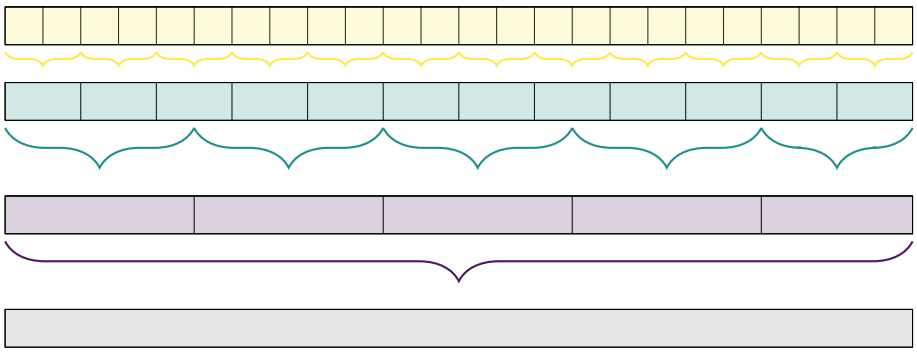


Figure 5.16: Visualisation of splitting the 24 hours of a day in 24, 12, 5 and 1 slots respectively

Table 5.7 depicts a summary of the different kinds of discrete observations we will consider. In addition to considering observations containing information about location, date or time alone, we can also combine multiple factors, in order to provide more knowledge for next place prediction. This could be seen as having multiple observation variables, i.e. one for every kind of information. Or, if we continue to assume a single observation variable, its value now consists of two or more concrete assignments as opposed to one assignment before. As an example, one particular observation at time t can be

$o^{(t)} = home$, as well as $o^{(t)} = home\ on\ monday$ or $o^{(t)} = home\ on\ monday\ at\ ten\ o'clock$. Note here, that we can also consider a total of 24 different numbers of slots as possible observations.

location	date	time
location label	day of the week weekday / weekend	hour (slot) of the day

Table 5.7: Possible discrete observations for our HMM

5.4.2 Comparing HMMs with Cross Validation

Once we know the structure of the HMMs we want to train, we need to get an idea of how well each one of them performs for the task of next place prediction. In section 2.3.10 we introduced the basic concepts of performance evaluation for HMMs. Particularly due to the small size of the dataset that was collected by us, we apply cross validation, as discussed in section 4.3.3.

Prior to the application of cross validation, the datasets have to be prepared. More precisely, we divide the data into multiple sequences, each containing records of one particular day (cf. section 4.3.3). The low amount of data per user restrains us from also considering longer sequences, such as weeks. To maintain a temporal order, the sequences of days, as opposed to the samples within a day, are shuffled, before they are split into $k = 4$ folds. Despite the restriction of using users which have at least seven recorded days of data available, the total number of days is very limited for various users in both datasets. Due to this, we chose not to keep apart a test set for assessing the quality of our model, and instead focus on finding the most suitable out of the different models we consider.

Our final goal is to maximise the *accuracy*, averaged over all users within a dataset. The accuracy is the relative number of correct predictions, i.e.

$$accuracy = \frac{number\ of\ correct\ predictions}{number\ of\ total\ predictions}. \tag{5.2}$$

In what follows, we will first look at the validation accuracy of different models on our collected dataset, followed by the experiments with the Geolife dataset.

5.4.3 Prediction Accuracy: Own Dataset (GPS only)

In this section, we compare the prediction accuracy of the different models suggested in section 5.4.1, on the dataset collected by us. Subsequent experiments for our dataset consist of two parts. As described in section 5.1.2, we recorded network data in addition to GPS data. First, we look at the results obtained by using GPS data only. We did not consider network data by itself, since it is too sparse and was not used by all participants. The second part of the experiments will use GPS and network data combined.

Table 5.8 contains the validation accuracy of multiple next place prediction HMMs, trained on our dataset with GPS data only. The left column lists observations that were fed to the different models, while the middle column shows the estimated prediction accuracies. Note here, that, regardless of what we observe, we only predict the likeliest next significant place. The column to the right in table 5.8 contains the standard deviation of the prediction accuracies, describing how the accuracy varies among different users. The number in parentheses specifies how many slots per day were considered (cf. Figure 5.16).

High validation accuracies in table 5.8 could be misleading. To get a better idea of how good different models actually are, we provide two basic baselines for the prediction accuracy. First, we consider a relatively simple model, that always predicts the same next place, namely *noise*. Since people do not necessarily have to spend the majority of their time at a significant place, a lot of observations are classified as noise. By always predicting noise as a next place, we get an accuracy of 0.85. This means that the majority of points is labelled as noise. On the one hand, this suggests that the accuracy as introduced in section 5.4.2 is not the ideal performance measure, as it does not consider this imbalance regarding different labels. However, due to the challenge of extending balanced accuracy to a multi-class prediction task, as well as the lack of valid predictions of our model as soon as new observations are encountered, we chose to continue using the accuracy as defined previously. On the other hand, the high amount of *noise* also gives rise to the question, whether or not this label has been assigned too frequently in the pre-processing routine. While we could reduce the overall amount of noise by introducing additional clusters (cf. [19]), this likely remains an issue as long as these clusters outweigh the clusters of interest, i.e. our significant locations.

model	mean accuracy	σ
locations	**0.958**	0.021
days	0.350	0.137
weekday / weekend (ww)	0.344	0.220
slot (1)	0.706	0.167
slot (24)	0.491	0.187
locations + days	0.699	0.240
locations + ww	0.944	0.035
locations + slot (1)	**0.958**	0.021
locations + slot (24)	0.814	0.125
days + slot (20)	0.492	0.141
ww + slot (24)	0.544	0.140
locations + days + slot (6)	0.743	0.159
locations + ww + slot (1)	0.944	0.035

Table 5.8: Mean prediction accuracies and their standard deviations for our collected dataset, considering GPS data only

As our second baseline, we consider how well a HMM performs when taking the input as its prediction. Using the identity function would be a naive solution for the task of next place prediction, which is why we want to achieve a higher prediction accuracy with our advanced HMM. Forwarding observations and using them as next place predictions, results in a mean accuracy of 0.989, with a standard deviation of 0.001. This means, that it predicts with a high accuracy for every user within the dataset.

Table 5.8 shows, that one of the simplest models, considering only the location label of previous time step in order to predict the next location label, reaches the highest accuracy with 0.958. This is higher than predicting solely *noise*, but not quite as high as our second baseline. Using the day of the week as an observation on the other hand results in a significantly lower accuracy of only 0.35. The information loss due to a much coarser granularity when only differentiating between weekday or weekend, however, does not influence the accuracy, now of 0.344, notably.

The best results of taking the time factor exclusively as an observation is achieved by a slot number of one. We reach a validation accuracy of 0.706, as can be seen in table 5.8. If we think back to how the number of slots within a day is defined (cf. section 5.4.1), this result is striking. If we split the hours within a day into one single slot, we obtain the same value for every observation we make, therefore not providing any information at all to the HMM. Splitting the day into 24 slots, on the other hand, lowers the accuracy to 0.491. The reason for the relatively low accuracies when using 24 slots or discrete date information, is likely the issue of encountering observations during testing that were not observed in the training process (cf. section 4.3.3). Whenever these observations occur during the testing phase, they are treated as a wrong prediction, as the HMM fails to predict validly. Note that the coarser granularity of weekday vs. weekend, as opposed to the day of the week, increases the chances for a measurement for each possible observation value. This might explain, why the accuracy does not drop significantly compared to using solely which day of the week we observe.

Table 5.8 also depicts the prediction accuracies achieved when combining multiple kinds of observations within a model. Adding further information to observed location labels leads to an often significantly reduced accuracy, once more due to a multitude of observations that are first seen in the validation, as opposed to the training, set. One example is considering the day of the week, next to location labels. Note here, that while the day of the week alone performed better than simply differentiating between weekday and weekend, combining latter with location information actually achieves considerably better accuracy. The reason for this is the coarser granularity, which reduces the amount of observation values that are first encountered during the validation step. Considering location labels and one slot together achieves the same high accuracy as the locations alone, as we do not actually observe any additional information. On the other hand, taking locations and the precise hour within a day as an observation, leads to a still relatively high next place prediction accuracy of 0.814. Leaving out location information, and instead combining the date and time factor in our observations, leads to accuracies of around 0.5. Once more the lower granularity, i.e. considering only weekday or weekend, influences the result positively, likely as it provides additional information without leading to too much unseen observations.

Finally, combining all three kinds of observations, does not lead to any further improvements. The highest accuracy we can achieve here is 0.945 with location labels, observing whether it is a weekday or weekend, and no additional time information due to a single split of the hours within a day.

It also should be noted, that the standard deviation of all prediction accuracies is relatively low, in particular for models that achieve a high mean accuracy. In other words, this means that well-performing models tend to do so for every user from our dataset.

5.4.4 Prediction Accuracy: Own Dataset (GPS & Network)

After considering GPS data alone in our previous experiments, we now take a look at how additional network data (cf. section 5.1.2) influences the accuracy of next place prediction. Table 5.9 summarises the achieved results here.

As for previous experiments, we want to compare our HMM with some baselines. In particular, it is interesting how the addition of this kind of data influences the performance of our model, and why it does so. The first factor we looked at in section 5.4.3 was the accuracy a model achieves when always predicting *noise* as its next place. In previous section 5.4.3, this model achieved a rather high accuracy of 0.85, suggesting that our dataset might consists of too much *noise* entries. For the dataset that combines GPS and network data however, the accuracy when predicting solely *noise* is 0.787. This is still a high prediction accuracy, as it performs better than a majority of the models we trained in our experiments. Nevertheless, the additional network data appears to reduce the noise in our dataset.

The other factor we considered, was the performance of a model that uses its input as the next place predictions. For previous experiments, this accuracy was as high as 0.989. When adding network information, the accuracy is comparable with 0.987, and a standard deviation of 0.003. The identity function thus solves our task of next place prediction with a relatively high accuracy for all users within the dataset, as the low standard deviation suggests.

As before, the model with location labels in its observation space performs best, and improves upon the model in the case of considering GPS data alone with an accuracy of 0.97 and a standard deviation of 0.016. While we perform significantly better with this model for next place prediction, than if we would predict *noise* for every future location, we still cannot outperform our second baseline, i.e. the identity function. Even though the performance of this model is improved with the addition of network data, this does not appear to apply to other combinations of observations. The majority of models now leads to lower prediction accuracies. Observing information regarding the date alone achieves accuracies significantly lower than 0.5. The information loss due to a coarser granularity of the sole differentiation between weekday and weekend is clearly more noticeable compared to previous experiments.

Looking at the performance of a HMM based on hour-slots as observations, once again a single slot achieves the highest prediction accuracy. The drop from 0.706 in previous experiments to 0.642 here indicates however, that the additional network data

model	mean accuracy	σ
locations	**0.970**	0.016
days	0.345	0.139
weekday / weekend (ww)	0.295	0.091
slot (1)	0.642	0.120
slot (24)	0.357	0.111
locations + days	0.777	0.160
locations + ww	0.944	0.041
locations + slot (1)	**0.970**	0.016
locations + slot (24)	0.860	0.092
days + slot (24)	0.442	0.128
ww + slot (24)	0.461	0.100
locations + days + slot (3)	0.777	0.160
locations + ww + slot (1)	0.944	0.041

Table 5.9: Mean prediction accuracies and their standard deviations for our collected dataset, considering GPS and network data combined

amplifies the importance for meaningful input. Remember that using one slot per day for observations results in the same observation value for every time step, i.e. providing no useful information for the model. Adding network data makes it harder to predict correct next places randomly, i.e. without meaningful input, as the relative amount of noise is decreased, which we could conclude from our first baseline. Using a model that observes the precise hour within a day achieves a lower accuracy as opposed to previous experiments as well. Once more, the low accuracy is likely due to the issue concerning new, unseen, observations.

When using HMMs with combinations of two or three different kinds of observations, all models which consider discrete locations once more exhibit significantly higher accuracies than the ones that do not observe them. The results are comparable to the performances we saw in previous experiments. Again, the highest accuracies are often achieved by disregarding the hour of the day, i.e. by taking a slot number of one. This suggests that for our data, wrong predictions due to unseen observations outweigh the benefits of additional information we would provide to the model.

The standard deviations remain relatively small for the experiments in this section. As before, the prediction accuracies from different users vary particularly little whenever we achieve a high overall accuracy.

Comparing the results of training a model with data originating solely from GPS, as opposed to GPS and network data combined, allows us to reason about the contributions of the additional data within our dataset. In general, GPS can take relatively long until enough satellites are found to determine an accurate location, which is why network data often has a higher sampling rate, i.e. leads to more data-points. The price to pay is less accurate locations. As stated previously, the decreased accuracy of a model which

only predicts noise suggests that the network data reduces the relative amount of *noise* within our records. In other words, network data tends to contribute a lot of data-points to significant places within a relatively short time span. This way, it does not necessarily provide us with observations on unseen days or time stamps, but rather stabilises the prediction of significant places, due to e.g. more observations of a particular location to learn from.

5.4.5 Prediction Accuracy: Geolife Dataset

Now that we have covered the experiments on our own dataset, we take a look at the performance of our models on the Geolife dataset (cf. section 5.1.1). As opposed to our own dataset, Geolife only provides GPS data for training and testing the HMMs. Due to the lack of ground-truth for significant locations, we use the same parameters for the extraction of significant places in both datasets. How this might influence the results of next place prediction will be discussed in this section.

Before taking a look at the prediction results, we want to determine the two baselines introduced previously. The first factor we looked at in sections 5.4.3 and 5.4.4 was the accuracy of predicting *noise* in every time-step. Previously, the high prediction accuracy of splitting a day into one single slot, and using this as the only possible observation value, already suggested that our dataset is highly unbalanced. Predicting *noise* as a next place in every time-step for Geolife leads to an accuracy as high as 0.999, which amplifies this assumption once more.

The second baseline is the prediction accuracy of a model predicting its input. For previous experiments, we achieved a performance measure of 0.989. When using the input as next place prediction for the Geolife dataset on the other hand, we reach an accuracy of 0.999. For both of the two baselines, the standard deviation is close to 0.

Table 5.10 summarises the prediction accuracies of the models we introduced in section 5.4.1. However, this time we tested the HMMs for next place prediction on the Geolife dataset. As for all other experiments, the model that only uses location labels as observations performs best for the task. With an accuracy of 0.948 it is comparable to previous results, especially when considering solely the GPS data of our own dataset. However, this final model does not predict as much next locations correctly as a HMM that would learn the identity function, or as the model that predicts *noise* in every time-step.

Using date-related observations alone achieves a relatively low prediction accuracy, similar to what we found previously. The discrepancy of the results between the two granularities of our observations is again relatively small. While the higher accuracy is now achieved with the coarser granularity, i.e. the differentiation between weekday and weekend, the increased standard deviation suggests that this level of abstraction might be suited better only for a fraction of the users represented in the Geolife dataset.

Also for the previous dataset, choosing a slot number of one, and therefore having a single observation value as the only input, results in a high accuracy. For the Geolife dataset, we raise the performance from an accuracy of 0.706 for our dataset, to 0.828. This gives an indication that what we observe with our trained model does not influence

model	mean accuracy	σ
locations	**0.948**	0.105
days	0.350	0.222
weekday / weekend (ww)	0.362	0.325
slot (1)	0.828	0.174
slot (24)	0.428	0.197
locations + days	0.765	0.230
locations + ww	0.907	0.146
locations + slot (1)	**0.948**	0.105
locations + slot (24)	0.775	0.201
days + slot (24)	0.478	0.170
ww + slot (24)	0.463	0.182
locations + days + slot (4)	0.771	0.220
locations + ww + slot (1)	0.907	0.146

Table 5.10: Mean prediction accuracies and their standard deviations for the Geolife dataset

the output significantly. In other words, our model often manages to correctly predict the next location of a user, despite the model not knowing which day of the week or time of the day it is, and also without information on previous whereabouts. This could mean that there are not a lot of different places to predict for a user, or that we simply have too much noise in our data-points.

Combining multiple kinds of observations in a model for next place prediction, influences the performance on Geolife similarly to what we saw in previous sections. Giving the HMM information about the current or past whereabouts seems to be a prerequisite to obtain prediction accuracies significantly over 0.5. As in previous experiments, the standard deviation of the achieved prediction accuracies suggest, that for models with a high overall accuracy, the performance between different users varies the least. Compared to the results we achieved with our dataset, a slight increase of the standard deviations is noticeable.

Achieving a high prediction accuracy with predicting the same next place, i.e. *noise*, for every time-step, does not necessarily mean that our final model is unsuited for the task. Once more, it should be noted, that the accuracy does not consider the imbalance within our data. While a different performance measure could give us a better idea of how various HMMs perform for the task, invalid predictions of models after encountering new observations complicate their usage. In addition to that, we should consider whether the annotations, resulting from our pre-processing routine, are fitting representations of the reality. For the Geolife dataset in particular, these results suggest that adopting the parameters found for our dataset has likely not been a suitable choice. The lack of ground-truth however, complicates finding proper parameters for the pre-processing step considerably.

6 Summary and Conclusion

Processing location traces has various applications and motivations, reaching from content improvements for context-aware systems (cf. [12]) to aiding in the authentication procedure of users (cf. [19]). In this thesis, we trained a HMM for the task of next place prediction. The performance of models is compared based on two different datasets. The first one is the existing Geolife GPS trajectory dataset by Microsoft [33]. As a second source of data, we collected our own dataset with the help of a mobile application for Android and NFC tags. In addition to the GPS data that is also present in the Geolife dataset, we recorded location data based on WiFi and cell towers, if available.

The discrete state space of HMMs allows us to predict one out of a discrete set of possible future locations. However, the continuous nature of raw location data such as coordinates, requires us to pre-process the data prior to the prediction. In order to discretise continuous coordinates, we used the approach of Li et al. to extract stay-points in a first step [16]. These are then clustered to form significant locations (cf. [30]).

A lot of common clustering algorithms encounter issues when grouping spatial data for our task. In particular, we found the need to fix a number of clusters and a lack of support for noise to be troublesome for detecting significant locations. In our experiments, we therefore used the DBSCAN algorithm, and tested its performance on our dataset which comes with a ground-truth. The density-based clustering approach is designed with a focus on spatial data, and achieves the closest approximation of the real number of significant places out of multiple clustering algorithms. Different experiments applying DBSCAN with differently pre-processed data, emphasise the importance of a stay-point extraction, and the positive influence on normalisation.

After pre-processing, we can use the set of extracted significant places as the state space of our model for next place prediction. More precisely, the HMM we trained has one state for every significant place, and an additional state for noise, i.e. for all the data that has not been assigned to any significant location. The second factor defining the structure of our HMM is the observation space, i.e. the set of values that can be observed in the environment. The final model for next place prediction has an observation space corresponding to the result of our pre-processing step, which means we infer the next most probable location of a person based on their latest whereabouts.

The averaged validation accuracy of our final HMM for next place prediction, i.e. the relative number of correct predictions, is higher than 0.95 for all datasets in our experiments. Concretely, we looked at the Geolife dataset, as well as our collected dataset in two variations. One variation considered only the GPS data within our dataset, whereas the other variant consisted of both the GPS data and the location information derived from the network. Comparing the results for these two variants, we find that the additional information from the network helps to stabilise significant

places and reduces the relative amount of noise. At the same time, it does not appear to provide a lot of additional information over GPS, e.g. of days or points in time we do not know a lot about. Treating network data separately, e.g. in a HMM specifically trained on it, could provide better insights of its advantages and disadvantages. This, however, would require a higher amount of such kind of data.

Despite the high accuracy of our model for the task of next place prediction, we also discovered a high presence of noise in our desired predictions. In other words, a model that predicts *noise* as every future location works as well as, or even better than our next place prediction model. This leads to the conclusion that better data might be necessary to get meaningful results. In particular, our collected dataset should be expanded to contain more users and longer recording sequences, so that we can create a benchmark that allows to compare various methods.

6.1 Future Work

In the next place prediction approach presented in this thesis we still work with a lot of limitations. The main motivation behind our work is to have next place predictions of a user as a baseline for comparing it with real sensor measurements, which only have a limited confidence in the current location of a user. This would allow us to reason about the actual current location of a user, or even the likelihood of this person being currently impersonated.

However, so far we only looked at the typical, day-to-day routine of a person. Whenever this person does something unexpected or irregular, e.g. travelling to a foreign country during holidays, we do not want our system to fail. Instead, we need to consider how likely new, unexpected transitions are, e.g. by looking at factors such as the time it would take the legit user to reach the new destination.

Similarly, the significant places of a person are not necessarily immutable. As a concrete example, we can think of moving to another residence, therefore changing the significant place that represents *home*. The most straightforward way to treat this would be to re-start the training process of our model, including the pre-processing of the data of our user. Nevertheless, future work could also consider to learn incrementally, e.g. not only using new observations for reasoning, but also for further training of the next place prediction model (cf. [14]).

New observations are also sources for trouble when training and testing our model, as a discrete observation space in a HMM does not support previously unseen observations natively. Whenever we observed something new in our implementation, it is treated as a wrong next place prediction. However, we also mentioned that a more advanced approach to circumvent this issue is to e.g. assign observation probabilities larger than zero to unseen observations (cf. [19]). Another possible alternative to consider for the future would be to look for a more incremental approach to learn HMM parameters, as a way to incorporate new observations directly into the active learning process.

We also want to emphasise the importance of working on quality as well as quantity of the data for next place prediction in the future. While larger, existing datasets often do

not provide information such as ground-truth for the discretisation process, our collected dataset still exhibits a lot of limitations primarily due to its size. A longer time span of recording location data would e.g. simplify the process of finding significant places while making it more robust at the same time. More precisely, in order to find stay-point clusters, a higher minimum number of points could be used, therefore reducing significant places that correspond to arbitrary locations without deeper significance. Additionally, a higher number of recorded days would allow to use a test set for a more unbiased performance estimation of a next place prediction model.

In the future it might be also interesting to look at a broader range of sensors that allow us to record location data. In particular, it could be explored whether we gain information when we train different models for different sensors, as opposed to using all data to train the same model, as we tried in our experiments. In order to obtain a final next place prediction, the models representing the prediction of a particular sensor could be combined based on the confidence in the sensors, e.g. the precision of their measurements.

In our experiments, we could include even more measurements in our data. As an example, also the altitude of a location point could be considered in the future. This would be interesting in particular if a person has significant places that can only be differentiated based on the altitude of their location. Furthermore, also the precision of a recorded location could be of interest. It could, for example, be used to compute a level of confidence in the measurements of a sensor, as suggested when using multiple different sensors for the task of next place prediction.

Finally, differently structured HMMs or completely different models that e.g. allow predicting specific coordinates could be considered for the prediction of future locations. While ANNs, as an example, would make the pre-processing step for a discretisation of location data redundant, models such as the HMM might be preferable in a security-based application of next place prediction, as the base of decisions should be interpretable in this setting. In addition to that, the pre-processing step itself could be adapted. So far, we only have one state for noise. However, multiple additional states could be taken into account, as e.g. proposed by Mahbub and Chellappa [19]. Examples would be states for *unknown* data samples, or a *transition* state for locations a person only visited during transit. Also, continuous as opposed to discrete observations could be considered in the future.

Acronyms

ANN Artificial Neural Network.

BN Bayesian Network.

CDR Call Detail Record.

CPD Conditional Probability Distribution.

DBN Dynamic Bayesian Network.

DBSCAN Density Based Spatial Clustering of Applications with Noise.

GMM Gaussian Mixture Model.

GPS Global Positioning System.

HMM Hidden Markov Model.

kNN K-Nearest Neighbours.

MCMC Markov Chain Monte Carlo.

MMC Mobility Markov Chain.

MSE Mean Squared Error.

NFC Near Field Communication.

NN Neural Network.

RNN Recurrent Neural Network.

RSS Received Signal Strength.

Bibliography

[1] Sherif Akoush and Ahmed Sameh. 'Mobile User Movement Prediction using Bayesian Learning for Neural Networks'. In: *Proceedings of the 3rd ACM International Conference on Wireless Communications and Mobile Computing*. ACM. 2007, pp. 191–196.

[2] Theodore Anagnostopoulos, Christos Anagnostopoulos and Stathes Hadjiefthymiades. 'Efficient Location Prediction in Mobile Cellular Networks'. In: *International Journal of Wireless Information Networks* 19.2 (2012), pp. 97–111.

[3] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[4] Jesús Bobadilla et al. 'Recommender Systems Survey'. In: *Knowledge-Based Systems* 46 (2013), pp. 109–132.

[5] Yohan Chon et al. 'Evaluating Mobility Models for Temporal Prediction with High-Granularity Mobility Data'. In: *Proceedings of the 10th IEEE International Conference on Pervasive Computing and Communications*. IEEE. 2012, pp. 206–212.

[6] Alexandre De Brébisson et al. 'Artificial Neural Networks applied to Taxi Destination Prediction'. In: *Proceedings of the 2015 International Conference on ECML PKDD Discovery Challenge*. CEUR-WS.org. 2015, pp. 40–51.

[7] Martin Ester et al. 'A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.' In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*. Vol. 96. 34. AAAI Press, 1996, pp. 226–231.

[8] Brendan J Frey and Delbert Dueck. 'Clustering by Passing Messages between Data Points'. In: *Science* 315.5814 (2007), pp. 972–976.

[9] Lex Fridman et al. 'Active Authentication on Mobile Devices via Stylometry, Application Usage, Web Browsing, and GPS Location'. In: *IEEE Systems Journal* 11.2 (2017), pp. 513–521.

[10] Sébastien Gambs, Marc-Olivier Killijian and Miguel Núñez del Prado Cortez. 'Next Place Prediction using Mobility Markov Chains'. In: *Proceedings of the 1st ACM Workshop on Measurement, Privacy, and Mobility*. ACM. 2012, p. 3.

[11] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[12] Jeffrey Hightower and Gaetano Borriello. 'Particle Filters for Location Estimation in Ubiquitous Computing: A Case Study'. In: *Proceedings of the 6th International Conference on Ubiquitous Computing*. Springer. 2004, pp. 88–106.

[13] Sibren Isaacman et al. 'Identifying Important Places in People's Lives from Cellular Network Data'. In: *Proceedings of the 9th International Conference on Pervasive Computing*. Springer. 2011, pp. 133–151.

[14] Wael Khreich et al. 'A Survey of Techniques for Incremental Learning of HMM Parameters'. In: *Information Sciences* 197 (2012), pp. 105–130.

[15] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[16] Quannan Li et al. 'Mining User Similarity based on Location History'. In: *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2008, p. 34.

[17] Qiujian Lv et al. 'Spatial and Temporal Mobility Analysis in LTE Mobile Network'. In: *Proceedings of the 2015 IEEE Wireless Communications and Networking Conference*. IEEE. 2015, pp. 795–800.

[18] David JC MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[19] Upal Mahbub and Rama Chellappa. 'PATH: Person Authentication using Trace Histories'. In: *Proceedings of the 7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*. IEEE. 2016, pp. 1–8.

[20] Wesley Mathew, Ruben Raposo and Bruno Martins. 'Predicting Future Locations with Hidden Markov Models'. In: *Proceedings of the 14th ACM Conference on Ubiquitous Computing*. ACM. 2012, pp. 911–918.

[21] Mendhak. *Lightweight GPS Logging Application For Android*. `https://github.com/mendhak/gpslogger`. 2018.

[22] Navigation National Coordination Office for Space-Based Positioning and Timing. *GPS.GOV GPS Educational Poster*. 2016. URL: `https://www.gps.gov/multimedia/poster/` (visited on 08/08/2018).

[23] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[24] Andrei D Polyanin and Alexander V Manzhirov. *Handbook of Mathematics for Engineers and Scientists*. Chapman and Hall / CRC, 2006.

[25] Yuanyuan Qiao et al. 'A Hybrid Markov-based Model for Human Mobility Prediction'. In: *Neurocomputing* 278 (2018), pp. 99–109.

[26] Lawrence R Rabiner. 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition'. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[27] Lawrence R Rabiner and Biing-Hwang Juang. 'An Introduction to Hidden Markov Models'. In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16.

[28] Jacob Schreiber. 'Pomegranate: Fast and Flexible Probabilistic Modeling in Python'. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 5992–5997.

[29] Dongkuan Xu and Yingjie Tian. 'A Comprehensive Survey of Clustering Algorithms'. In: *Annals of Data Science* 2.2 (2015), pp. 165–193.

[30] Jie Yang et al. 'Predicting Next Location using a Variable Order Markov Model'. In: *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming*. ACM. 2014, pp. 37–42.

[31] Yingxiang Yang et al. 'Mobility Sequence Extraction and Labeling Using Sparse Cell Phone Data.' In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI. 2016, pp. 4276–4277.

[32] Josh Jia-Ching Ying et al. 'Semantic Trajectory Mining for Location Prediction'. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2011, pp. 34–43.

[33] Yu Zheng et al. *Geolife GPS Trajectory Dataset - User Guide*. July 2011. URL: `https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/`.

[34] Changqing Zhou et al. 'Discovering Personally Meaningful Places: An Interactive Clustering Approach'. In: *ACM Transactions on Information Systems* 25.3 (2007), p. 12.

[35] Mu Zhou et al. 'SCaNME: Location Tracking System in Large-Scale Campus Wi-Fi Environment using Unlabeled Mobility Map'. In: *Expert Systems with Applications* 41.7 (2014), pp. 3429–3443.

Katharina Prinz

Personal

Birthplace	Steyr, Austria	Birthday	23-07-1994
Marital Status	Single	Nationality	Austrian

Education

2016–now **Master's degree**, *Johannes Kepler University*, Linz, Austria.
Computer Science - Computational Engineering

2013–2016 **Bachelor of Science**, *Johannes Kepler University*, Linz, Austria, *mit Auszeichnung bestanden.*
Informatics

2008–2013 **High School**, *Höhere Bundeslehranstalt für Wirtschaftliche Berufe*, Steyr, Austria, *mit Auszeichnung bestanden.*
Health Management

Working Experience

Relevant Student Jobs

2017 / 2018 **Student Employee Institute of Networks and Security**, *INS Institute at JKU*, Linz.
Working on my thesis

2017 **Tutor Computer Graphics Institute**, *CG Institute at JKU*, Linz.
Tutor for Computer Graphics labs

2016 / 2017 **Tutor Institute of System Software**, *SSW Institute at JKU*, Linz.
Tutor for Compiler Construction class

2016 **Student Employee Institute of System Software**, *SSW Institute at JKU*, Linz.
Working on Inline Assembly support for Sulong

Miscellaneous Jobs

summer 2016
2015
2014 **Office Employee**, *Red Cross*, Steyr.
Holiday replacement

summer 2011 **Intern**, *Klinikum Bad Hall*, Bad Hall.
Three-month internship as part of my high school education

prinz.katharinamaria@gmail.com

Languages

German	Native	
English	Advanced or upper intermediate	*Cambridge English: Advanced (CAE)*
Italian	Elementary or pre-intermediate	*Third foreign language in high school*
Dutch	Pre-intermediate	*Learning by doing*
French	Beginner	*Basic course*

Statutory Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

This printed thesis is identical with the electronic version submitted.

Linz, February 2019 Katharina Prinz