Contents

List of Algorithms

List of Figures

List of Tables

Distributed Spatio-temporal Prediction

Master Thesis

Rufat Babayev

Abstract. Precise and detailed prediction of the user location has many potential applications such as geographical user profiling, context-aware services and targeted advertising for mobile users. Because of the age of mobile computing and recent improvements on efficient storage mechanisms and cloud computing, mobile devices can store more data and can sync them with cloud storage if necessary. One of the crucial parts of data which is stored or exchanged is location history data. Since, location history data is large, location prediction techniques should be capable of processing it. There is an important demand to parallelize location prediction techniques for better runtime and predictive performance.

In this thesis, we present *Distributed NextPlace* which builds on top of the serial NextPlace and is a novel method for parallel user location prediction which considers spatial, temporal and semantic perspectives of user location history data. The serial NextPlace is a state-of-the-art algorithm based on nonlinear model and has an advantage for long-term estimations, detailed predictions and runtime performance as compared to other serial location prediction algorithms. In spite of improvements on long-term predictions, serial NextPlace does not provide promising results in terms of runtime performance on large datasets. In comparison, *Distributed NextPlace* paralellizes user location history data by maintaining its statistical stability, improves predictive performance by few adjustments, and extends serial NextPlace with support for the semantic background knowledge influencing the predictions. In our empirical study, we evaluate the strength and optimality of this method on various datasets.

1 Introduction

Predicting future location of mobile users has become an important and challenging task. With the recent advancements in mobile and wireless computing, applications of location prediction include traffic planning, location-targeted advertising, intelligent resource allocation, as well as context-aware services. With the help of new techniques, these applications can be implemented in a more efficient way. In location prediction tasks, efficiency is both speed and accuracy of generating the predictions. For example, geo-targeting is the practice of delivering appropriate content to users through mobile or web by using geographical profile of users. In this domain, there are different aspects for targeting the mobile users, but location often provides much detailed, more meaningful and distinguishable properties that signify what a person wants, needs or is interested in. Therefore, improved location prediction speed and accuracy means improved targeting for such domains.

Mobile devices are equipped with GPS, WiFi hardware and GSM, CDMA technologies which enables location-based computing to increase in popularity. User location can be estimated through GPS sensors, cell tower location estimations and nearby WiFi access points in case of connectivity. Collected location information can be sent to processing servers; where through statistical analysis future location estimates can be generated.

With this in mind, we present *Distributed NextPlace* which adds a parallelization to the serial NextPlace prediction framework. The original NextPlace is based on nonlinear time series analysis [Kantz and Schreiber, 2004] for predicting future user locations from spatial and temporal aspects. It looks for determinism in patterns of visits of users to their significant locations. One property of it is that it does not focus on transitions between different locations. For its second property, it is not only able to predict the next location of a user, but also locations further in the future. As a third property, it can generate detailed predictions; namely arrival time and residence time (time of stay) of future location predictions. *Distributed NextPlace* takes advantage of all these properties. We also study how additional semantic information can influence the predictions generated by the Distributed NextPlace.

As in the original NextPlace [Scellato et al., 2011], we focus on visit patterns of users which belong to significant locations. Significant locations are characterized by frequency of visits to them and time of stay in them. We demonstrate that such patterns of visits can be utilized by nonlinear models for effective future location predictions. Predictions of *Distributed NextPlace* are generated from movement data of mobile users which are either sequence of GPS coordinates or registration logs to WiFi access points. Movement data or location history data of a user represents time series. *Distributed NextPlace* deals with data of multiple users. Straightforwardly, it processes time series of each user in parallel. This provides improvement on runtime performance which is up to the magnitude of the number of users.

The presented method is composed of three operations. Firstly, it identifies significant locations which users visit frequently and stay there for a reasonable amount of time. Secondly, by the help of nonlinear time series analysis [Kantz and Schreiber, 2004] *Distributed NextPlace* generates a sequence of predictions to significant locations along with arrival time and residence time. Finally, we utilize semantic information of locations to refine predictions and compare the results. Summary of steps:

- We explain *Distributed NextPlace* which is a novel method for parallel user location prediction utilizing nonlinear time series of analysis of the history data of significant locations.

- We examine four movement datasets: Cabspotting [Piorkowski et al., 2009], CenceMe

GPS [Musolesi et al., 2008], Dartmouth WiFi [Kotz et al., 2009] and Ile Sans Fils [Lenczner et al., 2007] First two of these datasets are GPS datasets and remaining ones are WiFi datasets. We preprocess these datasets by generating sequence of visits and by extracting significant locations.

- We analyze the underlying time series of the history of visits of users to different significant locations in terms of predictability. We further examine different distance functions and number of users which are affected by different configurations. After that, we apply nonlinear model to identify regularities in the patterns of visits to significant locations.

- We perform empirical study of *Distributed NextPlace* by comparing it to serial NextPlace [Scellato et al., 2011] in terms of runtime and predictive performance on four datasets.

- We demonstrate how different adjustments and utilizing the semantic meaning of locations can improve prediction accuracy.

Organization of the rest of this thesis is as follows: Section 2 provides preliminary information for the nonlinear time series analysis. Section 3 explains *Distributed NextPlace*, its parallelization scheme for location history data and significant location extraction techniques for GPS and WiFi datasets. Section 4 describes practical issues and empirical evaluation of *Distributed NextPlace* on proposed datasets by comparing it to the serial NextPlace. Section 5 presents related work and Section 6 concludes the thesis and discusses future work.

2 Preliminaries

In this section, we briefly discuss a prediction technique [Scellato et al., 2011] based on nonlinear time series analysis [Kantz and Schreiber, 2004] and its empirical implementation challenges. The outline is as follows:

- We start by explaining the representation of the given time series in the nonlinear analysis framework. The time series is embedded in an embedding space with appropriate configuration parameters.

- Then we describe how the neighborhood search is performed in the given time series. For this purpose, we provide ϵ-neighborhood and k-nearest neighbors techniques and give their asymptotic complexities.

- Finally, we show how the prediction is generated from the elements in the neighborhood.

2.1 Embedding Spaces

A time series is a sequence of data points or scalar observations of a given system recorded in time order. Data points can be positioned in uniform or non-uniform time intervals [Chatfield, 2016].

A sequence of scalar observations may seem completely random at a first glance, but analysis of sub-sequences can unveil important characteristic patterns. Sub-sequencing operation is a transformation of time series into sequence of vectors named *delay embedding*.

In short, a time series $(s_0, s_1, \ldots, s_N)$ is embedded to m-dimensional vectors using suitable delay parameter ν. This creates an embedding vector for the time series value s_n for $\forall n \in \{(m-1)\cdot\nu, (m-1)\cdot\nu+1, \ldots, N\}$ as follows [Scellato et al., 2011]:

$$\beta_n = (s_{n-(m-1)\cdot\nu}, s_{n-(m-2)\cdot\nu}, \ldots, s_{n-\nu}, s_n)$$

where all embedding vectors β_n have m components (m-th component is s_n). All embedding vectors form a so called *embedding space* where N is the index of the last embedding vector in the created space. Note that, number of embedding vectors in an embedding space is exactly $N+1-(m-1)\cdot\nu$, because scalar values of time series $(s_0, s_1, \ldots, s_N)$ have a starting index of 0. Assume that, time series $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$ is given where $N = 10$ and $m = 3$, $\nu = 1$. Then, embedding space are constructed by the following $N+1-(m-1)\cdot\nu = 10+1-(3-1)\cdot 1 = 9$ embedding vectors: $\beta_2 = (s_0, s_1, s_2)$, $\beta_3 = (s_1, s_2, s_3)$, $\beta_4 = (s_2, s_3, s_4)$, $\beta_5 = (s_3, s_4, s_5)$, $\beta_6 = (s_4, s_5, s_6)$, $\beta_7 = (s_5, s_6, s_7)$, $\beta_8 = (s_6, s_7, s_8)$, $\beta_9 = (s_7, s_8, s_9)$, $\beta_{10} = (s_8, s_9, s_{10})$.

Embedding parameter m and delay parameter ν have an influence on the precision of embedding construction. According to the embedding theorem [Kantz and Schreiber, 2004], there exists an appropriate m for the given system which represents its underlying complexity. Delay parameter ν is used to skip some values from original time series in embedding space construction, because sometimes successive values in time series do not have strong correlation.

We can generate suitable predictive model through delay embedding in the nonlinear time series analysis. Suppose that, predictions need to be generated for time series $(s_0, s_1, \ldots, s_N)$. Last value of time series is s_N, therefore, predictions which are Δl steps ahead of s_N should be generated, namely $s_{N+\Delta l}$ where $\Delta l = 1, 2, 3, \ldots$ is applied to the index. The prediction process is composed of following steps:

1. With the appropriate delay parameter ν and the embedding parameter m, the time series is transformed to embedding vectors which constitute the embedding space. Here is the algorithm for embedding space generation from time series $(s_0, s_1, \ldots, s_N)$:

Algorithm 1 Embedding Space Generator

Input: Time series $(s_0, s_1, \ldots, s_N)$, dimension $m \geq 1$, delay $\nu \geq 1$
Output: Embedding space vector α

```
 1: create empty vector $\alpha$
 2: for $n = 0, \ldots, N$ do
 3:     if $n \geq (m-1) \cdot \nu$ then
 4:         create empty embedding vector $\beta_n$
 5:         for $i = 1, \ldots, m$ do
 6:             add $s_{n-(m-i)\cdot\nu} \rightarrow \beta_n$
 7:         end for
 8:         add $\beta_n \rightarrow \alpha$
 9:     end if
10: end for
11: return $\alpha$
```

Its asymptotic runtime is $\mathcal{O}(m \cdot N)$ which is roughly $\mathcal{O}(N)$, since $m \ll N$. We use this algorithm in our location prediction method. We choose $\nu = 1$ as in the original NextPlace [Scellato et al., 2011], because we utilize only the time-of-day information of user visits and we do not want to skip any of them.

2. The created embedding space is explored for all vectors which are close to the embedding vector β_N according to the given distance metric. This will create a neighborhood of β_N. There are two well-known techniques for the neighborhood search in embedding spaces: ϵ-neighborhood and k-nearest neighbors.

   * The ϵ-neighborhood is straightforward. The neighborhood $U_\epsilon(\beta_N)$ is constructed from vectors in the embedding space whose distance to β_N is up to predefined ϵ [Scellato et al., 2011]. According to [Kantz and Schreiber, 2004], reasonable value for ϵ is 10% of standard deviation of the given time series. The time complexity for ϵ-neighborhood search is $\mathcal{O}(N)$; distances between β_N and the other embedding vectors are calculated in $\mathcal{O}(m \cdot N)$ time, value of ϵ is found in $\mathcal{O}(m \cdot N)$ time (where $m \ll N$), and the embedding space is searched in $\mathcal{O}(N)$ time. Overall, the time complexity is $\mathcal{O}(2 \cdot m \cdot N + N)$ which is roughly $\mathcal{O}(N)$.

   * It is possible to obtain k-nearest neighbors with $\mathcal{O}(N)$ runtime using algorithmic improvement [josliber]:

     – Compute distance in advance between β_N and each embedding vector in the embedding space. Runtime complexity of this step is $\mathcal{O}(m \cdot N)$.

     – Run randomized *QuickSelect* algorithm to find k-th smallest distance in $\mathcal{O}(N)$ runtime.

     – Create neighborhood $U_{knn}(\beta_N)$ from embedding vectors whose distance is not larger than the computed k-th smallest distance. Runtime of this step is still $\mathcal{O}(N)$.

   So, in total it is $\mathcal{O}(m \cdot N + 2N)$ which is $\mathcal{O}(N)$.

The main computational effort is performed in this step. The two methods presented above are computationally efficient in this respect.

3. Capturing patterns is a prediction of future occurrences from past occurrences in the time series. All embedding vectors $\beta_n \in U_{\cdot}(\beta_N)$ characterizes past occurrences similar to β_N, then predicted future occurrence $p_{N+\Delta l}$ (which is the prediction for $s_{N+\Delta l}$) is calculated as average of values of past occurrences $s_{n+\Delta l}$ (which are the past scalar values those are Δl steps ahead of past scalar values s_n):

$$p_{N+\Delta l} = \frac{1}{|U_{\cdot}(\beta_N)|} \sum_{\substack{s_n \in \beta_n \\ \forall \beta_n \in U_{\cdot}(\beta_N)}} s_{n+\Delta l}$$

where $|U_{\cdot}(\beta_N)|$ is the number of embedding vectors in the neighborhood $U_{\cdot}(\beta_N)$.

The proposed prediction process above searches the time series for past history of occurrences which are similar to recent ones. If dynamic evolution of time series contains determinism and characteristic patterns, then based on similarity to past states, similar state will come after the given recent state.

We continue our explanation with the prediction of spatio-temporal attributes of users in the next section where we utilize the prediction process described in this section.

3 Prediction of Spatio-temporal Attributes of Users

Prediction of future user actions is a challenging problem. It is based on assumption of characteristic patterns in past user actions. Making a prediction by an assumption of patterns is to find a specific state in past actions which can be a model for future actions. Daily, weekly and/or monthly activities of humans are more or less the same in communities so that there is a specific level of order and characteristics for the predictability [Gonzalez et al., 2008].

Both the original NextPlace [Scellato et al., 2011] and Distributed NextPlace are nonlinear location prediction methods and they use the concept of determinism. The idea is that series of locations that user visits every day is fixed in general and in case of small changes in user behavior those changes can be captured deterministically by nonlinear models [Kantz and Schreiber, 2004]. For example, a man goes to yoga classes on Mondays, Wednesdays and Fridays in a repeating manner. He may decide to change the schedule to other days. However, the new schedule will be similar over different time spans such as weeks and/or months. So, the series of visit patterns may still be captured for future prediction.

Formally, let us assume a set of mobile users, where user u simply visits different locations according to his routine. Our focus here is not to characterize the favorite locations that a user visits. Instead, we suppose that arrival time and residence time for a specific location can be identified; when a visit is performed to that location then arrival time is the start time of this visit and residence time is the duration of this visit respectively. Arrival time and residence time can represent one instant of a time series in this regard. In a broader sense, we model building blocks of the nonlinear location prediction method as follows:

- Location l is a tuple of (n, lat, lng), where n is the name, lat is latitude and lng is longitude respectively.
- Arrival time t and residence time d form $s = (t, d)$ which is one instant of a time series, where t is chronological date/time and d is the number of seconds. It is possible to obtain instant $\hat{s} = (c, d)$ where c represents number of day seconds (in the interval of [0, 86400]) of chronological date/time t. Note that we use the same s notation proposed in Section 2 to represent one value of a time series. The index of s is the corresponding chronological position in the given time series (namely in the underlying time series of the list of visits).
- Visit v of a user is portrayed by $(u, l, s) \rightarrow (u, l, t, d)$ tuple which models visit of user u to location l with arrival time t and residence time d. A chronological list of visits to different locations models the movements of a user.
- Visit history Ψ of a user u is chronologically ordered vector of visits performed to the same location. For example, $\Psi^{(l)} = \{v_1^{(l)}, v_2^{(l)}, \ldots, v_n^{(l)}\}$ is visit history of a user for the location l, where $\forall i \in \{1, \ldots, n\}$, $v_i^{(l)}$ defines i-th visit to the location l in visit history $\Psi^{(l)}$. The collection of visit histories is used to capture regularities and patterns for future predictions.

We now describe theory and practices for three steps of our nonlinear prediction method. Firstly, we show practices for preprocessing of timestamped GPS data and WiFi connectivity logs to generate sequence of visits to various locations and how we manipulate these visits to extract significant locations using a technique presented by [Kim et al., 2006]. Secondly, we demonstrate how a nonlinear location prediction method is used for the prediction of future arrival and residence times in many significant locations and define how these predictions can be used to determine where the user will be after certain time interval. Finally, we talk about semantic meaning for location predictions and how we integrate it to our method to improve predictions.

3.1 Obtaining Meaningful Sequence of Visits from The Recorded Visits

In timestamped GPS readings and WiFi connectivity messages, discovering arrival time t and residence time d is straightforward. While GPS readings contain timestamps and related GPS coordinates, WiFi connectivity messages contain anonymized WiFi locations (in most cases) along with the timestamps of the messages. Assume that each GPS reading or WiFi connectivity message represent one visit, then related timestamp can be chosen as a start time (arrival time) of the visit. In this case, duration of the visit (residence time) can be obtained by finding a difference between two consecutive timestamps. Practically, GPS measurement operation requires periodic sampling of GPS coordinates. When the user stays within the same area for a long period, this makes several sequential short-lived visits being registered whose duration is subject to a chosen GPS sampling interval. The same sampling problem can emerge in WiFi connectivity messages. Depending on hardware capabilities, availability of WiFi connectivity can be periodical, therefore through WiFi handoff procedures a long period of stay can be divided to different mini-periods.

To induce more precise residence time (time of stay) of a user in particular areas, we utilize a merging mechanism to a sequence of original recorded visits. Given a sequence of visits $v_1 = (u, l, t_1, d_1), v_2 = (u, l, t_2, d_2), \dots, v_n = (u, l, t_n, d_n)$ of a user u to the same location l. In that case, $\forall i \in \{1, \dots, n-1\}$, if an end time of a visit $v_i = (u, l, t_i, d_i)$ is adjacent to a start time of a next visit $v_{i+1} = (u, l, t_{i+1}, d_{i+1})$, i.e. $t_{i+1} - (t_i + d_i) \leq \delta$ (where δ is a chosen sampling interval), then v_i and v_{i+1} is merged to a new visit $v_i = (u, l, t_i, t_{i+1} - t_i + d_{i+1})$ which starts at t_i and ends at $t_{i+1} + d_{i+1}$. The following algorithm achieves a proposed merging mechanism:

Algorithm 2 Visits Merger

```
 1: function MERGE(Vector $\varphi$ of unprocessed visits of user $u$, threshold $\delta$)
 2:     create empty vector $\tilde{\varphi}$ of merged visits
 3:     add first unprocessed visit $\varphi_1 \to \tilde{\varphi}$
 4:     for $j = 2, \dots, |\varphi|$ do
 5:         if $\varphi_j.l = \tilde{\varphi}_{|\tilde{\varphi}|}.l \wedge \varphi_j.t - (\tilde{\varphi}_{|\tilde{\varphi}|}.t + \tilde{\varphi}_{|\tilde{\varphi}|}.d) \leq \delta$ then
 6:             update $\tilde{\varphi}_{|\tilde{\varphi}|} \leftarrow$ COMBINE($\tilde{\varphi}_{|\tilde{\varphi}|}$, $\varphi_j$)
 7:         else
 8:             update $\tilde{\varphi}_{|\tilde{\varphi}|}.d \leftarrow \varphi_j.t - \tilde{\varphi}_{|\tilde{\varphi}|}.t$
 9:             add $\varphi_j \to \tilde{\varphi}$
10:         end if
11:     end for
12:     return $\tilde{\varphi}$
13: end function

14: function COMBINE(Visit $v_1$, Visit $v_2$)
15:     if $v_1$ before $v_2$ then
16:         create merged visit $\tilde{v} \leftarrow (u, v_1.l, v_1.t, v_2.t - v_1.t + v_2.d)$
17:     else
18:         create merged visit $\tilde{v} \leftarrow (u, v_2.l, v_2.t, v_1.t - v_2.t + v_1.d)$
19:     end if
20:     return $\tilde{v}$
21: end function
```

The MERGE function receives vector φ of unprocessed visits of a given user u and sampling interval threshold δ in seconds. Vector φ is assumed to be chronologically ordered. In general, visits contained in φ have 0 residence time, because they just represent recorded timestamps. In line 3, algorithm adds first unprocessed visit φ_1 to $\tilde{\varphi}$, because it is a starting point for the first merged visit. Then, for all remaining visits in φ, algorithm performs merge operation if necessary by comparing them to visits which are already merged. Between lines 5-6, if next unprocessed visit φ_j has the same location as the last merged visit and their time difference is within the interval of δ, algorithm combines φ_j and $\tilde{\varphi}_{|\tilde{\varphi}|}$ to the new visit $\tilde{v}$ and updates $\tilde{\varphi}_{|\tilde{\varphi}|}$ with it. COMBINE is general-purpose function which combines visits based on their chronological ordering. Lines 8 and 9 are simple; if visits are not performed to the same location, then residence time of last merged visit is updated by its time difference with the next unprocessed visit.

Crucial point in location comparison is to define equality of locations. As presented earlier, location l is a tuple of (n, lat, lng), where n is the name, lat is latitude and lng is longitude. Latitude and longitudes are double precision floating point numbers. We use at least 5 decimal places after a decimal point which provides accuracy of 1.11 meters according to degree precision[1] In this case, two locations are equal if their names are equal, and their IEEE 754 floating-point "double format" bit layouts [Kahan, 1996] are the same. For example, locations such as (31.89452, -12.22945) and (31.89451, -12.22945) are not equal, since latitudes differ on one decimal place. WiFi locations we process do not have the specific latitude and longitude, they only have unique names. We explicitly set their latitude and longitude to -1.0 and they are compared with respect to their names.

Now, we show examples of proposed merging mechanism with merge operator $\oplus$. This operator is equivalent to the COMBINE function. For simplicity, we only consider time series instant of each visit, namely $\hat{s} = (c, d)$ of $s = (t, d)$ where c is day seconds. Assume that, $\delta = 60$ seconds:

(1) $v_i.l = v_{i+1}.l \Rightarrow \hat{s}_i = (c_i, d_i) = (10, 0) \oplus \hat{s}_{i+1} = (c_{i+1}, d_{i+1}) = (50, 0) \rightarrow \hat{s}_i = (c_i, c_{i+1} - c_i + d_{i+1}) = (10, 40)$.

(2) $v_i.l \neq v_{i+1}.l \Rightarrow \hat{s}_i = (c_i, d_i) = (10, 0) \oplus \hat{s}_{i+1} = (c_{i+1}, d_{i+1}) = (100, 0) \rightarrow [\hat{s}_i = (c_i, c_{i+1} - c_i) = (10, 90), \hat{s}_{i+1} = (c_{i+1}, d_{i+1}) = (100, 0)]$.

(3) $v_i.l = v_{i+1}.l = v_{i+2}.l = v_{i+3}.l \Rightarrow$

$\hat{s}_i = (c_i, d_i) = (10, 50) \oplus \hat{s}_{i+1} = (c_{i+1}, d_{i+1}) = (100, 40) \rightarrow$

$\hat{s}_i = (c_i, c_{i+1} - c_i + d_{i+1}) = (10, 130)$;

$\hat{s}_i = (c_i, d_i) = (10, 130) \oplus \hat{s}_{i+2} = (c_{i+2}, d_{i+2}) = (160, 30) \rightarrow$

$\hat{s}_i = (c_i, c_{i+2} - c_i + d_{i+2}) = (10, 180)$;

$\hat{s}_i = (c_i, d_i) = (10, 180) \oplus \hat{s}_{i+3} = (c_{i+3}, d_{i+3}) = (230, 70) \rightarrow$

$\hat{s}_i = (c_i, c_{i+3} - c_i + d_{i+3}) = (10, 290)$.

This mechanism of handling visits creates merged visits which are more promising to define patterns of availability of users in certain locations, hence enhancing our prediction method.

[1] http://im.lternet.edu/sites/im.lternet.edu/files/BestPractices-2010.pdf

3.2 Extraction of Significant Locations

We present two ways of extracting significant locations from GPS and WiFi connectivity data which are widely present sources for movements of mobile users.

3.2.1 Extracting Locations from GPS data

Different methods to extract significant locations from GPS readings have been proposed in several research papers [Ashbrook and Starner, 2003, Kang et al., 2004, Zhou et al., 2007]. For example, [Ashbrook and Starner, 2003] clusters GPS points with residence time of at least t into significant locations. They utilize the clustering algorithm similar to k-means such that no GPS point gets unclustered. The paper by [Kang et al., 2004] performs the clustering of GPS points into significant locations which can be executed in a mobile device. They do not use the traditional clustering algorithms in their approach because of performance issues. Instead, they use time-based clustering such that each GPS point is assigned to clusters in a time order. If a GPS point has a distance which is up to threshold d to one of the existing clusters, then it is assigned to that cluster, otherwise new cluster is created. At the end, if a cluster's time duration is above time threshold t, then that cluster is considered as a significant location. Finally, [Zhou et al., 2007] provides the density-based algorithm DJ-Cluster which is similar to DBSCAN [Ester et al., 1996] to cluster GPS points into significant locations. However, they do not weigh the GPS points according to their residence times. DJ-Cluster can avoid memory issues emerged from DBSCAN, if large number of GPS points are clustered. However, clustering results of DJ-Cluster and DBSCAN can be different, since the former can join multiple clusters together [Zhou et al., 2007].

Our choice is the extraction method which considers residence time of a user at locations; the idea is that persistence at a location signifies the importance of it to the user. As compared to other approaches, we do not fix specific residence time threshold t to attribute the significance of a GPS point for a user, because some GPS points can even be significant if the duration is less than t or they cannot have any statistical importance (e.g. outlier) even they have very large residence time. Instead, we utilize the method proposed by [Kim et al., 2006] such that 2-D Gaussian distribution weighted by residence time of each GPS point is employed to quantify the significance. We find the important GPS points and cluster them into significant locations using DBSCAN algorithm. For our case, DBSCAN is a better option than k-means and GMM [Reynolds, 2009], since numer of clusters are not defined beforehand and it can create clusters of arbitrary shape [Ester et al., 1996]. We use the words *GPS location* and *GPS point* interchangeably throughout the paper.

In the first step, all unique GPS locations L from visits are extracted and for each location $l \in L$ visit histories $\Psi^{(l)}$ are generated. Then, for each $l \in L$, the weight of l is defined as total residence seconds of visits performed to it (w_l) divided by total residence seconds of all visits (W), namely $\frac{w_l}{W}$, where:

$$w_l = \sum_{\forall v^{(l)} \in \Psi^{(l)}} v^{(l)}.d \quad \text{and} \quad W = \sum_{\forall k \in L} \sum_{\forall v^{(k)} \in \Psi^{(k)}} v^{(k)}.d$$

In the equations, $v^{(l)}$ represents a visit performed to location l and $v^{(k)}$ represents a visit performed to location k.

In the second step, weighted cumulative Gaussian distributions are calculated. The calculation includes a computation of mutual Gaussian values between chosen location $\mu \in L$ and each location $l \in L$, multiplication of corresponding value by a weight of the location l and summation of results. In this case, locations nearby to the given location μ contribute to the sum performing uniform smoothing. In other words, each nearby location contributes to a peak of the cumulative Gaussian distribution of location μ. According to practices for typical GPS accuracy[2], standard deviation σ of Gaussian distributions is selected to be 10 meters [Scellato et al., 2011]. Hence, $\forall \mu \in L$, weighted cumulative distribution function $\hat{F}_\mu$ is defined as follows:

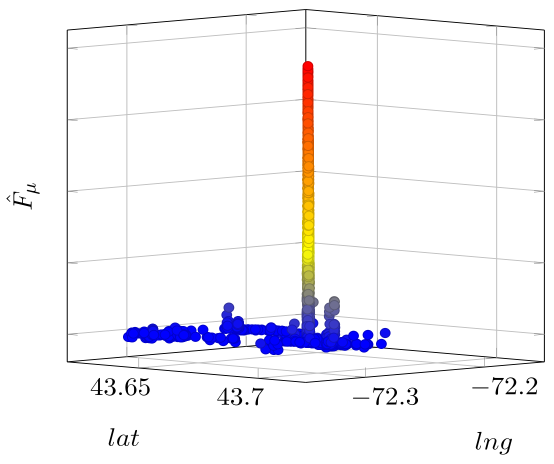
$$\hat{F}_\mu = \sum_{l \in L} F_l(\mu) \cdot \frac{w_l}{W}$$

where $F_l(\mu)$ is the value of the 2-D Gaussian function centered at a location l with respect to the location μ. Here, the value of $\hat{F}_\mu$ represents a peak of the cumulative Gaussian distribution of the location μ. Instead of sum of squared differences of respective latitudes and longitudes of l and μ, squared geographical (haversine) distance λ is calculated between them. So, $\forall l \in L$, the formula of $F_l(\mu)$:

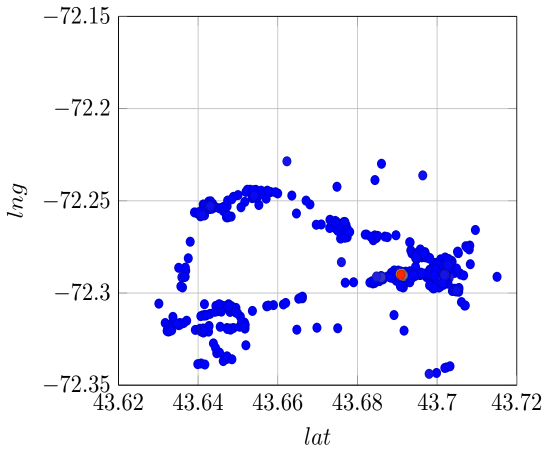
$$F_l(\mu) = \frac{1}{2\pi\sigma^2} \cdot e^{(-\frac{\lambda^2(l,\mu)}{2\sigma^2})}.$$

Consequently, $\forall \mu \in L$, calculation of $\hat{F}_\mu$ results in a *3D peak graph*. In this graph, x, y and z axes represent latitude, longitude and weighted cumulative Gaussian value $\hat{F}_\mu$ of a location μ respectively. Since nearby locations contribute to the peak of μ, this peak belongs to the specific area of those locations. Interestingly, peaks of nearby locations are also numerically close to one another. In this case, higher peaks reveal popular areas in which users spend most of their time. We characterize areas passing specific threshold T as popular ones and central GPS location of each area becomes a significant location. The threshold T can be selected as a proportion of the highest peak in the *3D peak graph*. We demonstrate the usage of this extraction method and a practice to choose threshold T for two GPS datasets (*Cabspotting* and *CenceMe GPS*) in Section 4.

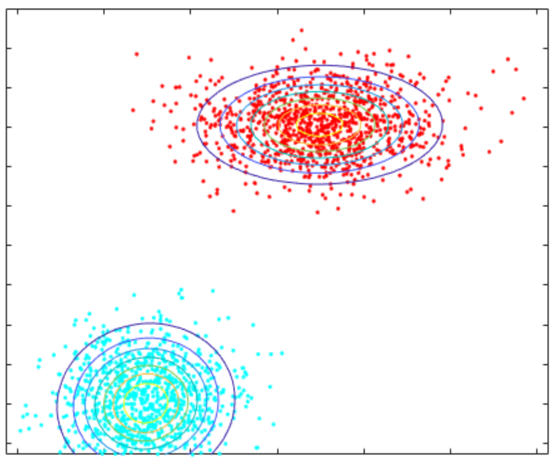
[2] https://www.gps.gov/systems/gps/performance/accuracy/



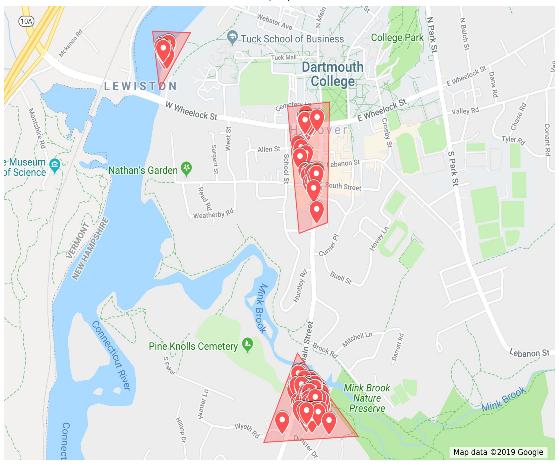
(a)



(b)



(c) [clu]



(d)

Figure 1: Example of a significant location extraction from GPS readings. Higher peaks in (a) determine areas of GPS points in which a user stayed for a long time. These areas can be specified by clustering GPS points in (b) which is a 2D view of a peak graph. Clustering results in contours of areas similarly defined in (c) and these areas in turn define significant locations. In (d), popular areas are shown, where average of all GPS locations within each area specifies significant locations (Note that, red areas are drawn by hand using maps tool).

The Fig. 1a shows the *3D peak graph*. When a threshold T is utilized, only peaks above threshold are chosen and areas specified by these peaks are used to generate significant locations. GPS locations within each specified area are averaged and obtained location is qualified as a significant one. All visits performed to GPS locations within each specified area are rerouted to the (averaged) significant location for that area. For instance, if the threshold T is defined as 20% of the maximum peak in the peak graph, we obtain popular areas as shown in Fig. 1d.

Significant location extraction for GPS datasets can be summarized in the following algorithm:

Algorithm 3 GPS Significant Location Processor

```
Input: Vector $\tilde{\varphi}$ of merged visits of a user $u$, threshold $T$, radius $\epsilon$, min points $m$
Output: Vector $\varphi'$ of visits to significant locations
1: obtain vector $L$ of unique locations from $\tilde{\varphi}$
2: for each $l \in L$ do
3:     create visit history $\Psi^{(l)}$ from $\tilde{\varphi}$
4: end for
5: initialize $W \leftarrow 0$
6: for each $l \in L$ do
7:     initialize $w_l \leftarrow 0$
8:     for each $v^{(l)} \in \Psi^{(l)}$ do
9:         update $W \leftarrow W + v^{(l)}.d$
10:        update $w_l \leftarrow w_l + v^{(l)}.d$
11:    end for
12: end for
13: for each $\mu \in L$ do
14:    initialize $\hat{F}_\mu \leftarrow 0$
15:    for each $l \in L$ do
16:        calculate $F_l(\mu)$ with respect to haversine distance $\lambda$ between $l$ and $\mu$
17:        update $\hat{F}_\mu \leftarrow \hat{F}_\mu + F_l(\mu) \cdot \frac{w_l}{W}$
18:    end for
19: end for
20: create empty vector $\hat{L}$ of above threshold locations
21: for each visit $v \in \tilde{\varphi}$ do
22:    obtain $\mu \leftarrow v.l$
23:    if $\hat{F}_\mu \geq T$ then
24:        add $\mu \rightarrow \hat{L}$
25:    end if
26: end for
27: obtain clusters $C$ with $DBSCAN$ from $\hat{L}$ using $\epsilon$ and $m$
28: for each cluster $c \in C$ do
29:    obtain cluster centroid $\mu_c$ by averaging locations in $c$
30: end for
31: create empty vector $\varphi'$ of significant visits
32: for each visit $v \in \tilde{\varphi}$ do
33:    if $v.l$ is contained in $\exists c \in C$ then
34:        create new visit $v' \leftarrow (v.u, \mu_c, v.t, v.d)$
35:        add $v' \rightarrow \varphi'$
36:    end if
37: end for
38: return $\varphi'$
```

As presented earlier, $\forall \mu \in L$, the algorithm above calculates weighted cumulative Gaussian values $\hat{F}_\mu$. Then, between lines 21-26, locations of visits whose weighted cumulative Gaussian value passes T are added to the vector $\hat{L}$. In this case, $\hat{L}$ can contain the same locations more than once, because more than one visit can be performed to a specific location. In other words, $\hat{L}$ is a multi-set or a bag.

Weighted cumulative Gaussian values of nearby GPS locations are close to one another. They can be used to define popular areas. Therefore, to specify such areas, the locations in the vector $\hat{L}$ are clustered with *DBSCAN* algorithm [Ester et al., 1996] using predefined radius ϵ and minimum points m. This algorithm is suitable for spatial databases. It clusters locations close in proximity, in the range of ϵ meters, and clustering is performed if there are least m locations close to each other. The choice of parameter m is crucial, it defines minimum number of locations that each cluster will have. We explain how we choose appropriate ϵ and m for GPS datasets in Section 4.

When clustering is completed, if $\forall v \in \tilde{\varphi}$, the location of visit v is contained in one cluster c of generated clusters, then v is rerouted to a new visit v' with the centroid location μ_c of cluster c. Centroid location of each cluster is characterized as a significant location. Rerouting visits to corresponding significant locations is a useful technique to generate visit histories (of the created significant locations) with sufficient length for prediction.

Note that, DBSCAN is only used to cluster the GPS points above the specific threshold T rather than raw location data. When raw location data without the time attribute is clustered, unimportant clusters of intermediate and transitory GPS points between popular areas can be created or transitory locations can be included in the existing clusters possibly having a negative effect on location predictions. Moreover, DBSCAN can also eliminate some GPS points which are above the threshold by marking them as outliers; either stays in these GPS points are accidental or number of nearby GPS points is very low which cannot be used for prediction. It is also possible to assign outlier GPS points to the nearest clusters. However, it should be checked how the location predictions are affected by this assignment.

When different threshold values are investigated (see Section 4), the weighted cumulative Gaussian distributions between lines 13-19 can be pre-calculated, since they require more computational effort. On the other hand, the time complexity of DBSCAN can be reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log(n))$ using spatial R-tree index. In any case, it is recommended to parallelize the execution of GPS significant location extraction to speed up performance.

Specifying popular areas is a challenging task. Different methods can be used to pick locations in specific areas of Gaussian peaks, but, we think DBSCAN is a pragmatic choice for this kind of tasks because of its simplicity.

3.2.2 Extracting Locations from WiFi Connectivity Data

As another option, we can generate significant locations of users from registration logs to WiFi access points. On one hand, the locations of WiFi access points are generally stationary. On the other hand, they are easily distinguishable from their universal MAC addresses. With this straightforward knowledge, one can extract visits of users to different significant locations. In this case, access points which are observed most often can be nominated to be significant locations. Thus, locations which are important for a user are the access points he is registered frequently by taking into account adequate number of recorded visits performed to these access points. More practically, an access point is characterized as a significant location, if a user has performed at least n visits to it.

Algorithm 4 WiFi Significant Location Processor

```
Input: Vector $\tilde{\varphi}$ of merged visits of a user $u$, visit frequency threshold $n$
Output: Vector $\varphi'$ of visits to significant locations
1: obtain vector $L$ of unique locations from $\tilde{\varphi}$
2: for each $l \in L$ do
3:     create visit history $\Psi^{(l)}$ from $\tilde{\varphi}$
4: end for
5: create empty vector $\varphi'$
6: for each $l \in L$ do
7:     if $|\Psi^{(l)}| \geq n$ then
8:         for each $v^{(l)} \in \Psi^{(l)}$ do
9:             add $v^{(l)} \rightarrow \varphi'$
10:        end for
11:    end if
12: end for
13: sort $\varphi'$ in chronological order
14: return $\varphi'$
```

We choose $n = 20$ as in the original NextPlace [Scellato et al., 2011], which allows us to get rid of visits performed to infrequent access points and have enough statistical observations for the analysis.

3.3 Prediction of User Movements

The idea behind the serial NextPlace and *Distributed NextPlace* is as follows: for forecasting future movements of a user, visit histories for each of his significant locations are utilized. For each significant location, algorithm predicts when the next visit will happen, and how long it will continue. Then, predictions generated for different locations are examined to obtain a single prediction of user location at a given time in the future. For the prediction of next visits and their durations, we use the technique proposed in Section 2.

3.3.1 Prediction of The Next Visits To A Given Location

For each user u, we retain chronologically ordered vector of visits to different (significant) locations. Then we generate visit history Ψ to each significant location which is also a chronological ordered vector. For location l, predictions are produced from its respective visit history $\Psi^{(l)} = (v_1^{(l)} = (u, l, t_1, d_1), v_2^{(l)} = (u, l, t_2, d_2) \dots, v_N^{(l)} = (u, l, t_N, d_N))$ or simply from its visit history's time series $(s_1 = (t_1, d_1), s_2 = (t_2, d_2), \dots, s_N = (t_N, d_N))$:

1. Time series $(\hat{s}_1 = (c_1, d_1), \hat{s}_2 = (c_2, d_2), \dots, \hat{s}_N = (c_N, d_N))$ is obtained from $(s_1 = (t_1, d_1), s_2 = (t_2, d_2), \dots, s_N = (t_N, d_N))$. Here, $\forall j \in \{1, \dots, N\}$, time series instant $\hat{s}_j$ is the version of s_j whose start time (arrival time) t_j is represented by day seconds c_j (in the interval of [0, 86400]).

2. With predefined embedding parameter m (delay parameter ν is always chosen as 1) and $\forall i \in \{m, m+1, \dots, N\}$ (where $m \ll N$ and note that indices of time series are not 0-based in this section), embedding vectors $\beta_i = (\hat{s}_{i-(m-1)}, \hat{s}_{i-(m-2)}, \dots, \hat{s}_i)$ are constructed. This creates an embedding space α as in algorithm Embedding Space Generator. Last embedding vector in α is $\beta_N = (\hat{s}_{N-(m-1)}, \hat{s}_{N-(m-2)}, \dots, \hat{s}_N)$.

3. According to a given distance metric, embedding vectors similar to β_N is searched in embedding space α. For that purpose, neighborhood $U_{.}(\beta_N)$ is created as presented earlier. All embedding vectors in the neighborhood are considered during the prediction process.

   In this step, the choice of distance metric is important. It defines the construction of the neighborhood of β_N. Distance metric between embedding vectors is based on distance between their respective time series instants. Distance between time series instants $s' = (t', d')$ and $s'' = (t'', d'')$ is the distance between their respective day seconds versions $\hat{s}' = (c', d')$ and $\hat{s}'' = (c'', d'')$, namely $d(s', s'') = d(\hat{s}', \hat{s}'') = |c' - c''|$. For the distance calculation, two embedding vectors must have the same number of components.

   Consider two embedding vectors $\beta_1 = (s_1, s_2, s_3)$ and $\beta_6 = (s_4, s_5, s_6)$, where distance between them is defined by the distances between their corresponding time series instants, i.e. $d(s_1, s_4)$, $d(s_2, s_5)$ and $d(s_3, s_6)$. This seems straightforward. Now, assume that, a user visits a location e.g. at 8:05am, 13:10pm and 18:15pm on one occasion. But, on another occasion, he visits the same location e.g. at 8:15am, 13:20pm and 18:30pm; in this case, distance between corresponding values makes sense. According to established human rhythms, the proposed routine might be different such that on the next occasion the user visits the same location at 10:10am, 15:05pm and 20:25pm; in this case, a comparison in terms of internal intervals makes sense. The given location that user visits can be his favorite gym. It is possible that, he visits the gym at certain time intervals, but on different times of day in different weekdays. So, it is interesting to explore how the intervals between internal components changes from one embedding vector to the other. For example, how $d(s_1, s_2)$ differs from $d(s_4, s_5)$ and $d(s_2, s_3)$ differs from $d(s_5, s_6)$. With this in mind, numerous distance metrics can be defined for embedding vectors.

   Given an embedding space α containing embedding vectors β_k and β_r with arbitrary indices k and r.

- Manhattan distance between β_k and β_r is the cumulative distance between their respective components:

$$d_{man}(\beta_k, \beta_r) = \sum_{j=1}^{m} d(\hat{s}_{k-(m-j)}, \hat{s}_{r-(m-j)}).$$

- Euclidean distance between them is the square root of cumulative squared distance:

$$d_{euc}(\beta_k, \beta_r) = \sqrt{\sum_{j=1}^{m} d^2(\hat{s}_{k-(m-j)}, \hat{s}_{r-(m-j)})}.$$

- Chebyshev distance between them is maximum distance between their corresponding components. Therefore, $\forall j \in \{1, \ldots, m\}$:

$$d_{che}(\beta_k, \beta_r) = \max_j \left[d(\hat{s}_{k-(m-j)}, \hat{s}_{r-(m-j)}) \right].$$

- Comparative Manhattan distance between β_k and β_r is calculated using internal component-wise distances. First, $\forall j \in \{1, \ldots, m-1\}$, component-wise distances between components i.e. $d_j^{(k)} = d(\hat{s}_{k-(m-j)}, \hat{s}_{k-(m-j)+1})$ and $d_j^{(r)} = d(\hat{s}_{r-(m-j)}, \hat{s}_{r-(m-j)+1})$ are computed. Then, $\forall j \in \{1, \ldots, m-1\}$, comparative distance $d_j^{(k,r)} = |d_j^{(k)} - d_j^{(r)}|$ between component-wise distances is obtained. Final distance is the summation of comparative distances:

$$d_{comman}(\beta_k, \beta_r) = \sum_{j=1}^{m-1} d_j^{(k,r)}.$$

- Comparative Euclidean distance is calculated using the same routine of comparative Manhattan distance. The only difference is that final distance is the square root of summation of squared comparative distances:

$$d_{comeuc}(\beta_k, \beta_r) = \sqrt{\sum_{j=1}^{m-1} \left(d_j^{(k,r)} \right)^2}.$$

- Comparative Chebyshev distance has the final distance as the maximum comparative distance, i.e. $\forall j \in \{1, \ldots, m-1\}$:

$$d_{comche}(\beta_k, \beta_r) = \max_j \left[d_j^{(k,r)} \right].$$

- Comparative looping distance $d_{comloop}$ uses the same heuristics of comparative Manhattan, Euclidean and Chebyshev distances, but distance computation is stopped when there is only one element left in the vector $d^{(k)}$ (or $d^{(r)}$). It can be summarized in the following algorithm:

Algorithm 5 Comparative Looping Distance

```
Input: Embedding vector $\beta_k$, embedding vector $\beta_r$
Output: Distance between $\beta_k$ and $\beta_r$
 1: initialize $m \leftarrow |\beta_k|$
 2: if m = 1 then return $d(\hat{s}_k, \hat{s}_r)$
 3: end if
 4: create empty vectors $d^{(k)}$ and $d^{(r)}$ of length $m-1$
 5: for $j = 1, \ldots, m-1$ do
 6:     update $d_j^{(k)} \leftarrow d(\hat{s}_{k-(m-j)}, \hat{s}_{k-(m-j)+1})$
 7:     update $d_j^{(r)} \leftarrow d(\hat{s}_{r-(m-j)}, \hat{s}_{r-(m-j)+1})$
 8: end for
 9: repeat
10:     create empty vectors $\lambda^{(k)}$ and $\lambda^{(r)}$ of length $|d^{(k)}|-1$
11:     for $j = 1, \ldots, |d^{(k)}|-1$ do
12:         update $\lambda_j^{(k)} \leftarrow |d_j^{(k)} - d_{j+1}^{(k)}|$
13:         update $\lambda_j^{(r)} \leftarrow |d_j^{(r)} - d_{j+1}^{(r)}|$
14:     end for
15:     update $d^{(k)} \leftarrow \lambda^{(k)}$
16:     update $d^{(r)} \leftarrow \lambda^{(r)}$
17: until $|d^{(k)}| = 1$
18: return $|d_1^{(k)} - d_1^{(r)}|$
```

Comparative distance functions are useful when $m \geq 2$. In case of $m = 1$, $d_{comman}(\beta_k, \beta_r) = d_{comeuc}(\beta_k, \beta_r) = d_{comche}(\beta_k, \beta_r) = d_{comloop}(\beta_k, \beta_r) = d(\hat{s}_k, \hat{s}_r)$. When $m = 2$, comparative looping distance is equivalent to comparative Manhattan, comparative Euclidean and comparative Chebyshev distances.

4. After neighborhood creation, algorithm predicts next values of the time series $(s_1 = (t_1, d_1), s_2 = (t_2, d_2), \ldots, s_N = (t_N, d_N))$. For that purpose, day seconds version of the time series, $(\hat{s}_1 = (c_1, d_1), \hat{s}_2 = (c_2, d_2), \ldots, \hat{s}_N = (c_N, d_N))$ is used. Next values are determined by the step count, which is defined by $\Delta l = 1, 2, 3, \ldots$. Firstly, $\forall \beta_n \in U_{\cdot}(\beta_N)$, $c_{N+\Delta l}$ is predicted by averaging values $c_{n+\Delta l}$, where c_n is obtained from $\hat{s}_n \in \beta_n$:

$$c_{N+\Delta l} = \frac{1}{|U_{\cdot}(\beta_N)|} \sum_{\substack{\hat{s}_n \in \beta_n \\ \forall \beta_n \in U_{\cdot}(\beta_N)}} c_{n+\Delta l}$$

Secondly, $d_{N+\Delta l}$ is obtained by the same way:

$$d_{N+\Delta l} = \frac{1}{|U_{\cdot}(\beta_N)|} \sum_{\substack{\hat{s}_n \in \beta_n \\ \forall \beta_n \in U_{\cdot}(\beta_N)}} d_{n+\Delta l}$$

Then, $\hat{p}_{N+\Delta l} = (c_{N+\Delta l}, d_{N+\Delta l})$ is created, which is the prediction for future time series value for $\hat{s}_{N+\Delta l}$. Obtaining prediction $p_{N+\Delta l} = (t_{N+\Delta l}, d_{N+\Delta l})$ for future date/time based time series value $s_{N+\Delta l}$ is simple. In this case, $t_{N+\Delta l}$ becomes a date/time which is $c_{N+\Delta l}$ day seconds after the Unix Epoch date (1st of January, 1970, 00:00, UTC) by default. Produced prediction $p_{N+\Delta l}$ is utilized for future visit $v_{N+\Delta l} = (u, l, t_{N+\Delta l}, d_{N+\Delta l})$ of user u to the location l. We consider day-specific predictions, therefore chronological date of the prediction is not really necessary.

Now we give an example for the prediction of the next visit (in turn, its start time and residence time) from the past history of visits. We call an m-dimensional embedding vector (or embedding vector with m components) as an m-sequence. Assume that embedding parameter m = 3 and the last three visits of a user (in this case, corresponding time series instants of these visits comprise β_N) to a given location l are performed at 3:15pm, 11:00pm on Wednesday and 07:30am on Thursday. In order to make a prediction, a visit history of the location l is searched for 3-sequences which are close to (3:15pm, 11:00pm, 07:30am). Now, assume that, when one of the traditional distance functions proposed before is used, search gives (3:30pm, 11:15pm, 07:45am) and (3:10pm, 11:25pm, 07:15am) and follow-up visits that come after these 3-sequences start at 12:50pm and 1:40pm and have a residence time of 20 minutes and 30 minutes respectively. Then, the next visit to a location l is predicted to start at 1:15pm and to last for 25 minutes by averaging start times and residence times of follow-up visits.

The key idea behind the proposed technique is based on the assumption of strong behavioral patterns in daily human activities. For that purpose, start times (arrival times) of visits is mapped to the interval of [0, 86400] by discarding other aspects such as a week, month or year during the prediction.

The embedding parameter m has a strong effect on the accuracy of predictions. Note that with large m, the search is performed in a smaller set of m-sequences (namely $N - (m-1) \cdot \nu$) by comparing the components of every m-sequence to the components of the last m-sequence. In this case, depending on a structure of the last m-sequence and number of available m-sequences, the search is more likely to result in an empty set of similar m-sequences. But, if sufficient number of visits are taken into account, larger value of m can reveal specific patterns of m-sequences that may be available in fixed date/time intervals, e.g., in particular days.

3.3.2 Serial NextPlace

The (serial) NextPlace algorithm generalizes the proposed technique from the previous section such that it generates a prediction not only for the next visit to the given location, but also for the consecutive future visits. In a straightforward manner, it can average next values that follow each m-sequence as well as the values which are more than 1 step ahead [Scellato et al., 2011]. Note that, the accuracy of predictions can be negatively affected when observations in a time series are utilized to calculate further future values. [Kantz and Schreiber, 2004].

With the provided generalization scheme, future visits with start times and residence times for all significant locations can be predicted. At an arbitrary time T, NextPlace can predict a location of a user u after ΔT seconds. It consists of the following steps:

(i) For each significant location of a user u, a visit at step count K (beginning with $K = 1$) is predicted and a chronological sequence of predicted visits $v_1 = (u, l_1, t_1, d_1), v_2 = (u, l_2, t_2, d_2), \ldots, v_n = (u, l_n, t_n, d_n)$ which satisfies $t_1 \leq t_2 \leq \cdots \leq t_n$ is created;

(ii) If for every $i \in \{1, \ldots, n\}$, there is a single visit $v_i = (u, l_i, t_i, d_i)$ that satisfies the predicate $t_i \leq T + \Delta T \leq t_i + d_i$, then v_i is chosen and l_i is returned as a predicted location (if there are multiple visits which satisfy the predicate, one of them is randomly chosen and its location is returned as a predicted location);

(iii) If none of the predicted visits satisfies the given predicate above, there are two possibilities:

- If the minimum start time t_1 of the predicted visits of the step count K is before $T + \Delta T$, i.e. $t_1 < T + \Delta T$, then prediction is expanded further by doubling the step count K. Hence, the algorithm is restarted by taking into account the predicted visits of the new step count K.
- Otherwise, expanded prediction yields visits which begin after $T + \Delta T$, i.e. $T + \Delta T < t_1$, then those visits cannot be employed for location prediction. Therefore, the algorithm ends by returning that a user u will not be in any significant location at time $T + \Delta T$.

For (i), the steps 1, 2, 3, 4 from the previous section are utilized. Note that in step (iii), for the minimum start time t_1, if $t_1 < T + \Delta T$ is satisfied, then $t_1 + d_1 < T + \Delta T$ is also satisfied because of a predicate defined in step (ii).

Serial NextPlace algorithm also makes it practical to predict the state of a user outside of his significant locations such as changeover from one significant location to the other. The general purpose of location prediction applications is to find in which significant location that a user will be at a future instant of time. Therefore, we do not concentrate much on the practicality of outside states.

3.3.2.1 Maximum Predictability Rate

Frequency or length of a visit history is characterized as number of visits it comprises. Maximum predictability rate is the maximum step count that a visit history can be predicted. Note that, for an embedding parameter m and delay parameter ν, the maximum number of embedding vectors which can be constructed from a time series $(s_1, s_2, \ldots, s_N)$ is $N - (m - 1) \cdot \nu$. This is also particularly true for an arbitrary visit history $\Psi_u^{(l)} = \{v_1^{(l)}, v_2^{(l)}, \dots, v_N^{(l)}\}$ of length N, since, underlying visits have time components. So, maximum step count is basically maximum index increment that the first embedding vector in the embedding space can achieve. Index of the first embedding vector of any embedding space (with 1-based indices) is always $(m-1)\cdot\nu+1$ (where the embedding space is portrayed as $(\beta_{(m-1)\cdot\nu+1}, \beta_{(m-1)\cdot\nu+2}, \dots, \beta_N)$. As a prediction limit, the last existing value of a time series which is indexed at N should be reached, so that index $(m-1)\cdot\nu+1$ can get a maximum increment of $(N-1)-(m-1)\cdot\nu$. Therefore, maximum predictability rate $maxK$ of any visit history $\Psi_u^{(l)}$ with frequency N is $(N-1)-(m-1)\cdot\nu$ (this can also be thought of as a maximum value of Δl in Section 3.3.1). In this case, $maxK'$ can be defined as the predictability rate of a visit history with maximum length among all visit histories. Interestingly, $(N-1)-(m-1)\cdot\nu$ is also a maximum number of neighbors that any last embedding vectror β_N can have.

3.3.3 Distributed NextPlace

Distributed NextPlace takes advantage of the generalization scheme of the serial NextPlace algorithm. In comparison, *Distributed NextPlace* predicts locations for all users in parallel by making few adjustments to the original algorithm. Consider an arbitrary time T; to predict locations of users after ΔT seconds, we propose the following algorithm:

```
Algorithm 6 Distributed NextPlace
Input: Dataset $D$ of preprocessed user files, $m$, $\nu$, distance metric $d$, $T$, $\Delta T$
Output: Vector Ł of predicted locations of users
1: obtain $\forall u \in D$ the vector of significant visits $\varphi'_u$ of a user $u$ in parallel
2: create empty vector Ł
3: add $\forall u \in D$ the result of NEXTPLACE($\varphi'_u$, $m$, $\nu$, $d$, $T$, $\Delta T$) $\rightarrow$ Ł in parallel
4: return Ł
```

The algorithm receives the following parameters: dataset D of preprocessed user files, embedding parameter or dimension m, delay parameter ν, distance metric d, time T and ΔT seconds. In our case, dataset D is one of four datasets (*Cabspotting*, *CenceMe GPS*, *Dartmouth WiFi*, *Ile Sans Fils*). Embedding parameter m can be 1, 2, 3 and so on. We choose $\nu = 1$ in all our experiments. Distance metric d can be one of the distance metrics provided in the previous sections. Time T is an arbitrary time in the interval of [0, 86400]. The structure of an arbitrary line in a user file is u, l, t, d which is parsed to create a corresponding visit. Lines are ordered chronologically with respect to the visits they represent.

Distributed NextPlace contains six functions: NEXTPLACE, PREDICT, PICKVISIT, EPSN, ISININTERVAL and ISEXTENSIONPOSSIBLE. It is important to define the pseudo-codes for these functions to bound the asymptotic runtime of the Distributed NextPlace algorithm.

The algorithm starts with the creation of the vectors φ'_u for each user u. Significant visits φ'_u are the visits which are performed by a user u to his significant locations. After creating vectors of visits, the algorithm calls NEXTPLACE to get future locations of all users in D in parallel. Note that, the NEXTPLACE function is conceptually equivalent to serial NextPlace algorithm with few differences which we mention in the later paragraphs.

```
function NEXTPLACE(Significant visits $\varphi'_u$, $m$, $\nu$, distance metric $d$, $T$, $\Delta T$)
    obtain vector of unique significant locations $L$ from $\varphi'_u$
    for each $l \in L$ do
        create visit history $\Psi_u^{(l)}$ from $\varphi'_u$
    end for
    for each $l \in L$ do
        obtain time series $(s_1, s_2, \ldots, s_N)$ of $\Psi_u^{(l)}$
        create embedding space $\alpha_u^{(l)}$ of $\Psi_u^{(l)}$ from $(s_1, s_2, \ldots, s_N)$ using $m$ and $\nu$
        calculate standard deviation $\sigma_u^{(l)}$ of $(s_1, s_2, \ldots, s_N)$
        obtain neighborhood $\eta_u^{(l)} \leftarrow$ EPSN($\alpha_u^{(l)}$, $\sigma_u^{(l)}$)
    end for
    create empty vector $P$ of chronologically ordered predicted visits
    for each $l \in L$ do
        obtain future visit $v^{(l)} \leftarrow$ PREDICT(1, $\Psi_u^{(l)}$, $\eta_u^{(l)}$)
        if $v^{(l)} \neq$ NULL then
            add $v^{(l)} \rightarrow P$
        end if
    end for
    if $P$ is still empty then
        return NULL
    else
        pick a visit $v \leftarrow$ PICKVISIT(1, $P$, $L$, $\Psi_u$, $\eta_u$, $T$, $\Delta T$)
        if $v =$ NULL then
            return NULL
        else
            return $v.l$
        end if
    end if
end function
```

NEXTPLACE acquires all different significant locations L that a user u visited. Then $\forall l \in L$, it creates visit history $\Psi_u^{(l)}$ of a user u to his significant location l. In line 8, embedding space vector $\alpha_u^{(l)}$ of visit history $\Psi_u^{(l)}$ is created. For that purpose, the algorithm Embedding Space Generator is utilized. Then, standard deviation $\sigma_u^{(l)}$ of the underlying time series of $\Psi_u^{(l)}$ is calculated. The vectors $\alpha_u^{(l)}$ and $\sigma_u^{(l)}$ are then used to obtain neighborhood vector $\eta_u^{(l)}$ from the EPSN function. It tries to find embedding vectors whose distance is smaller than ϵ which is 10% of the provided standard deviation [Kantz and Schreiber, 2004]. If no neighbors are found within ϵ radius, then EPSN returns empty neighborhood which in turn makes the PREDICT function return NULL as a predicted visit in line 14 of the NEXTPLACE function. In our implementation, we only used ϵ-neighborhood, because for k-nearest neighbors, a problem of defining an optimal number of neighbors for each visit history should be handled.

```
function EPSN(embedding space $\alpha$, standard deviation $\sigma$)
    initialize $\epsilon \leftarrow \sigma \cdot 0.01$
    create empty neighborhood $U$
    remove $\beta_N$ from $\alpha$
    for each $\beta_n \in \alpha$ do
        calculate distance $d(\beta_n, \beta_N)$
        if $d(\beta_n, \beta_N) \leq \epsilon$ then
            add $\beta_n \rightarrow U$
        end if
    end for
    add $\beta_N \rightarrow \alpha$
    return $U$
end function
```

When the neighborhood vector $\eta_u^{(l)}$ is available, the next visit for $\Psi_u^{(l)}$ can be predicted. To hold the next visits for all visit histories, vector P is created which will keep track of chronological ordering of predicted visits. Suppose that, for significant locations $l_1, l_2, \dots, l_{|L|}$, $|L|$ number of visits are predicted; then the predicted visits $v_1 = (u, l_1, t_1, d_1), v_2 = (u, l_2, t_2, d_2), \dots, v_{|L|} = (u, l_{|L|}, t_{|L|}, d_{|L|})$ will satisfy $t_1 \leq t_2 \leq \dots \leq t_{|L|}$. To predict the next visit, the PREDICT function is utilized.

```
function PREDICT(step $\Delta l$, $\Psi_u^{(l)}$, $\eta_u^{(l)}$)
    filter embedding vectors in $\eta_u^{(l)}$ whose indices cannot see $\Delta l$ steps ahead
    if $\eta_u^{(l)}$ becomes empty then
        return NULL
    end if
    initialize $c_{avg} \leftarrow 0$
    initialize $d_{avg} \leftarrow 0$
    obtain time series $(s_1, s_2, \dots, s_N)$ of $\Psi_u^{(l)}$
    for each $\beta_n \in \eta_u^{(l)}$ do
        obtain $\hat{s}_n$ of $\beta_n$
        obtain $\hat{s}_{n+\Delta l}$ from $(s_1, s_2, \dots, s_N)$
        obtain $c_{n+\Delta l}$ and $d_{n+\Delta l}$ of $\hat{s}_{n+\Delta l}$
        update $c_{avg} \leftarrow c_{avg} + c_{n+\Delta l}$
        update $d_{avg} \leftarrow d_{avg} + d_{n+\Delta l}$
    end for
    update $c_{avg} \leftarrow c_{avg} \;/\; |\eta_u^{(l)}|$
    update $d_{avg} \leftarrow d_{avg} \;/\; |\eta_u^{(l)}|$
    generate $\hat{p}_{N+\Delta l} \leftarrow (c_{avg}, d_{avg})$
    obtain $p_{N+\Delta l}$ from $\hat{p}_{N+\Delta l}$
    generate $v_{N+\Delta l} \leftarrow (u, l, p_{N+\Delta l})$
    return $v_{N+\Delta l}$
end function
```

The PREDICT function accepts the step count Δl and $\Psi_u^{(l)}$, $\eta_u^{(l)}$ of a significant location l as parameters. In line 2, it filters embedding vectors in $\eta_u^{(l)}$ whose indices cannot see Δl steps ahead. Index of the last embedding vector β_N in $\alpha_u^{(l)}$ of $\Psi_u^{(l)}$ is N. After filtering operation, embedding vectors $\beta_n \in \eta_u^{(l)}$ whose index n satisfies $n < N - (\Delta l - 1)$ will be used in a prediction process. If no embedding vectors remain in the neighborhood after filtering, the PREDICT returns NULL. Original $\eta_u^{(l)}$ in the NEXTPLACE function is kept untouched.

With neighborhood vector $\eta_u^{(l)}$, the PREDICT function performs a prediction using the same steps described in Section 3.3.1. Predicted visits by the PREDICT function are added to the vector P in line 16 of the NEXTPLACE function, which will be later used to pick a visit (and its location) at time $T + \Delta T$. Starting dates of visits in P are adjusted to Unix Epoch date, so that comparison between them are concentrated on arrival time in day seconds. If in case, P is still empty in line 19 of the NEXTPLACE function, then user's visit histories cannot see 1 step ahead. If visit histories cannot see 1 step ahead, they are also not able to see further steps ahead, therefore, NEXTPLACE returns NULL in line 20.

In line 22 of the NEXTPLACE function, the PICKVISIT function is called to pick a visit at time $T + \Delta T$ from the vector of predicted visits P. If PICKVISIT returns NULL, then the user u will not have a visit to any significant location (or he will not be in any significant location).

```
1:  function PICKVISIT($K$, $P$, $L$, $\Psi_u$, $\eta_u$, $T$, $\Delta T$)
2:      create empty vector $V$ of picked visits
3:      for each $v \in P$ do
4:          if ISINTERVAL($v$, $T$, $\Delta T$) then
5:              add $v \to V$
6:          end if
7:      end for
8:      if $|V| = 1$ then
9:          return $V_1$
10:     else if $|V| > 1$ then
11:         return a visit with the highest residence seconds from $V$
12:     else
13:         if ISEXTENSIONPOSSIBLE($P_1$, $T$, $\Delta T$) then
14:             initialize $\hat{K} \leftarrow 2 \cdot K$
15:             create empty vector $\hat{P}$ of chronologically ordered predicted visits
16:             for each $l \in L$ do
17:                 obtain maximum predictability rate $maxK$ of $\Psi_u^{(l)}$
18:                 if $\hat{K} \leq maxK$ then
19:                     obtain future visit $v^{(l)} \leftarrow$ PREDICT($\hat{K}$, $\Psi_u^{(l)}$, $\eta_u^{(l)}$)
20:                     if $v^{(l)} \neq$ NULL then
21:                         add $v^{(l)} \to \hat{P}$
22:                     end if
23:                 end if
```

```
24:             end for
25:             if $\hat{P}$ is still empty then
26:                 return NULL
27:             else
28:                 return pickVisit($\hat{K}$, $\hat{P}$, $L$, $\Psi_u$, $\eta_u$, $T$, $\Delta T$)
29:             end if
30:         else
31:             return NULL
32:         end if
33:     end if
34: end function
```

The pickVisit function receives future step count K to control the recursion, vector of predicted visits P of the step count K, vector of unique significant locations L, time T and ΔT seconds. Other important parameters are collective vectors; the vector of visit histories Ψ_u and the vector of neighborhoods η_u of a user u. For instance, the vector of visit histories is specified for $l_1, l_2, \ldots, l_{|L|}$ such that $\Psi_u = \{\Psi_u^{(l_1)}, \Psi_u^{(l_2)}, \ldots, \Psi_u^{(l_{|L|})}\}$. Other collective vectors are defined in a similar way.

3.3.3.1 More Robust Approach to Pick Visits At The Specific Time

We utilize two operators for the comparison in the time series data: $<$, $\leq$, $>$ and $\geq$ operators only perform a time comparison, $\prec$, $\preceq$, $\succ$, $\succeq$ operators perform chronological comparison such that they consider both date and time.

In line 4, pickVisit checks whether the visit v contains $T+\Delta T$, i.e. $T+\Delta T$ are within bounds of start time and end time of visit v. This is done implicitly by the isInInterval function. Note that, in the original NextPlace, a double inequality (which has two single inequalities) $v.t \leq T + \Delta T \leq v.t + v.d$ is utilized for this purpose [Scellato et al., 2011]. Since, $T + \Delta T$ does not have a date component, comparison through time can possible neglect some visits which contain $T + \Delta T$. This is especially true for visits which starts on one day and end on the other day. Assume that, $v.t$ is 11:45pm (Jan 1, 1970), $v.t + v.d$ is 00:30am (Jan 2, 1970) and $T + \Delta T$ is 00:10am. In this case, $v.t \leq T + \Delta T$ is not satisfied, however, $T + \Delta T \leq v.t + v.d$ is satisfied which in turn makes the double inequality be not satisfied. Now take $T + \Delta T$ as 11:55pm, which makes the former single inequality be satisfied, but now the latter single inequality is not satisfied. To handle this problem, for $i = \{1, 2, \ldots, |P|\}$ and a visit v_i, we provide a date component for $T + \Delta T$ with respect to t_i or $t_i + d_i$ which means that $T + \Delta T$ gets the same date of t_i (we choose Unix Epoch date: January 1, 1970 UTC explicitly for all predicted visits) or $t_i + d_i$. In this case, because of an additional date comparison, the algorithm is able to perform correct interval search. As a side note; for $i = \{1, 2, \ldots, |P|\}$, the date of t_i is always January 1, 1970 UTC, and the date of $t_i + d_i$ can be either January 1, 1970 UTC or January 2, 1970 UTC.

```
1: function isInInterval(Visit $v_i$, $T$, $\Delta T$)
2:     obtain date/time $t_i \leftarrow v_i.t$
```

```
3:      obtain residence seconds $d_i \leftarrow v_i.d$
4:      initialize $\hat{T} \leftarrow (T + \Delta T)$ mod 86400
5:      obtain $\hat{T}^{(t_i)}$ which is the date/time version of $\hat{T}$ with respect to $t_i$
6:      if $\hat{T}^{(t_i)} \prec t_i$ then
7:          if $t_i$ and $t_i + d_i$ are on the same day then
8:              return false                                        ▷ do not pick $v_i$
9:          else                          ▷ $t_i$ and $t_i + d_i$ are not on the same day
10:             obtain $\hat{T}^{(t_i+d_i)}$ which is the date/time version of $\hat{T}$ with respect to $t_i + d_i$
11:             if $\hat{T}^{(t_i+d_i)} \preceq t_i + d_i$ then
12:                 return true                                     ▷ pick $v_i$
13:             else             ▷ $t_i \prec$ 00:00 $\preceq t_i + d_i \prec \hat{T}^{(t_i+d_i)}$; do not pick $v_i$
14:                 return false
15:             end if
16:         end if
17:     else                                                        ▷ $t_i \preceq \hat{T}^{(t_i)}$
18:         if $t_i$ and $t_i + d_i$ are on the same day then
19:             if $\hat{T}^{(t_i)} \preceq t_i + d_i$ then
20:                 return true
21:             else             ▷ $t_i \prec t_i + d_i \prec \hat{T}^{(t_i)} \prec$ 00:00; do not pick $v_i$
22:                 return false
23:             end if
24:         else                          ▷ $t_i$ and $t_i + d_i$ are not on the same day
25:             return true
26:         end if
27:     end if
28: end function
```

3.3.3.2 Extending The Predictions

In line 5 of the pickVisit function, if isInInterval returns true for some visit $v \in P$, then visit v is "picked". If only one visit is picked then it is returned as it is and if more than one visit is picked a visit with highest residence seconds (as compared to a random visit of the original NextPlace) from a set of picked visits is returned. Our assumption is that a visit with the higher residence seconds may have more importance (we evaluate this assumption in our empirical study). What happens if none of the visits from P is picked? According to the NextPlace algorithm, if none of the visits from P contains $T + \Delta T$, then there are two cases; either the end time of the first visit $v_1 = (u, l_1, t_1, d_1)$ is before $T + \Delta T$, i.e. $t_1 + d_1 < T + \Delta T$ or start time t_1 of the first visit is after $T + \Delta T$, i.e. $T + \Delta T < t_1$ (since P is chronologically ordered, t_1 is always a date/time at Unix Epoch date with the minimum time). In the former case, the prediction is extended further by doubling step count K and repeating the process of picking visits. In the latter case, the algorithm returns that a user will not be in any significant location (or NULL). We think this approach is debatable. Provided time T and in turn, $T + \Delta T$ do not have a date component, because the algorithm predicts where the user will be during a day based on the interval of [0, 86400]. The time has the repeating nature and single inequalities for time comparison can become tricky.

(i) Assume that, t_1 is 10:30pm (Jan 1, 1970), $t_1 + d_1$ is 11:00pm (Jan 1, 1970) and $T + \Delta T$ is 1:00am, then the inequality $t_1 + d_1 < T + \Delta T$ is not satisfied, no further predictions occur and the algorithm returns NULL. However, time difference between $t_1 + d_1$ and $T + \Delta T$ is just 2 hours with respect to the midnight 00:00 and it is possible that further predicted visits after doubling the step count K can contain $T + \Delta T$.

(ii) Now assume that, t_1 is 00:10am (Jan 1, 1970), $t_1 + d_1$ is 00:30am (Jan 1, 1970), and $T + \Delta T$ is 11:50pm, then the inequality $t_1 + d_1 < T + \Delta T$ is satisfied and further predictions will be performed to close the time gap of almost 24 hours.

(iii) As a final example, assume that, t_1 is again 10:30pm (Jan 1, 1970), $t_1 + d_1$ is 11:00pm (Jan 1, 1970), and now $T + \Delta T$ is 10:00pm, then $t_1 + d_1 < T + \Delta T$ is not satisfied, but why there is no reason for extending the predictions? If algorithm can reach 11:50pm from 00:30am, then it can also reach 10:00pm from 11:00pm by extending the predictions further. This is particularly true if visits are performed to a given location in specific days throughout the month and/or year.

So, the time comparison is different according to the approach that one chooses; in one approach e.g. 03:00am can be smaller than 11:45pm, but in the other approach 11:45pm can be smaller than 03:00am. For the original NextPlace algorithm, single inequalities such as $t_1 + d_1 < T + \Delta T$ and $T + \Delta T < t_1$ can be affected by the repeating nature of the time and make the algorithm stop earlier by not considering further possibilities. These two cases presented by the original NextPlace paper [Scellato et al., 2011] are summarized in the isExtensionPossible function. Note that, NP is a shorthand for the word “NextPlace”.

```
1:  function isExtensionPossible(Visit $v_{min}$, $T$, $\Delta T$)
2:      obtain date/time $t_{min} \leftarrow v_{min}.t$
3:      obtain residence seconds $d_{min} \leftarrow v_{min}.d$
4:      initialize $\hat{T} \leftarrow (T + \Delta T)$ mod 86400
5:      obtain $\hat{T}^{(t_{min})}$ which is the date/time version of $\hat{T}$ with respect to $t_{min}$
6:      if $t_{min}$ and $t_{min} + d_{min}$ are on the same day then
7:          if $t_{min} + d_{min} \prec \hat{T}^{(t_{min})}$ then
8:              return true                              ▷ by the def. of NP
9:          else if $\hat{T}^{(t_{min})} \prec t_{min}$ then
10:             return false      ▷ by the def. of NP, but extension can still be possible
11:         else
12:             return false   ▷ $\hat{T}^{(t_{min})} = t_{min} + d_{min}$, already checked in isInInterval
13:         end if
14:     else                              ▷ $t_{min}$ and $t_{min} + d_{min}$ are not on the same day
15:         return false          ▷ by the def. of NP, but extension can still be possible
```

```
16:     end if
17: end function
```

Counter examples for the line 10 of IsExtensionPossible are **(i)** and **(iii)** from the previous paragraph. Line 12 represents the equality case which means no extension is required (since, that check is already performed by isInInterval, line 12 is never executed). A counter example for line 15 can be any visit v_{min} satisfying $t_{min} \prec 00:00 \preceq t_{min} + d_{min} \prec \hat{T}^{(t_{min}+d_{min})}$ where $\hat{T}^{(t_{min}+d_{min})}$ is the date/time version of $\hat{T}$ with respect to $t_{min} + d_{min}$ (i.e. $\hat{T}^{(t_{min}+d_{min})}$ gets the same date of $t_{min} + d_{min}$). For example, t_{min} can be 11:30pm (Jan 1, 1970), $t_{min} + d_{min}$ can be 00:30am (Jan 2, 1970) and $\hat{T}$ can be 01:00 am. Only 30 minutes of the time gap separates $\hat{T}$ and $t_{min} + d_{min}$ which can be fixed by an extension. The predicates $t_{min} \prec 00:00 \preceq \hat{T}^{(t_{min}+d_{min})} \preceq t_{min} + d_{min}$ and $t_{min} \preceq \hat{T}^{(t_{min})} \prec 00:00 \preceq t_{min} + d_{min}$ are already checked by isInInterval and there is no need to recheck them. Note that, the implicit double inequality used by the isInInterval function to check whether a predicted visit contains $T + \Delta T$ is not affected by the repeating nature of the time, since it is a time interval search with the start and end, and $T + \Delta T$ is given a date component in appropriate places.

If we consider the extension for line 10 and 15 of isExtensionPossible, then it is possible that the recursion of pickVisit may continue for a long time, i.e. until the chronological sequence of predicted visits $\hat{P}$ gets empty in line 25 of the pickVisit function. In the worst case, this can happen when all visit histories are predicted up to their predictability limit, i.e. when the predictability limit $maxK'$ of the max-length visit history is reached. There are two main disadvantages of predicting every visit history up to its limit; *(i)* prediction further values in the future can become inaccurate which can negatively affect the location prediction accuracy, *(ii)* users can have very long visit histories such that predicting them up to $maxK$ can degrade the runtime performance, because for each new step count, a prediction needs to be generated. In this case, stopping the execution earlier by a trick of the isExtensionPossible function makes sense (i.e. we do not extend in line 10 and 15). Note that, in the worst case, it can still make the recursion continue until $maxK'$, but in practice, it is much faster. Our findings also indicate that using isExtensionPossible over $maxK'$ yields better predictive performance.

When the prediction is extended, K is doubled in line 14 of the pickVisit function. In this case, $\hat{K}$ gets $2, 4, 8, 16, \ldots$ and so on. It is also possible to increment K instead of doubling it. When K is doubled, the values $3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, \ldots$ of $\hat{K}$ are missed when compared to the scenario where the K is incremented. This means that visits which can possible contain $T + \Delta T$ are not checked for the missed values of $\hat{K}$. We learned from our empirical study that the incrementation of K provides slightly better predictive performance in some cases, but it is not much higher than the scenario where the K is doubled. Moreover, the runtime performance worsens, since the K can be incremented $maxK'$ times in the worst case. However, in the worst case, doubling K can be executed $\log(maxK')$ times. In other words, it provides an advantage of $\log(maxK')$ recursive steps in the worst case. Therefore, we utilize the doubling option.

3.3.3.3 Complexity

The time complexity of *Distributed NextPlace* algorithm is dominated by the runtime of the NextPlace function, since this function is called for all users in a dataset D in parallel. We consider the worst-case scenario in our analysis. In this case, the parallel execution runtime of the NextPlace function will be dominated by a user whose product of the number of significant visits by the length of max-length visit history is the highest among all users. For this reason, we provide the following assumption: consider $\forall i \in \{1, \ldots, |D|\}$ and for the vector of *unique* significant locations L_{u_i} of a user u_i, there is $\exists \hat{l}^{(u_i)} \in L_{u_i}$ such that

$$|\Psi_{u_i}^{(\hat{l}^{(u_i)})}| = \max_j [|\Psi_{u_i}^{(l_j^{(u_i)})}|; \forall j \in \{1, \ldots, |L_{u_i}|\}]$$

then, we assume that there is a user $u \in D$ whose

$$|\varphi'_u| \cdot |\Psi_u^{(\hat{l}^{(u)})}| \geq |\varphi'_{u_i}| \cdot |\Psi_{u_i}^{(\hat{l}^{(u_i)})}|.$$

We will bound the worst-case runtime with $|\hat{\varphi}| = |\varphi'_u|$, and $\hat{N} = |\Psi_u^{(\hat{l}^{(u)})}|$.

Line by line analysis of the NextPlace function reveals the overall worst-case running time of the algorithm. In line 2, *unique* significant locations are obtained in $\mathcal{O}(|\hat{\varphi}|)$. Between lines 3 and 5, when appropriate data structures are used, visit histories can be created in $\mathcal{O}(|\hat{\varphi}|)$ time. Obtaining time series in line 7 takes $\mathcal{O}(1)$ time. In line 8, embedding spaces are created with Embedding Space Generator algorithm; it is asymptotically $\mathcal{O}(\sum_{l \in L_u} m \cdot |\Psi_u^{(l)}|) \sim \mathcal{O}(\sum_{l \in L_u} |\Psi_u^{(l)}|) = \mathcal{O}(|\hat{\varphi}|)$, since, $\forall l \in L_u$, $m \ll |\Psi_u^{(l)}|$. In line 9, the time complexity of calculating standard deviations for every location in L_u is $\mathcal{O}(|\hat{\varphi}|)$. Line 10 calls the epsN function. Inside this function, $\forall l \in L_u$, $|\alpha_u^{(l)}| = |\Psi_u^{(l)}| - (m-1) \cdot \nu - 1$ because of a removal of the last embedding vector and the distance between two embedding vectors are calculated in m steps. So, epsN takes $\mathcal{O}(\sum_{l \in L_u} m \cdot |\alpha_u^{(l)}|) \subseteq \mathcal{O}(\sum_{l \in L_u} m \cdot |\Psi_u^{(l)}|) \sim \mathcal{O}(\sum_{l \in L_u} |\Psi_u^{(l)}|) = \mathcal{O}(|\hat{\varphi}|)$. Overall, lines 6-11 of nextPlace take $\mathcal{O}(3 \cdot |\hat{\varphi}|)$ which is $\mathcal{O}(|\hat{\varphi}|)$.

Line 14 of the nextPlace function calls the predict function. Running time of the predict function is dominated by the runtime of filtering indices and the next visit prediction. Line 2 of this function takes

$$\mathcal{O}\left(\sum_{l \in L_u} |\eta_u^{(l)}|\right) \subseteq \mathcal{O}\left(\sum_{l \in L_u} |\alpha_u^{(l)}|\right) \subseteq \mathcal{O}\left(\sum_{l \in L_u} |\Psi_u^{(l)}|\right) = \mathcal{O}(|\hat{\varphi}|)$$

time, because $\forall l \in L_u$, $|\eta_u^{(l)}| \leq |\alpha_u^{(l)}| \leq |\Psi_u^{(l)}|$. Lines 9-15 of the predict function are still $\mathcal{O}(\sum_{l \in L_u} |\eta_u^{(l)}|) \subseteq \mathcal{O}(\sum_{l \in L_u} |\alpha_u^{(l)}|) \subseteq \mathcal{O}(|\hat{\varphi}|)$. Overall, running time of the predict function is $\mathcal{O}(|\hat{\varphi}| + |\hat{\varphi}|) \sim \mathcal{O}(|\hat{\varphi}|)$. In this case, line 14 of nextPlace is also $\mathcal{O}(|\hat{\varphi}|)$. The vector P is chronologically ordered, therefore in Line 16 of nextPlace, the additional *log* factor is applied during the insertion. Maximum $|L_u|$ visits can be added to P, therefore, the complexity of line 16 becomes $\mathcal{O}(\sum_{i=\{1,\ldots,|L_u|-1\}} \log i) \subseteq$

$\mathcal{O}(\log(|L_u|!)) \subseteq \mathcal{O}(|L_u| \cdot \log(|L_u|))$. In total, the complexity of lines 12-18 of NEXTPLACE is $\mathcal{O}(|\hat{\varphi}| + |L_u| \cdot \log(|L_u|))$. Will $\mathcal{O}(|\hat{\varphi}|)$ dominate $\mathcal{O}(|L_u| \cdot \log(|L_u|))$? To answer this question, we provide the analysis over the cardinality of visit histories, the number of significant visits and the number of significant locations of the user u. To have enough statistical observations, we defined a location as a significant one with at least 20 visits for WiFi datasets and 19 visits for GPS datasets (more information on these numbers is given in the next section). This means that, $\forall l \in L_u$, $|\Psi_u^{(l)}| \geq 19$ and in turn, $|\hat{\varphi}| \geq 19$. It is also clear that the number of significant visits $|\hat{\varphi}| = \sum_{l \in L_u} |\Psi_u^{(l)}|$. Then, $|L_u|$ is maximized, when $\forall l \in L_u$, $|\Psi_u^{(l)}| = 19$. So, at maximum $|L_u| = \frac{|\hat{\varphi}|}{19}$. The domination of $\mathcal{O}(|\hat{\varphi}|)$ over $\mathcal{O}(|L_u| \cdot \log(|L_u|))$ occurs when

$$|\hat{\varphi}| \geq |L_u| \cdot \log(|L_u|) \Rightarrow |\hat{\varphi}| \geq \frac{|\hat{\varphi}|}{19} \cdot \log(\frac{|\hat{\varphi}|}{19}) \Rightarrow 19 \geq \log(\frac{|\hat{\varphi}|}{19})$$
$$\Rightarrow 19 \geq \log(|\hat{\varphi}|) - \log 19 \Rightarrow 19 + \log 19 \geq \log(|\hat{\varphi}|) \Rightarrow \log(|\hat{\varphi}|) \leq 2^{23.247927513443585493}$$
$$\Rightarrow |\hat{\varphi}| \leq 9,961,472.$$

The number 9,961,472 can be reached when a user u performs a visit to some significant location every 1 minute during 28.5 years, by taking into account 8 hours of sleep time for every day. This is very strong assumption; because this kind of routine is not generally present in human societies and is out of the limit of physical capabilities of human beings. In general, every user has a few significant locations that he/she visits most and for such users, WiFi and/or GPS datasets are constructed from spatio-temporal data which spans maximum for a few years. Moreover, considering a cardinality of every visit history to be equal 19 is also a strong assumption, generally visit histories tend to have different cardinalities. Therefore, in practice $|L_u| \cdot \log(|L_u|)$ is negligible and the lines 12-18 of the NEXTPLACE function takes $\mathcal{O}(|\hat{\varphi}|)$ time.

In line 22, the PICKVISIT function is called. When the recursive nature of PICKVISIT is analyzed, it is clear that lines from 3 to 7 take $\mathcal{O}(|P|)$ and in the worst-case it is $\mathcal{O}(|L_u|)$, if P contains $|L_u|$ visits. The critical part is between lines 12 and 33, where the $\log(maxK')$ repetitions are performed in the worst case. Here, the behavior between the lines 15 and 24 is the same as the corresponding section in the NEXTPLACE function which takes $\mathcal{O}(|\hat{\varphi}|)$. To deduce the total runtime of the PICKVISIT function, we express the predictability rate $maxK'$ by $\hat{N}$ (since, $\hat{N}$ is the length of the max-length visit history), i.e. $maxK' = \hat{N} - 1 - (m-1) \cdot \nu$. In this case, with $\log(maxK')$ repetitions, the total runtime of PICKVISIT is $\mathcal{O}(\log(maxK') \cdot |L_u| + \log(maxK') \cdot |\hat{\varphi}|)$ which is roughly $\mathcal{O}(\log(\hat{N}) \cdot |\hat{\varphi}|)$. So, the runtime of line 22 of the NEXTPLACE function is $\mathcal{O}(\log(\hat{N}) \cdot |\hat{\varphi}|)$.

The summation of complexities of different lines in the NEXTPLACE function provides the total runtime of *Distributed NextPlace* algorithm. It is $\mathcal{O}(4 \cdot |\hat{\varphi}| + \log(\hat{N}) \cdot |\hat{\varphi}|)$ which is approximately $\mathcal{O}(\log(\hat{N}) \cdot |\hat{\varphi}|)$. Note that, the time complexity can become $\mathcal{O}(|\hat{\varphi}| \cdot \log(|\hat{\varphi}|))$, when $|L_u| \to 1 \Rightarrow \hat{N} \to |\hat{\varphi}|$.

We considered the worst case asymptotic runtime of the algorithm. However, on average it is $\Theta(|\hat{\varphi}|)$: in general, for every $l \in L_u$, the neighborhood size $|\eta_u^{(l)}|$ is much smaller than $|\alpha_u^{(l)}|$ and $\log(maxK')$ recursive steps are not performed.

The main purpose of the parallelization of algorithms is to reduce the runtime. For parallel algorithms, the space complexity is not at the top consideration. Since advanced memory equipment is present at affordable costs, storage is no longer a problem for such algorithms.

4 Empirical Evaluation

This section starts with presenting the datasets used in the evaluation. To transform the datasets into a structured form, data preprocessing is performed. After demonstrating crucial preprocessing steps, we show how the structured data is processed to obtain significant locations. Furthermore, a predictability measure for the time series obtained from visits of every user to his significant locations is examined. Finally, the predictive and runtime performance of *Distributed NextPlace* is compared against the serial NextPlace algorithm using appropriate measures.

4.1 Datasets

For the empirical evaluation of our prediction method, we make use of four datasets of user movement traces:

(i) The **Cabspotting** dataset consists of movement traces of taxi cabs (users) in San Francisco, USA. Movement traces contain timestamped GPS coordinates of more than 500 taxis collected over 30 days in the San Francisco Bay Area. Every cab vehicle is provided with a GPS tracker that is used by dispatchers to adequately reach clients [Piorkowski et al., 2009]. The sampling interval for two successive GPS readings is smaller than 60 seconds.

(ii) The **CenceMe GPS** dataset was created in the course of the deployment of CenceMe application [Miluzzo et al., 2008] - a personal sensing system relying on sensor data obtained using mobile devices at Dartmouth College. The timestamped GPS coordinates were collected through 20 Nokia N95 smartphones of graduate students and staff members.

(iii) The **Dartmouth WiFi** dataset was collected from association logs to the WiFi network of the Dartmouth College campus. Close range of the campus buildings enables the signals of interior WiFi access points to even reach outdoor areas in the campus [Scellato et al., 2011]. Timestampted WiFi association data was collected between 2001 and 2004 from over 450 access points and several thousands of users using syslog recording, SNMP polling and network sniffing [Kotz et al., 2009, Henderson et al., 2004].

(iv) The **Ile Sans Fils** dataset contains session traces of several thousands of users which were collected from over 200 free WiFi hotspots in Montreal, Quebec, Canada for three years [Lenczner et al., 2007]. Ile Sans Fils a non-profit community wireless network that supplies free public wireless hotspots to mobile users in public places. Deployment of these hotspots ranges from cafes, restaurants and bars to libraries and outdoor areas such as parks and commercial streets [Scellato et al., 2011].

4.2 Data Preprocessing

Before extracting significant locations and performing predictions, data preprocessing is performed for the datasets. We implemented a different preprocessing technique for each original dataset because they differ in terms of the structure.

- For Dartmouth WiFi dataset, we handled possible "holes" (missing values) in association logs where the period between two consecutive measurements is larger (which can be several days). If the "hole" is discovered between the timestamp of the current measurement and the timestamp of the next measurement, the duration of a visit represented by the current measurement is set to the number of seconds of the adopted sampling interval (more in the next paragraph).
- For Ile Sans Fils dataset, timestamps for both login and logout times to WiFi access points are available. In this case, the duration of stay (the residence time of a visit) is extracted as a difference between the corresponding login and logout timestamps. For some measurements, the logout timestamps are missing. For that case, if the difference between the login timestamp of the current measurement (whose logout timestamp is missing) and login timestamp of the next measurement is higher than 8 hours, the duration of stay is set to the adopted WiFi sampling interval for a visit which is constructed from the current measurement. We think the difference of 8 hours (which represents a typical number of working hours) is a reasonable choice for such a case.
- Original GPS datasets are structure-wise much simpler than WiFi datasets. They contain a chronological sequence of timestamped GPS samples such that each sample represents one visit. In Cabspotting dataset, each timestamp corresponds to one GPS sample so that it is easier to extract visits. However, in CenceMeGPS dataset one timestamp can have multiple GPS samples depending on the movement of a user. In this case, we take the first recorded GPS sample of each timestamp to create a suitable visit, since recorded GPS locations of the same timestamp are close to one another and intuitively user could only exist in one location at a specific date/time.

For all datasets, the duration of the last visit corresponding to the last timestamp is set to the respective GPS or WiFi sampling interval, because there is no next timestamp to set a timestamp difference as a duration.

4.2.1 Adopted Sampling Intervals

After performing adjustments from the previous section, original visits with respect to timestamps of GPS readings or WiFi logs are obtained. Then, those visits are merged with respect to the adopted GPS and WiFi sampling intervals using a method described in Section 3.1. We make use of the same sampling intervals presented in [Scellato et al., 2011]. The sampling interval δ for Cabspotting and CenceMe GPS datasets is taken as 60 and 180 seconds respectively which are the typical values of the look-over time for the GPS data collection. On the other hand, the sampling interval δ for Dartmouth WiFi and Ile Sans Fils datasets is chosen as 300 seconds which is a pragmatic choice to filter incidental connection interruptions to the WiFi access points present for a small period.

Dataset	U	V	L	V_s	L_s	l_s	v_s	D_s	Trace length	Significant time
Cabspotting	536	10909769	6045606	9258288	10312	19.24	17272	84979	23 days	87.46%
CenceMe GPS	19	19959	19959	10699	52	2.74	563	191076	20 days	61%
Dartmouth WiFi	13888	30826020	602	29344655	583	5.72	2112	695674	1176 days	54.69%
Ile Sans Fils	69689	525054	206	316475	177	0.06	4	454167	1095 days	4.52%

Table 1: Features of used datasets: total number of users U, total number of visits V, total number of locations L, total number of significant visits V_s, total number of significant locations L_s, average number of significant locations per user l_s, average number of significant visits per user v_s, average residence seconds in significant locations D_s, trace length of the original dataset, average percentage of time spent by each user in significant locations.

4.2.2 Obtaining Significant Locations (WiFi)

Locations on Dartmouth WiFi and Ile Sans Fils datasets are anonymized stationary WiFi access point locations. The total number of locations in DartmouthWiFi and Ile Sans Fils datasets is 602 and 206 respectively. To extract significant locations from WiFi access points we adopted visit frequency threshold $n = 20$. This means that locations which have at least $n = 20$ visits performed to them are defined as significant locations (see Section 3.2.2). Extraction of significant locations results in 583 significant locations for Dartmouth WiFi dataset and 206 significant locations for Ile Sans Fils dataset as reported in Table 1.

4.2.3 Obtaining Significant Locations (GPS)

For the extraction of significant locations of each user in Cabspotting and CenceMe GPS datasets, an appropriate threshold T should be chosen. It is typically a proportion of the highest peak in the *3D peak graph* as depicted in Fig. 1a. More formally, the values of T are ranging between the value of the lowest peak and the value of the highest peak in the peak graph. The peaks are represented by the weighted cumulative Gaussian distribution of each GPS point that a user visited (see Section 3.2.1). Each peak is then compared to the threshold T and if the threshold is passed, then the corresponding GPS point is considered in the significant location generation. Above threshold GPS points are clustered using DBSCAN algorithm with the proper radius ϵ and min (minimum) points m and a centroid of each cluster is characterized as a significant location (visits performed to each GPS point in each cluster are rerouted to the corresponding centroid location).

4.2.3.1 Choosing Min Points m

The radius ϵ defines how close two GPS points should be and min points m specifies how many points should be in close proximity to form clusters. The min points m is taken as 18 for both GPS datasets. The value 18 of m is found with respect to the analysis of the predictability rate. This value also means that each cluster will have at least 19 GPS points because clustering starts if there are at least 18 points in proximity of some GPS point by the definition of the DBSCAN algorithm [Ester et al., 1996]. In this case, rerouted visits to the respective centroid will make the corresponding visit history (of that centroid) to have the length of at least 19. Since, the embedding parameter or dimension is 3 as its maximum value, then the predictability rate is $19 - 1 - (3 - 1) \cdot 1 = 16$. This predictability rate is useful such that visit histories can be predicted up to the step count $\mathrm{K} = 16$ and enough observations will be present in the visit histories for the neighborhood search. Note that, for WiFi datasets we specify the locations as a significant one if there are at least $n = 20$ visits performed to them which means that visit histories can have the length of at least 20 with the predictability rate of at least $20 - 1 - (3 - 1) \cdot 1 = 17 \geq 16$. So, both WiFi and GPS datasets have a correspondence such that they carry common characteristics in terms of statistically sufficient visit histories and predictability rate.

4.2.3.2 Choosing Radius ϵ

To find the proper radius ϵ, we test the heuristic called *sorted k-distance plot* which is proposed by the original DBSCAN paper [Ester et al., 1996]. It is a simple but effective way to obtain some hints about the density distribution of points in the given dataset.

For each user in a GPS dataset, we draw *sorted k-distance plot*. In this plot, the function *k-dist* maps each GPS point to the distance from its k-th nearest neighbor. All the points are sorted in a descending order of k-distances and plotted. According to the considered heuristic, the "knee" point in this plot describes the proper radius ϵ for the DBSCAN algorithm. To have an idea of radius ϵ, we investigate 18-dist plot for some portion of users, since minimum points $m = 18$. Our investigation reveals that the value of a knee point is changing between 500 - 1500 meters for different users. For example, Fig. 2 shows such a plot (with the dashed line marking the value of the knee point) for a sample user in Cabspotting dataset.

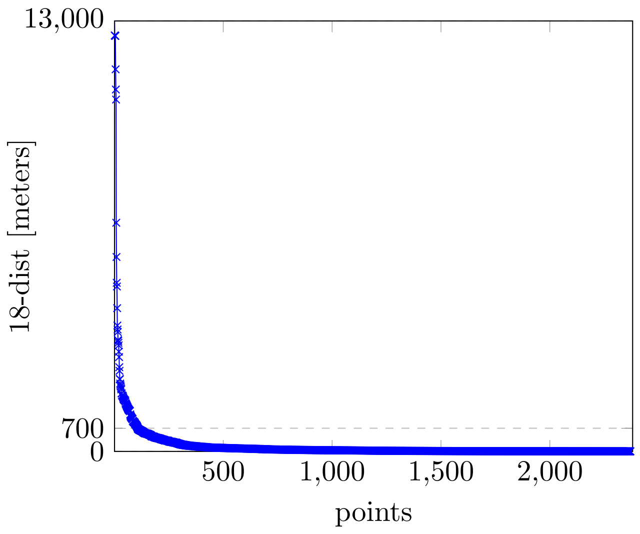


Figure 2: k-distance plot heuristic to find suitable radius ϵ for DBSCAN Algorithm. This plot is drawn for a sample user in Cabspotting dataset. For our case, $k = 18$. “Knee” point is at 700 meters.

The given heuristic provides k-distance plots which have numerically close knee points for all users in Cabspotting and CenceMe GPS datasets. For location predictions tasks, such a large radius generated by the given heuristic is quite inefficient. To enable our location predictions as practical and specific as possible, we want to have GPS points clustered with a small radius ϵ which would distinguish as many distinct clusters (groups of nearby GPS points) as possible. But, if the radius is taken too small, then a number of clusters would be small and most of the GPS points would be marked as outliers by the DBSCAN algorithm. Otherwise, if the radius is taken too large, this would make unrelated clusters (significant locations or areas) be joined; e.g. the cluster representing home can be merged with the cluster representing the university building. Both cases would make the prediction impractical. Therefore, we intuitively analyze the datasets for an appropriate ϵ.

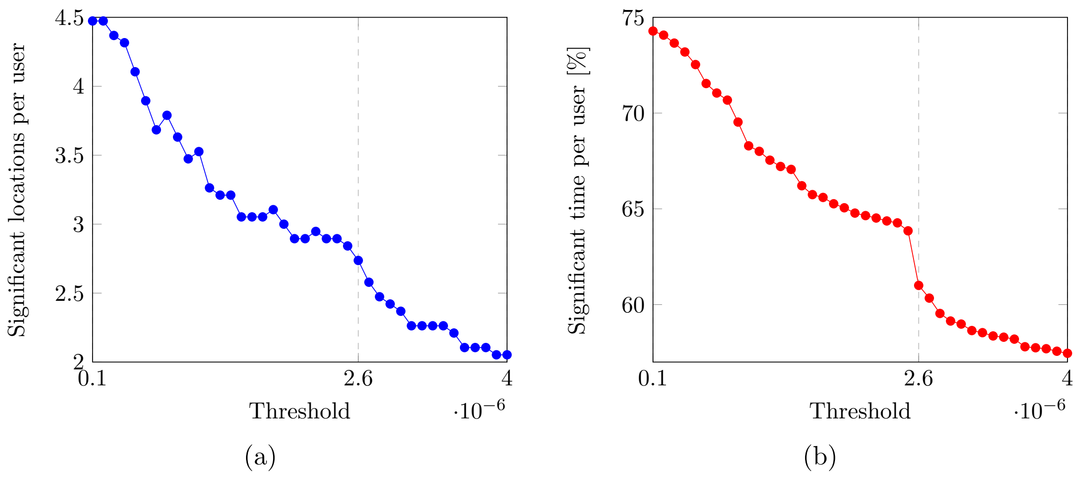
During preprocessing, we make all GPS points to be accurate as small as 5 digits after the decimal point for latitudes and longitudes. This guarantees that the comparison of two arbitrary GPS points would be accurate for 1 meter and less. CenceMe GPS dataset contains the GPS points recorded by students and staff members at Dartmouth College Campus. Since the recordings are made by (walking or sitting) persons inside and around the campus, then 100 meters (which guarantees accuracy for about 0.001 degrees) becomes a natural choice for ϵ. On the other hand, Cabspotting dataset consists of GPS points recorded by GPS trackers installed in taxi cabs moving in the San Francisco Bay Area. These GPS points belong to the areas where taxis visit and stay to get clients, such as movie theaters, touristic spots, shopping centers [Scellato et al., 2011]. We adopted $\epsilon = 200$ meters for Cabspotting dataset. When clustering is performed, a negligible number of GPS points becomes outlier (see Fig. 3c and Fig. 4c). The adopted radius ϵ

and min points m are then used to obtain the "significance" threshold for GPS datasets.

4.2.3.3 Choosing Threshold T

In order to find the appropriate threshold for both GPS datasets, the functions of "average significant locations per user", "average significant time (as a %) per user" and "total outlier GPS points with respect to different thresholds" are analyzed. The Fig. 3 and Fig. 4 depict those functions for CenceMe GPS and Cabspotting dataset, respectively. In Fig. 3a and Fig. 4a, average number of significant locations per user as function of T is demonstrated. In Fig. 3b and Fig. 4b, average significant time per user as function of T is shown where the significant time decreases as a threshold increases. Finally, Fig. 3c and Fig. 4c depicts the total number of discarded above threshold GPS points for each threshold.

Choosing threshold for CenceMe GPS dataset As shown in Fig. 3a, it is not possible to distinguish the knee point or the position with the biggest slope for CenceMe GPS dataset which can describe the state of a transition from not so significant locations to presumably more significant locations. As compared to Fig. 3a, Fig. 3b shows steep transition at $2.6 \cdot 10^{-6}$. Therefore, the value of T is $2.6 \cdot 10^{-6}$ for the CenceMe GPS dataset. All GPS points whose weighted cumulative Gaussian distribution is above this threshold are clustered using DBSCAN with adopted radius ϵ and min points m to obtain significant locations. According to Table 1, there are 19959 (unique) GPS points in CenceMe GPS dataset which is the same as the number of visits. This means that every GPS point receives just one visit making the creation of the sufficient-length visit histories impractical. After finding above threshold GPS points and performing clustering for them, 52 significant locations with 10699 visits are obtained. Note that, the DBSCAN algorithm is able to generate outliers. In Fig. 3c, for the threshold $T = 2.6 \cdot 10^{-6}$, the total outlier above threshold GPS points is 657. So, the clustering costs around 35 GPS points per user to be lost. This is very small as compared to the average number of significant visits per user $v_s = 563$ and we discard them since they do not have much statistical importance.



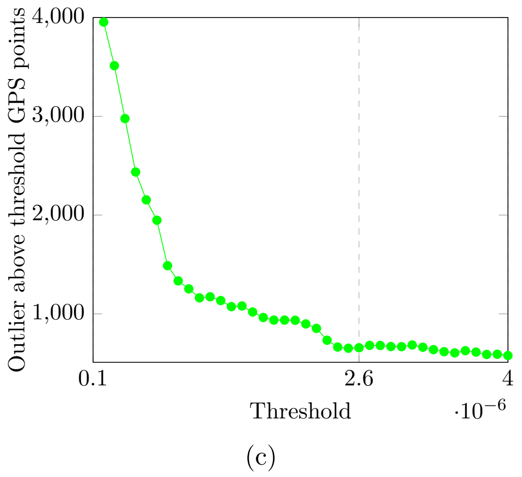


Figure 3: Threshold-based significant location extraction from CenceMe GPS dataset. The functions of average number of significant locations per user (a), average proportion of time spent in significant locations by each user (b) and total number of outlier above threshold GPS points with respect to threshold T (c).

Choosing threshold for Cabspotting dataset For Cabspotting dataset, 10909769 total visits are performed by users. Those visits are performed to 6045606 (unique) GPS points in total. This means that each GPS point gets 2 visits on average. So, it is not possible to build visit histories with sufficient-length from original GPS points unless significant locations are generated.

Significant location generation results for Cabspotting dataset are described by different shaped graphs in Fig. 4. The Fig. 4a does not provide much detail about the possible "knee" point. Note that, each cluster corresponds to one significant location. In Fig. 4a, the average number of significant locations rises up to a certain point and then drops. This can be explained by the fact that up to that point, possible connecting points between two groups of points are lost which causes an increase in the number of clusters, then after that point, clusters start to merge which causes a decrease. Therefore, it is not possible to find a reasonable knee point in Fig. 4a. In Fig. 4b and Fig. 4c, it is possible to distinguish a knee point (which is not explicitly drawn) using a similar way applied to CenceMe GPS dataset. However, we are not able to get an optimal set of clusters using a possible knee point which is practical in terms of predictability. In fact, a simple glance over all thresholds (including the threshold of a possible knee point) after $4.5 \cdot 10^{-8}$ does not yield better results. We discover that at $4.5 \cdot 10^{-8}$, a steep transition occurs in the other direction such that in Fig. 4b and Fig. 4c, average significant time and the total number of outlier above threshold GPS points start to fall rapidly. This implies that we lose more statistically important GPS points (which become below threshold) to the right of $4.5 \cdot 10^{-8}$ which negatively affects the predictability of the dataset. Moreover, a rapid drop in the number of outlier above threshold GPS points to the right of $4.5 \cdot 10^{-8}$ reinforces this argument; since more GPS points become below the threshold so that fewer GPS points are clustered which produces a small number of outlier above threshold GPS points. Therefore, we utilize $4.5 \cdot 10^{-8}$ as a threshold for Cabspotting dataset. There are 1326732 outlier above threshold GPS points at $4.5 \cdot 10^{-8}$. So, each user loses on average 2475 GPS points. This is still small as compared to the average number of significant visits per user $v_s = 17272$. It is possible to assign every outlier above threshold GPS point to the nearest cluster, however, this results in low predictive performance.

4.2.4 Properties of Preprocessed Datasets

The preprocessed datasets have different properties as reported in Table 1. Cabspotting and CenceMe GPS datasets have a small number of users as compared to other datasets. The chosen thresholds produce on average 3 and 19 significant locations per user for CenceMe GPS and Cabsptting datasets, respectively. Ile Sans Fils dataset has less than 1 significant location per user as compared to the fact that it has thousands of users. This dataset is constructed from WiFi logs of access points in public places where most of the users are observed in few locations. Moreover, public access points do not likely provide much significance as compared to work or home locations. Another difference is present at residence times of users in significant locations. Depending on a trace length, Dartmouth WiFi and Ile Sans Fils dataset exhibit 8 days and 5 days of average residence time, respectively. Trace length of Cabspotting and CenceMe GPS dataset is small, however, they still exhibit 1 day and 2 days of residence time.

The last column is of Table 1, i.e. "average percentage of time spent by each user in significant locations" is important for the predictive performance of the given prediction method. In Cabspotting and CenceMe GPS datasets, each user spends on average 87.46% and 61% of his time in significant locations. This parameter decreases to 54.69% for Dartmouth WiFi and 4.52% for Ile Sans Fil dataset. Low average significant time for Ile Sans Fils dataset can be explained by the fact that trace length is longer and its users exhibit less regularity during the measurement period.

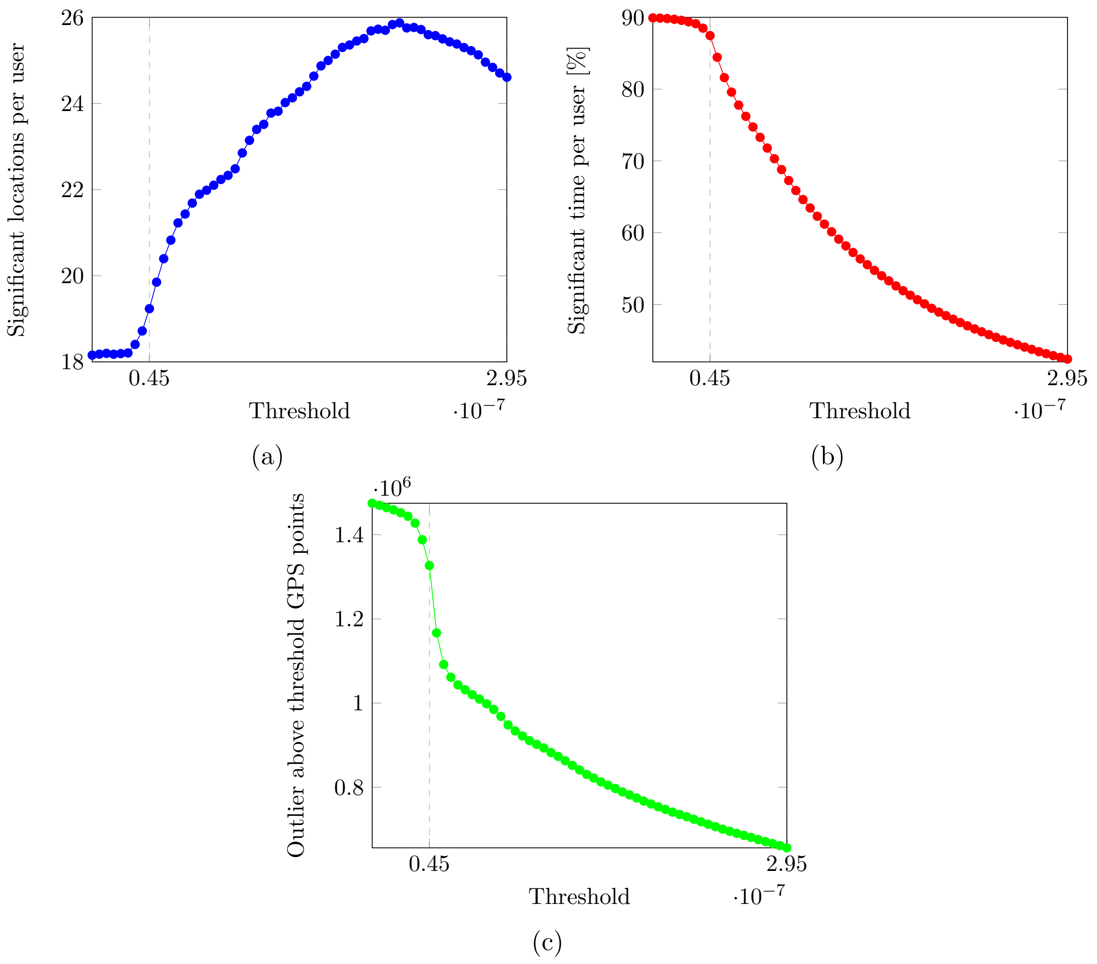


Figure 4: Threshold-based significant location extraction from Cabspotting dataset. The functions of average number of significant locations per user (a), average proportion of time spent in significant locations by each user (b) and total number of outlier above threshold GPS points with respect to threshold T (c).

4.2.5 Elimination of Users Based on The Number of Significant Locations

We eliminate some users from the datasets depending on the generation of significant locations. For instance, 4847 out of 13888 users are discarded in Dartmouth WiFi dataset, because for those users significant locations cannot be generated. This number is 66237 for Ile Sans Fils dataset. A similar case happens for GPS datasets such that in Cabspotting dataset only 1 user and in CenceMe GPS dataset 2 users are discarded. Table 2 shows some fixed features of Table 1 for non-discarded users in these datasets.

Dataset	U	l_s	v_s	Significant time
Cabspotting	535	19.27	17305	87.52%
CenceMe GPS	17	3.06	629	68.18%
Dartmouth WiFi	9041	8.78	3245	84%
Ile Sans Fils	3452	1.17	91	91.29%

Table 2: Fixed features of datasets for non-discarded users.

When averaging for certain features from Table 1 is performed for non-discarded users (i.e. "significant" users), significant time increases from 4.52% to 91.29% for Ile Sans Fils and from 54.69% to 84% for Dartmouth WiFi. Accordingly, average significant locations l_s and average significant visits v_s also rise. This case is also common for Cabspotting and CenceMe GPS datasets. We perform predictions on the non-discarded users in these datasets.

4.3 Predictability Measure for The Time Series

To predict user behavior depending on the underlying time series, the existence of determinism needs to examined in the time series, namely the predictability of the time series needs to be investigated. The concept of determinism is explained in the second paragraph of the Section 3.

Consider the time series $(s_1, s_2, \ldots, s_N)$. If the follow-up measurement s_{N+1} is provided, then the prediction error is defined as the difference between s_{N+1} and the corresponding predicted value p_{N+1}. If a predictive method is provided, then it is feasible to generate the predictions $(p_1, p_2, \ldots, p_N)$ for the given time series. After that, *mean squared prediction error* (MSPE) can be calculated as $\varepsilon = \frac{1}{N}\sum_{n=1}^{N}(s_n - p_n)^2$. Bigger ε implies that the generated predictions are not accurate and the given time series is not predictable.

The value of ε is assessed with the respect to the variance σ^2 of the given time series $(s_1, s_2, \ldots, s_N)$. The calculation of the variance is simple. First, mean of the values is calculated as $\tilde{s} = \frac{1}{N}\sum_{n=1}^{N} s_n$. Then, the variance is defined as $\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(s_n - \tilde{s})^2$ [Kantz and Schreiber, 2004]. To determine whether the ε is high or low, the ratio of $\frac{\varepsilon}{\sigma^2}$ is taken. If the ratio is close to 1, then MSPE is big and if it is close to 0, then MSPE is small. This ratio is called *predictability error* [Scellato et al., 2011] of the given prediction method. The original value of ε could get varying values without any meaning. Reasonable approach is to compare it against the average number of fluctuations that a time series carries. Division by the variance normalizes ε and makes it possible to compare the prediction accuracy of various time series.

We utilize this measure to evaluate the predictability for the underlying time series of user visit histories to different locations in the presented datasets. For this purpose, each visit history of a user is divided into two parts and underlying time series of each part is obtained. Then, predictive model is built on the first part and values for second part are predicted. Finally, the values in the original second part and predicted second part are compared. Predictability error equal to 1 implies that no determinism is available in the time series, since $\varepsilon = \sigma^2$; every prediction generated by the predictor is equal to mean value of the time series, i.e. $p_n = \tilde{s}$, $\forall n \in \{1, 2, \dots, N\}$. However, predictability error closer to 0 implies reasonable level of determinism in the time series.

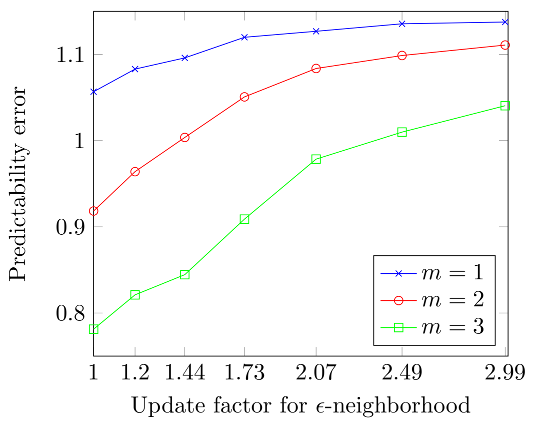
4.3.1 Density of ϵ-neighborhoods

Before providing results for datasets, we discuss how the density of the ϵ-neighborhood affects the prediction accuracy. As presented in Section 2, the time series $(s_1, s_2, \dots, s_N)$ is embedded in the embedding space $(\beta_{(m-1)\cdot\nu+1}, \beta_{(m-1)\cdot\nu+2}, \dots, \beta_N)$ with the embedding parameter m and the delay parameter ν. To generate predictions, the neighborhood of the last embedding vector β_N is constructed. Since we utilize ϵ-neighborhood, then the neighborhood of β_N is portrayed as $U_\epsilon(\beta_N)$. In this case, the density of ϵ-neighborhood of β_N is the number of elements in its neighborhood, namely $|U_\epsilon(\beta_N)|$. Note that, for the underlying time series of each visit history such last embedding vector exists and we investigate whether expanding a neighborhood of this vector can improve the predictability of the respective visit history (more formally, the respective underlying time series).

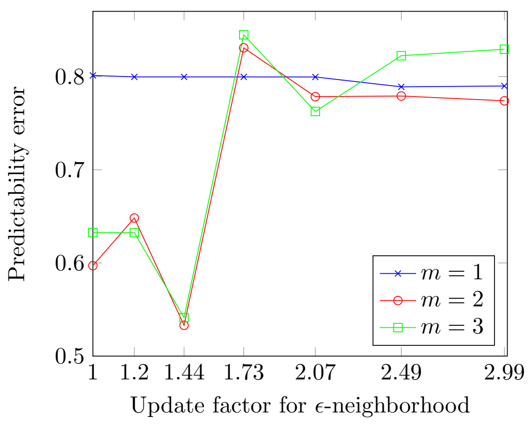
We analyze the predictability error (pe) of datasets with different densities of ϵ-neighborhood. To calculate the (average) predictability error of a dataset, predictability error for each user is calculated, the results are aggregated and divided by the number of users in the dataset. Average predictability error of a user is calculated using a similar routine: predictability error for each visit history of a user are summed up and the result is divided by the number of visit histories.

For each visit history, the radius of the ϵ-neighborhood is taken as 10% of the standard deviation of the underlying time series [Kantz and Schreiber, 2004]. The standard deviation is simply the square root of the variance σ^2 of the time series. During the neighborhood search, it is possible that ϵ-neighborhood can get empty which can result in no prediction for the time series. In this case, there are two options; either time series should be discarded in predictability error calculation, or ϵ-neighborhood should be extended and neighborhood search should be repeated hoping for a non-empty neighborhood. We investigate how the extended neighborhood affects the predictability error of a dataset. We split underlying time series of each visit history of a user to train and test splits; train split contains all values except the last value and test split only contains the last value of the time series. This experiment can be regarded as a simple 1-step ahead predictability error calculation for a time series. MSPE is normalized with respect to the variance of the original time series of each visit history. For convenience, in all other predictability error experiments, we apply the same normalization routine. Standard deviation is calculated from the train split since a neighborhood search is performed there.

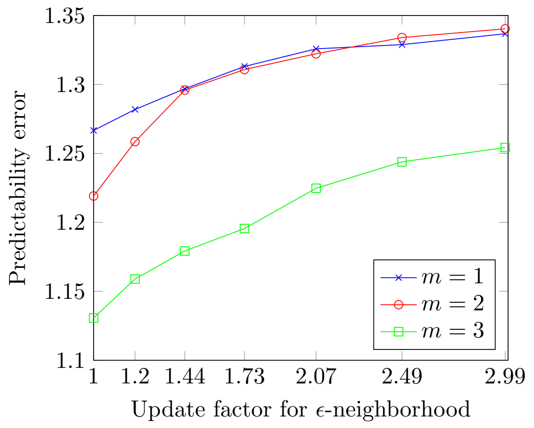
We generate the predictability errors for the original ϵ-neighborhood and its extensions which are updated by the factor of $(1.2)^i$, $\forall i \in \{1, 2, 3, 4, 5, 6\}$. During predictability error calculation, if ϵ neighborhood gets empty for some visit history of a user, we extend the neighborhood by the given factor and perform the search again. If neighborhood still gets empty, we discard that visit history from the calculation, such that all other non-empty neighborhood visit histories of a user are considered in the calculation. If all visit histories of a user get empty neighborhood in spite of an extension, we explicitly set the predictability error of a user to 1 marking that no determinism is available for that user. We depicted the results for different ϵ-neighborhood densities in Fig. 5.



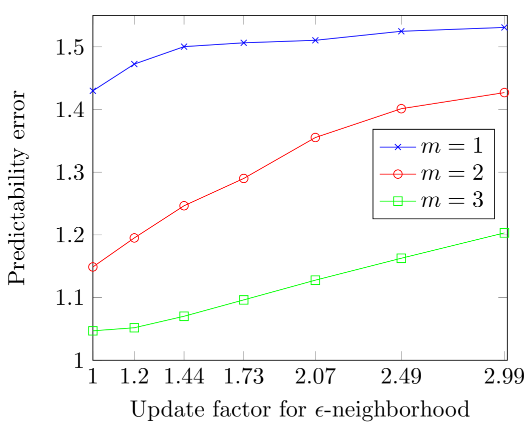
(a) Cabspotting



(b) CenceMe GPS



(c) Dartmouth WiFi



(d) Ile Sans Fils

Figure 5: Average predictability error of datasets with respect to different extended ϵ-neighborhoods by the factor of $(1.2)^i$, $\forall i \in \{0, 1, 2, 3, 4, 5, 6\}$.

For Cabspotting, Dartmouth WiFi and Ile Sans Fils datasets, it is clear from Fig. 5a, Fig. 5c and Fig. 5d that updated neighborhoods do not provide better predictability errors. However, for larger values of embedding dimension m, lower predictability errors are obtained which demonstrates the optimality of the non-linear models in predictions. In Fig. 5b, predictability errors for CenceMe GPS datasets are shown. CenceMe GPS dataset exhibits fluctuations in predictability errors for different ϵ-neighborhood densities. This is due to the fact that the number of users and number of visit histories for each user are very small and fluctuations are not smoothed out through the division by the number of users and by the number of visit histories, respectively. But, original

ϵ-neighborhood still provides useful predictability errors for $m = 1, 2, 3$. This experiment indicates that using original ϵ-neighborhood for predictions is a better option.

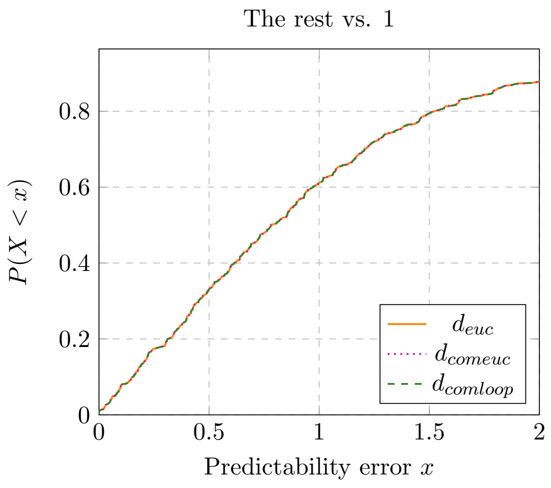
4.3.2 Predictability Errors With Respect to Different Distance Functions

We test predictability errors for different distance functions to choose a reasonable one for location predictions. For this purpose, we generate the Cumulative Distribution Function (CDF) of predictability errors for users in each dataset. Predictability errors are obtained from the underlying time series of user visit histories for a train and test split of "The rest vs. 1". Time series of size N is divided into two splits such that train split contains the first $N-1$ elements and test split contains only the last element. The results can simply be specified as 1-step ahead predictability errors of the time series of visit start times generated for different values of embedding dimension m. We think this heuristic would give an idea of how instants in the time series of different users are distributed and how they should be compared.

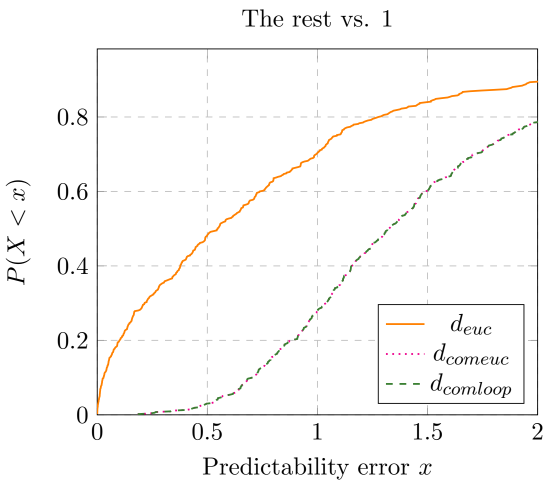
We select one traditional distance function and two "comparative" distance functions defined in Section 3.3. We compare Euclidean distance d_{euc}, comparative Euclidean distance d_{comeuc} and comparative looping distance $d_{comloop}$ for different values of embedding dimension m. These distance functions are defined for our non-linear prediction method such that they find distances between embedding vectors. Assume that, embedding vectors $\beta_3 = (s_1, s_2, s_3)$ and $\beta_6 = (s_4, s_5, s_6)$ are defined for a time series $(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$. Euclidean distance d_{euc} is based on the comparison of mutual components between these embedding vectors through the instant distance d, namely, through $d(s_1, s_4)$, $d(s_2, s_5)$ and $d(s_3, s_6)$. However, comparative distances d_{comeuc} and $d_{comloop}$ are based on the comparison of mutual intervals namely $d(s_1, s_2)$ and $d(s_4, s_5)$, $d(s_2, s_3)$ and $d(s_5, s_6)$. Comparative distances can be used to detect patterns at specific time intervals rather than at specific times of a day.

We present results for two datasets; Cabspotting and Ile Sans Fils. Results of the other datasets are similar to the results obtained for Cabspotting dataset and can be found in Appendix A. For all datasets, we select a set of common users such that none of the users has an empty ϵ-neighborhood in all visit histories. This means that for these users, some visit histories can have an empty neighborhood, but not all of them are empty. Apart from that, the predictability error of each user is calculated in a similar fashion as defined in Section 4.3.1.

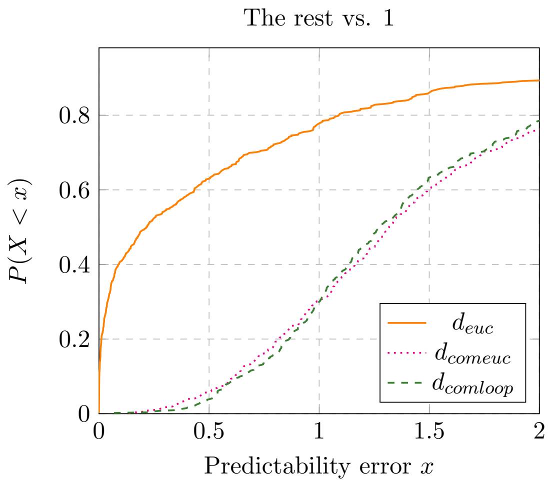
We choose 535 common users out of 535 and 134 common users out of 3452 for Cabspotting and Ile Sans Fils datasets, respectively. In Fig. 6a and Fig. 7a, CDF of the predictability errors with respect to the given distance functions are compared for embedding dimension $m = 1$. Note that, for $m = 1$, the distances are equivalent, i.e., $d_{euc} = d_{comeuc} = d_{comloop}$ and 60% and 62% of the respective datasets shows predictability error smaller than 1. In Fig. 6b, where $m = 2$, d_{euc} exhibits lower predictability errors than d_{comeuc} and $d_{comloop}$. The d_{euc} makes 70% of the Cabspotting dataset have predictability error smaller than 1. For $m = 2$, d_{comeuc} is equivalent to $d_{comloop}$. In Fig. 6c, where $m = 3$, d_{euc} makes 78% of the Cabspotting dataset have predictability error smaller than 1 surpassing d_{comeuc} and $d_{comloop}$. In Fig. 6d, we investigate the number of users with $pe \geq 2.0$ for the given distance functions. When m gets bigger, the number of users with a high predictability error increases for d_{comeuc} and $d_{comloop}$. However, this is not the case for d_{euc} for which stable number of users gets a high predictability error. The obtained results indicate that d_{euc} is able to capture patterns better than d_{comeuc} and $d_{comloop}$ for Cabspotting dataset.



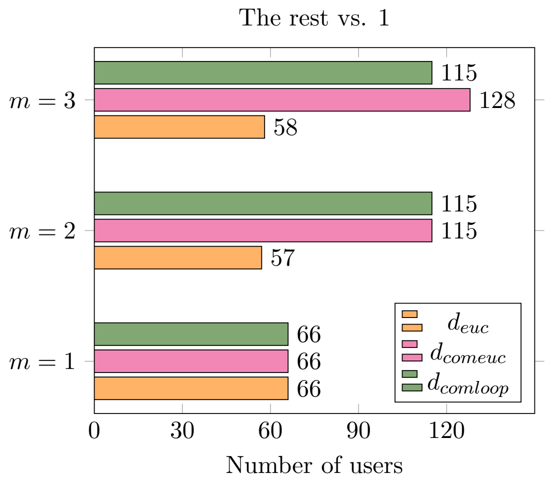
(a) Embedding dimension $m = 1$



(b) Embeddig dimension $m = 2$



(c) Embedding dimension $m = 3$



(d) $pe \geq 2.0$

Figure 6: CDF of 1-step ahead predictability errors for Cabspotting dataset with respect to different embedding dimensions and different distance functions.

Ile Sans Fils dataset demonstrates different characteristics for the given distance functions when $m = 2$ and $m = 3$. In Fig. 7b, where $m = 2$, almost 60% of the dataset shows predictability error smaller than 1 for d_{euc}, d_{comeuc} and $d_{comloop}$. However, d_{comeuc} and $d_{comloop}$ makes 80% of the dataset have predictability error smaller than 2 as compared to d_{euc} which yields 75%. A similar routine happens in Fig. 7c, such that d_{comeuc} and $d_{comloop}$ yield almost 80% and d_{euc} yields 70%. Note that, significant locations in Ile Sans Fils dataset are public WiFi access points. Users can show less regularity in these locations than in possibly more important locations (e.g. work or home) in terms of patterns at specific times of a day. From Fig. 7b and Fig. 7c, we discover that users tend to show more regular patterns in terms of specific time intervals through time interval based functions d_{comeuc} and $d_{comloop}$. Instead of visiting significant locations more regularly at specific times, users in Ile Sans Fils dataset tend to visit these locations at specific time intervals. They possibly visit a significant location at non-specific times of a day, but the time intervals between these visits become more regular. The Fig. 7d also supports this argument. It depicts how all 3452 users in Ile Sans Fils dataset react to different distance functions in terms of empty neighborhood visit histories. For d_{euc}, number of users which is lost (i.e. having an empty neighborhood in all visit histories) becomes 3308 and 2600 for $m = 3$ and $m = 2$, respectively. However, this case does not happen for d_{comeuc} (at least for $m = 2$) and $d_{comloop}$.

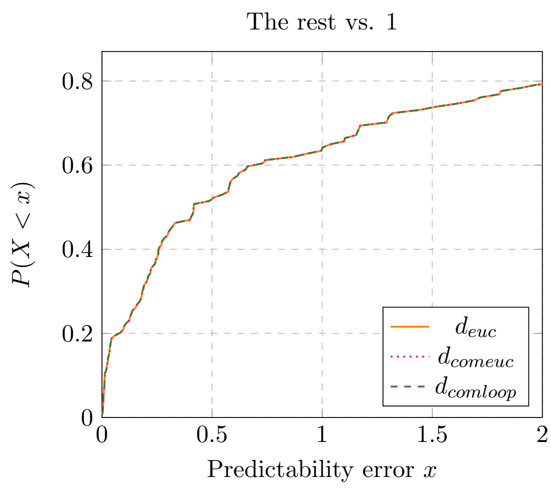
In Fig. 7b and Fig. 7c, we also discover that percentage of the dataset which has predictability error less than 0.5 is higher for d_{euc} than for d_{comeuc} and $d_{comloop}$. Higher number of users with predictability error close to 0 can be useful for location predictions and since, d_{euc} shows performance similar to d_{comeuc} and $d_{comloop}$ for the percentage of the dataset which exhibits predictability error less than 1. Therefore, we select d_{euc} as a distance function for Ile Sans Fils dataset in our further evaluation. We would pick a "good" set of users for locations predictions, therefore d_{euc} would not be affected by the "lost users" problem for Ile Sans Fils dataset. On the other hand, d_{euc} would also make location prediction accuracy comparable to the other datasets, since, for those datasets, we utilize d_{euc}.

4.3.3 Predictability Errors With Respect to Different Train and Test Splits

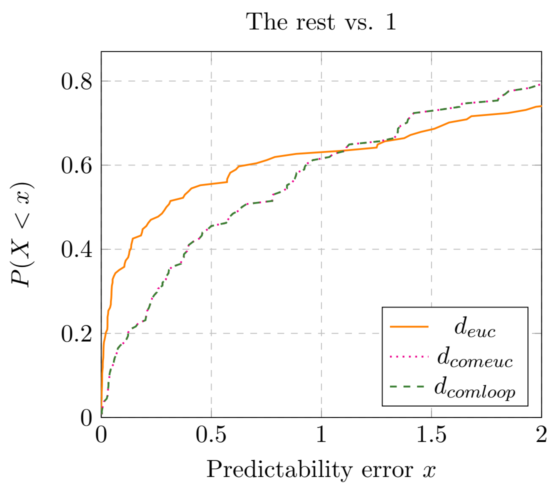
To identify the determinism in the datasets for different train and test splits, we perform four experiments. In each experiment, we divide the visit histories of users in each dataset into two parts; the first part is used for training and the second part is used for testing. We generate the results for four train and test splits: "The rest vs. 1", "90% vs. 10%", "70% vs. 30%" and "50% vs. 50%". We again choose a common set of users for each dataset such that none of the users has an empty neighborhood in all its visit histories. Some visit histories of users can have an empty neighborhood. In this case, those visit histories are discarded in predictability error calculation. For a test split having more than one element, we compare its elements to the predicted elements in index order. Assume that, the time series $(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$ is given. Further assume that, 70%-30% split is performed such that the time series is divided into to train split $(s_1, s_2, s_3, s_4, s_5, s_6, s_7)$ and test split (s_8, s_9, s_{10}). In this case, possible predictions (p_8, p_9, p_{10}) are compared to (s_8, s_9, s_{10}) and predictability error is calculated using the formula presented in Section 4.3. It is also possible to obtain predictions (p_8, NULL, NULL) where NULL denotes no prediction when no neighbors are found. For convenience, we still calculate the predictability error for this case comparing p_8 and s_8

through discarding NULL predictions and (s_9, s_{10}).

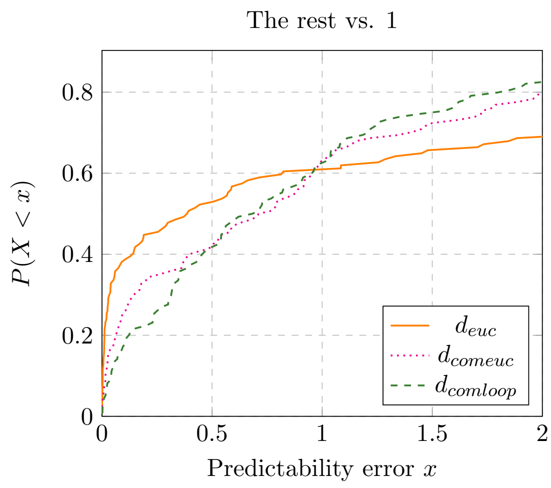
We report the results for different values of embedding dimension m for Cabspotting and Ile Sans Fils datasets. The results obtained for the other datasets are similar to the results of Cabspotting dataset and are available in Appendix B.



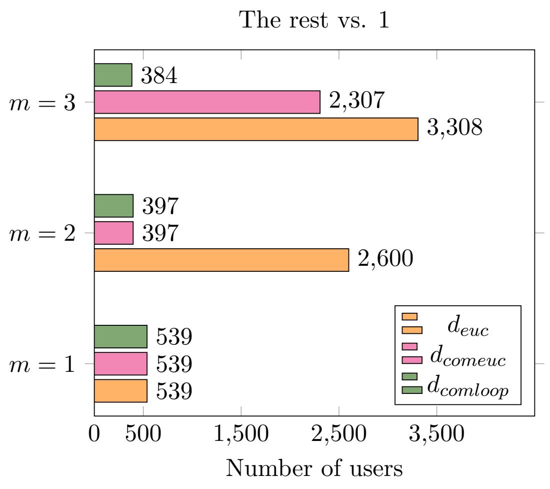
(a) Embedding dimension $m = 1$



(b) Embeddig dimension $m = 2$



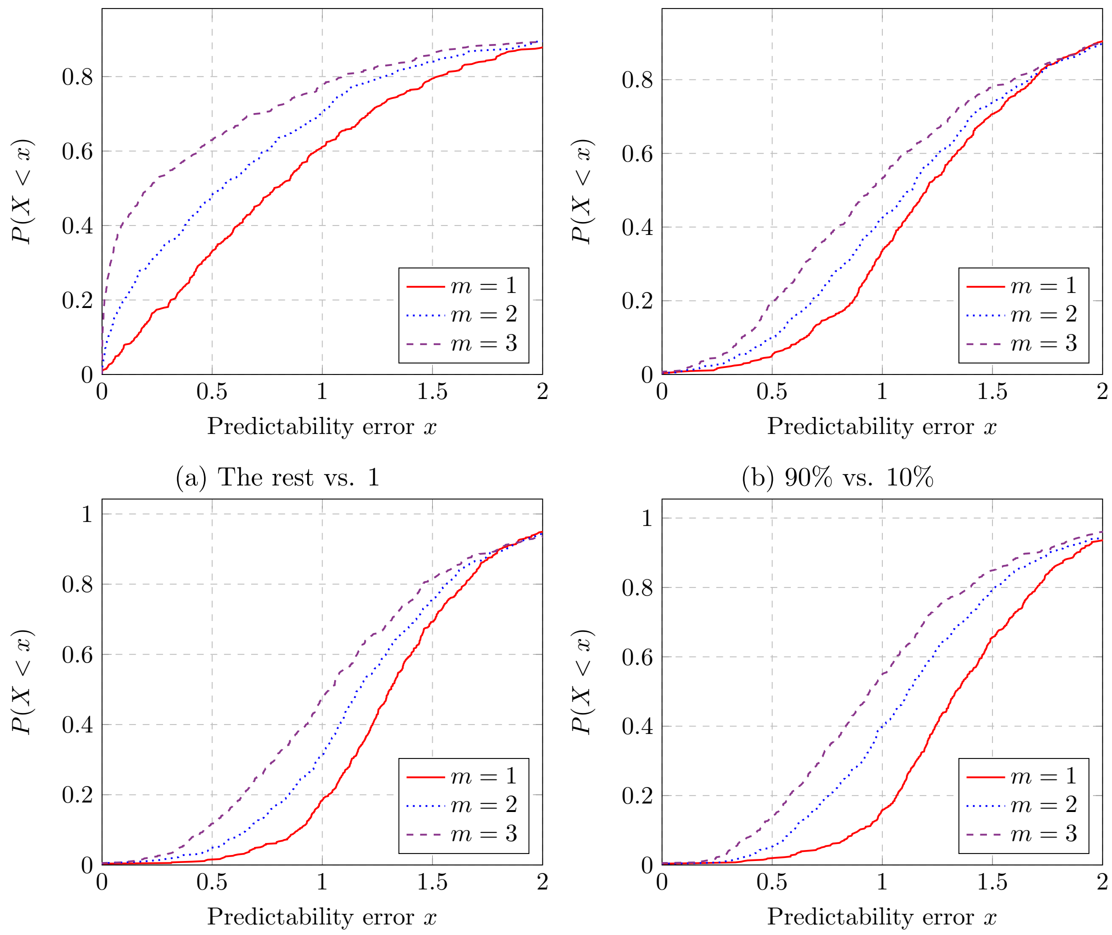
(c) Embedding dimension $m = 3$



(d) All visit histories have $|U_\epsilon| = 0$

Figure 7: CDF of 1-step ahead predictability errors for Ile Sans Fils dataset with respect to different embedding dimensions and different distance functions.

In Fig. 8 and Fig. 9, Cumulative Distribution Functions of the predictability errors for the time series of visit start instants for different values of embedding parameter m and for different train and test splits are shown. The Fig. 8a and Fig. 9a demonstrate 1-step ahead predictability errors for Cabspotting and Ile Sans Fils dataset. For Cabspotting dataset, 60%, 70% and 78% of the dataset show predictability error less than 1 for embedding dimensions $m = 1, 2, 3$, respectively. For Ile Sans Fils dataset, 62%, 62% and 60% of the dataset show predictability error less than 1 for the respective embedding dimensions.



(a) The rest vs. 1

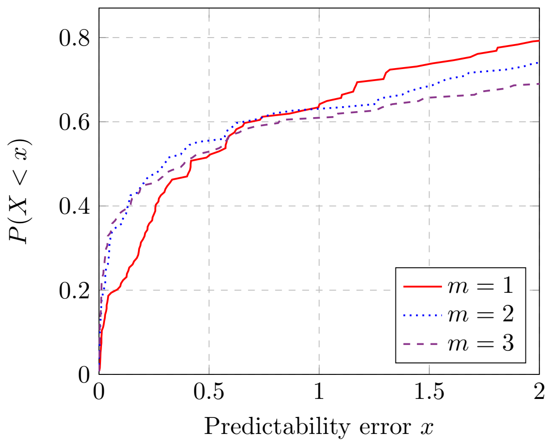
(b) 90% vs. 10%

(c) 70% vs. 30%

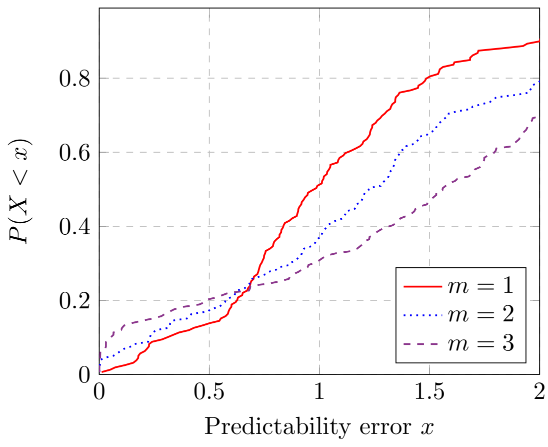
(d) 50% vs. 50%

Figure 8: Cumulative Distribution Function of predictability errors for Cabspotting dataset with respect to different embedding dimensions and different train and test splits.

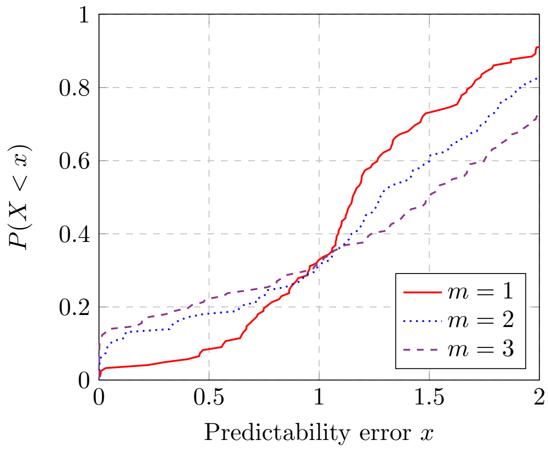
For the other three train-test splits, Cabspotting dataset shows similar characteristics. When the number of elements in the train and test splits are getting close to each other, the percentage of the dataset showing predictability error less than 1 drops accordingly. This is natural since our prediction method predicts further values in the future which causes a potential drop in the prediction accuracy. However, for each train and test split, higher values of m result in lower predictability errors. The same phenomenon also occurs in CenceMe GPS and Dartmouth WiFi datasets (see Fig. 18 and Fig. 19). This indicates that the nonlinear prediction method augments the quality of predictions since it can identify particular patterns of user visits and forecast the time of the next visit. However, our findings show that for embedding dimensions $m \geq 4$, the predictive performance worsens. This is because the datasets do not have enough statistical potential for higher dimensional embeddings to obtain an accurate prediction.



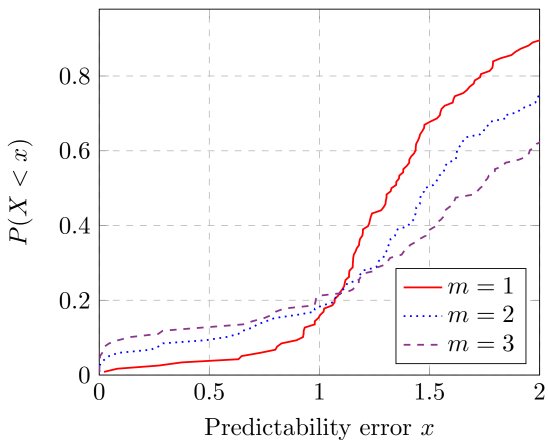
(a) The rest vs. 1



(b) 90% vs. 10%



(c) 70% vs. 30%



(d) 50% vs. 50%

Figure 9: Cumulative Distribution Function of predictability errors for Ile Sans Fils dataset with respect to different embedding dimensions and different train and test splits.

We obtain different behavior for Ile Sans Fils dataset in the other three train-test splits. For example, for “90% vs. 10%”, $m = 1$ makes 50% of the dataset have predictability error less than 1 as compared to $m = 2$ and $m = 3$ which yield 38% and 35%, respetively. For “70% vs. 30%” and “50% vs 50%”, all embedding dimensions show similar performance for the percentage of the dataset having predictability error less than 1 as we discover in the other datasets. In Fig. 9c, all embedding dimensions make around 35% of the dataset have predictability error less than 1. In Fig. 9d, $m = 3$

shows better performance. However, for all splits, $m = 1$ outperforms $m = 2$ and $m = 3$ for predictability errors less than 2 as shown in Fig. 9b, Fig. 9c and Fig. 9d. The reason is that some users do not visit their significant locations in similar patterns, since significant locations are public places. For some users it is also possible that their time series is not predictable in higher dimensional embedding (e.g. there is no m-consecutive pattern to capture for $m = 2$ and $m = 3$).

For all datasets, on average, a large number of users displays predictable behavior so that we can use them for location predictions. Only, Ile Sans Fils dataset exhibits a lower degree of regularity and it would be interesting to observe how the locations predictions get affected by this challenge.

4.4 Accuracy of Location Predictions

The performance of *Distributed NextPlace* is compared to the original NextPlace in terms of runtime and predictive performance. Before evaluating location prediction accuracy, we should choose a "good" set of users which can be utilized for location predictions. We divide the dataset into two splits and use the former for training and the latter for testing. More formally, we split each visit history of a user into two parts, make a visit prediction from the training part of each visit history, order the predicted visits in terms of time of a day and pick a location of a visit performed at a given instant of time as a final predicted location.

We evaluate how the number of users is affected by different train and test splits. We generated the results for 'the rest'-1, 90%-10%, 70%-30% and 50%-50% splits. The results are provided in Appendix C. We define the users as "usable" ones for whom not all visit histories have an empty neighborhood and a predictability error smaller than 2. The results indicate that when m gets larger, the number of "usable" users decrease. For all datasets, we pick the "usable" users obtained for embedding dimension $m = 3$ since the users which are "usable" for $m = 3$ are also usable for $m = 2$ and $m = 1$. For example, 5534 out of 9041 users become usable for the 90%-10% split and embedding dimension $m = 3$ in Dartmouth WiFi dataset. For each split, we utilize the "usable" users for location predictions.

We define the correctness of a location prediction as follows: if the prediction made at time T signifies the location l as a user's "next place" at time $T_P = T + \Delta T$, then this prediction is correct if the user is at the location l at the time which lies in the interval of $[T_P - \theta, T_P + \theta]$, where θ is defined as the error margin. A prediction algorithm can also forecast the case where the user will not be in any significant location. Therefore, the prediction can be considered correct if it signifies the user to be in a significant location l and he is in l or it signifies the user not to be in any significant location and in reality, he is not. In our evaluation, we focus on predictions which forecast the user to be in a significant location which is the primary goal for most of the location prediction algorithms.

To measure the accuracy of location predictions we utilize the accuracy metric called *prediction precision*. It is defined as the fraction of the number of correct predictions to the number of all performed predictions which estimates the user presence at a significant location.

We first report the results of the prediction accuracy at time T which is chosen uniformly at random from the interval [0, 86400]. In this experiment, we find out that how the NextPlace estimates the user location at any time T, rather than after ΔT interval. For this purpose, we divide each visit history of a user into two parts; use the first part to predict a visit at the future step K (starting from K=1). Then, we sort the visits in a "time of day order", pick a visit at the time which lies in the interval of $[T-\theta, T+\theta]$ and choose its location as a predicted significant location. The second parts of all visit histories of a user are joined together to form the list of "test" visits. From this list, we choose a visit v whose start time t and end time $t+d$ contains time T, namely $t-\theta \leq T \leq t+d+\theta$. Then, we compare the location of a visit v to the obtained predicted location to find the accuracy. Error margin is $\theta = 900$ seconds.

In all experiments, we perform 1000 predictions for each user for different values of embedding dimension m and then we compute the prediction precision. We performed the experiments in the 90%-10% split. Experiments on the other splits yield similar or lower predictive performance when the portion of the train split and test split is getting closer. Moreover, if the test split portion is taken smaller (e.g. 5%), then we end up with insufficient test data.

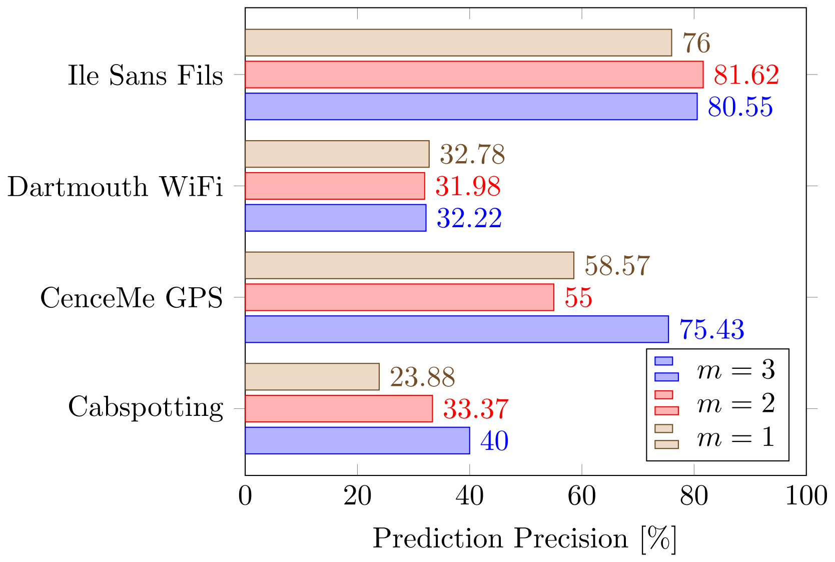


Figure 10: Prediction precision of different datasets at random time T for different values of embedding dimension m generated by the NextPlace predictor.

In Fig. 10, prediction precision of different datasets at random time T with respect to $m = 1, 2, 3$ are shown. We obtained higher prediction precision for the higher values of embedding dimension m in all datasets. So, higher values of m improve prediction quality. Our results indicate that Ile Sans Fils and CenceMe GPS datasets exhibit higher predictability which is around 80% and 75%, respectively, when $m = 3$. On the other hand, Cabspotting and Dartmouth WiFi datasets exhibit around 40% and 35%

for the same value of m. This can be explained by the fact that the average number of significant locations per user is 19 and 8 for Cabspotting and Dartmouth WiFi datasets, respectively and visit day seconds for different significant locations can lie in the same time interval affecting location prediction at the specific instant of time. Moreover, trace length of Dartmouth WiFi dataset is very long such that it is possible to find a user with a visit history which almost covers an entire interval of [0, 86400] by its visit start times. We talk more about this issue later when we report location prediction accuracy after ΔT interval.

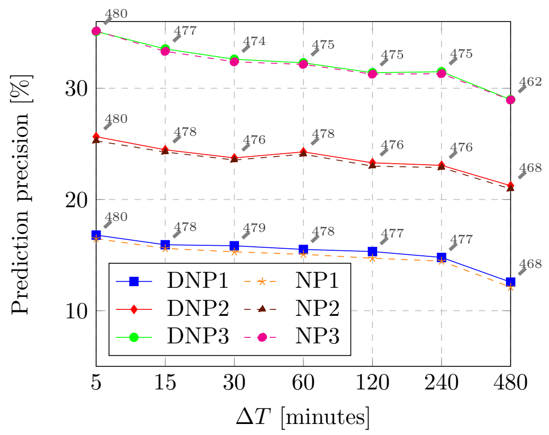
Note that, while calculating prediction precision we make few adjustments. For example, it is possible that no significant locations can be found at the specific time T from the test part of user visits. In this case, we do not consider that prediction in the number of attempted predictions, i.e., we do not increment the number of attempted predictions. Moreover, if no predicted location is obtained from the training part, we still do not consider that round in the number of attempted predictions. No predicted location can either mean a user will not be in any significant location or a location prediction cannot be obtained because of an empty neighborhood and/or the predictability limit when generating a future visit from each visit history of a user. Finally, the prediction precision of each user is summed up and averaged by the number of users which yields the prediction precision of the dataset.

In our next experiment, we report the results obtained for predictions after ΔT interval. The values of ΔT can be 5, 15, 30, 60, 120, 240 or 480 minutes. Location predictions from the training part of user visits are generated in a similar way. However, in this experiment, we find the real locations at time T and $T + \Delta T$ differently. We still join the visits from the test part of each visit history together into a list and sort this list in a "time of a day" order. Then we select one visit from this list uniformly at random, define the time T as a start time of this visit and its location becomes a user's real location at T. Then, to find a user's real location at $T + \Delta T$, we examine the visits which come after the selected visit in the list. If there is a visit v which contains $T + \Delta T$, then its location is chosen as a real location at $T + \Delta T$. Finally, the predicted location at $T + \Delta T$ is compared to the real location. Note that, "time of a day" ordering of test visits is important for correspondence since visit predictions from each visit history are also maintained by the same ordering. We have noticed that the chronological ordering of test visits negatively affects the predictive performance.

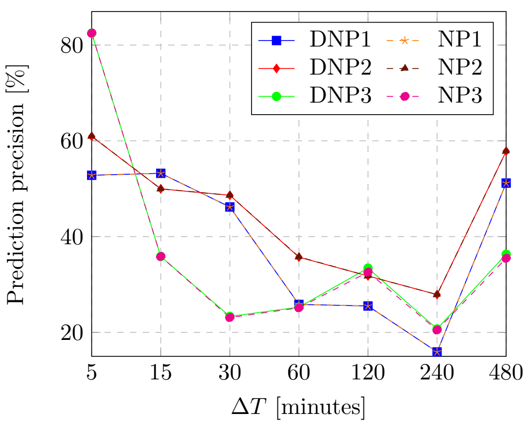
In Fig. 11, we demonstrate the prediction precision for different datasets obtained after ΔT interval. We compare two predictors: The serial NextPlace and Distributed NextPlace. The Distributed NextPlace has a probabilistic nature such that it does not return a random visit (where its location becomes a predicted location) from the list of future visits which happen at $T + \Delta T$. Instead, it returns a visit with the highest residence seconds by considering the fact that higher residence seconds can possibly indicate more importance.

We notice that higher values of m improves the quality of predictions for both predictors. The Distributed NextPlace predictor shows similar or slightly better performance than NextPlace in all datasets. We expected the Distributed NextPlace to show outperforming performance because of its probabilistic nature, however, we realized that choice of a visit with higher residence seconds can better assist in datasets where the measurements are performed for a long period and "average number of significant locations per user" is larger. This case is especially true for Dartmouth WiFi dataset and in all values of m, the Distributed NextPlace performs better than the original NextPlace. Interestingly, after $\Delta T = 15$ minutes, the predictors with $m = 1$ and $m = 2$ show better performance than the case where $m = 3$ in Dartmouth WiFi dataset. The reason is that after the long period, the patterns which can be captured by m-sequences ($m \geq 3$) may erode, i.e., more irregularities may appear in the user traces. For all values of m, we observe a falling trend in the predictions precision while ΔT becomes larger. This is because it becomes hard to estimate the user location after the long interval. However, a drop is not very sharp. When $m = 3$, we observe a drop of 33% to 29%, 38% to 19%, 80% to 38% and 82% to 39% in Cabspotting, Dartmouth WiFi, CenceMe GPS and Ile Sans Fils datasets, respectively. Reasonable predictive performance at higher values of ΔT demonstrates that the nonlinear prediction method which considers the temporal aspect of visits in significant locations is strong for long-term estimations. Another interesting fact is that the predictive performance obtained after $\Delta T = 480$ minutes is higher than some smaller values of ΔT in CenceMe GPS. Firstly, CenceMe GPS has a smaller number of users such that prediction precision is not smoothed out to proper value during averaging. Secondly, this dataset is constructed from users who perform measurements with their phones and after 8 hours (which is a typical number of working hours), such users tend to be at home or in residential buildings which represent more regular patterns. Note that, while ΔT gets larger, the number of users which can be predicted to be in a significant location after ΔT interval becomes smaller. In Fig. 11a, the number of users which can be predicted after ΔT interval drops from 480 to 462, 468, and 468 for $m = 1$, $m = 2$, and $m = 3$ between $\Delta T = 5$ minutes and $\Delta T = 480$ minutes in Cabspotting dataset. We do not provide these numbers for the other datasets because of maintaining clear a plot view in Fig. 11b, Fig. 11c and Fig. 11d.

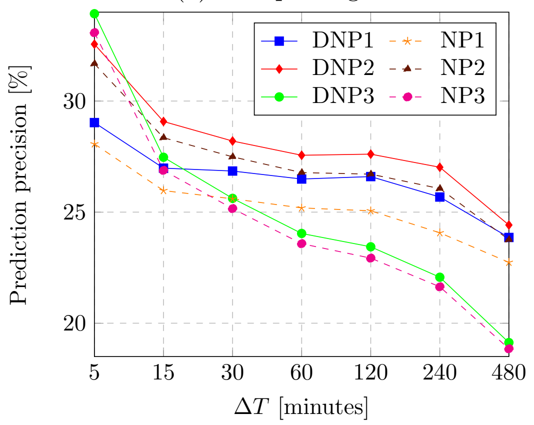
In Fig. 11, CenceMe GPS and Ile Sans Fils datasets demonstrate a prediction precision around 80% for $\Delta T = 5$ minutes. This precision then drops to 24% and 65% for $\Delta T = 60$ minutes, respectively. The precision is still lower for Cabspotting and Dartmouth WiFi datasets, as in the location predictions at random time T (shown in Fig. 10).



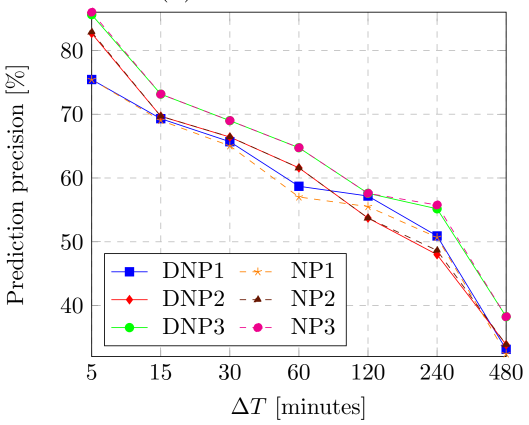
(a) Cabspotting



(b) CenceMe GPS



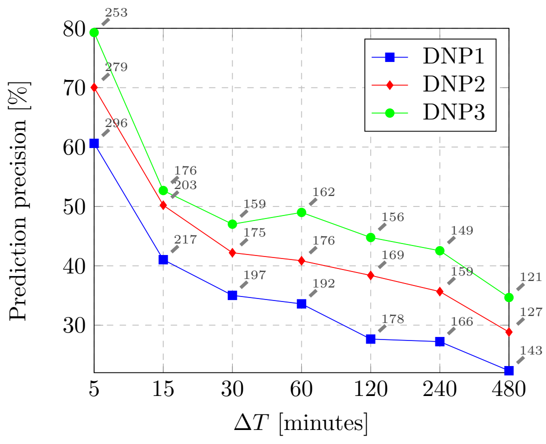
(c) Dartmouth WiFi



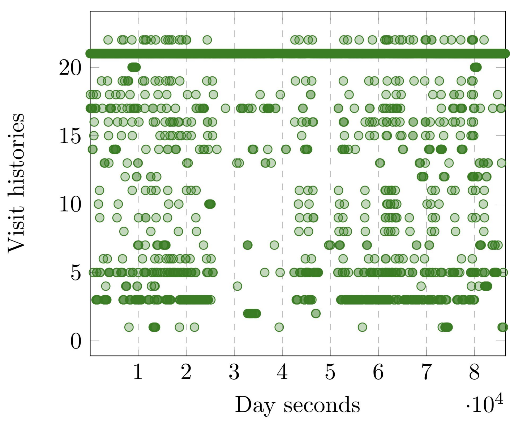
(d) Ile Sans Fils

Figure 11: Prediction precision as a function of time interval ΔT for different datasets and for different predictors: Nonlinear *Distributed NextPlace* predictor for the embedding dimension $m = 1, 2, 3$ (DNP1-DNP2-DNP3), Nonlinear *NextPlace* predictor for the same values of embedding dimension m (NP1-NP2-NP3). Error margin is $\theta = 900$ seconds. Pins of the coordinates represent the number of users which can be predicted after ΔT interval.

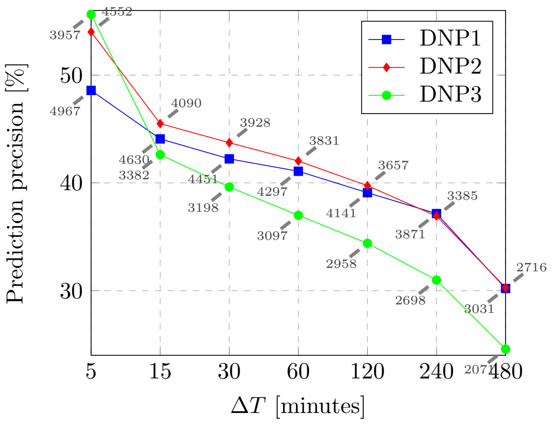
To find the reason for low performance in Cabspotting and Dartmouth WiFi datasets, we investigate how different visit histories of a user exhibit visit start times in the interval of [0, 86400]. In Fig. 12b, day seconds time span of 22 visit histories of a sample user from Cabspotting dataset and in Fig. 12d, day seconds time span of 7 visit histories of a sample user from Dartmouth WiFi dataset are shown. It is clear that between 40000-50000, 50000-60000, 60000-70000, 70000-80000 day seconds and other such mini-periods, different visit histories exhibit visits with the same or similar start times. In other words, a user visits different significant locations at similar times of a day (which can be possible in different days, in different weeks or in different months). Since, nonlinear predictor averages the elements found in the neighborhood, it is possible that different visit histories end up with visit predictions at $T + \Delta T$. Either the original NextPlace or Distributed NextPlace cannot pick a visit from those predictions whose location exactly matches with the location of real (test) visit at $T + \Delta T$. Statistical nature of such visit histories makes it possible that a user can be predicted to be in different significant locations at the same time. In this case, when less significant locations are considered, we should see higher prediction precision.



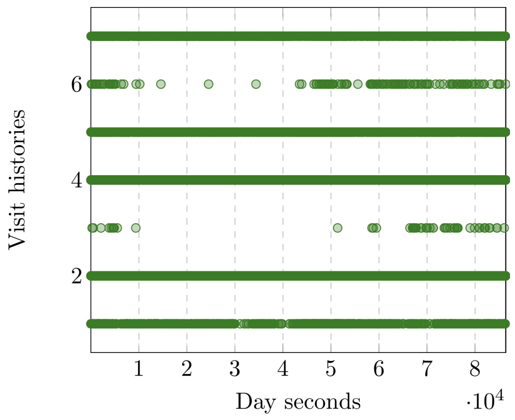
(a) Cabspotting



(b) Cabspotting



(c) Dartmouth WiFi

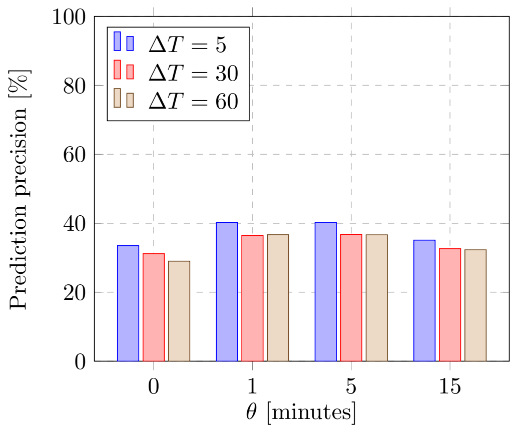


(d) Dartmouth WiFi

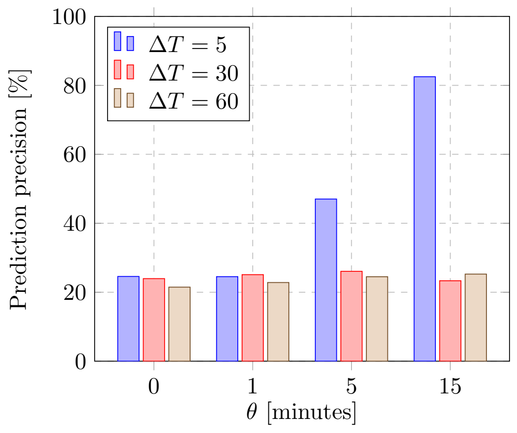
Figure 12: Prediction precision of Cabspotting dataset for users with 4 significant locations as a function of ΔT (a), visit time span of 22 significant locations of a sample user in Cabspotting dataset (b), prediction precision of Dartmouth WiFi dataset for users with 4 significant locations as a function of ΔT (c), visit time span of 7 significant locations of a sample user in Dartmouth WiFi dataset (d). Error margin is $\theta = 900$ seconds. Pins of the coordinates represent the number of users which can be predicted after ΔT interval.

In Fig. 12a and Fig. 12c, prediction precision of Cabspotting and Dartmouth WiFi datasets of users with 4 significant locations are shown. For each dataset, we randomly select 4 visit histories of each user and build a prediction model and testing based on them. We observe higher precision which rises from 34% to 80% in Cabspotting dataset and from 35% to 56% in Dartmouth WiFi dataset for the embedding dimension $m = 3$. For the other values of m, uptrend also occurs. These results support our argument from the previous paragraph.

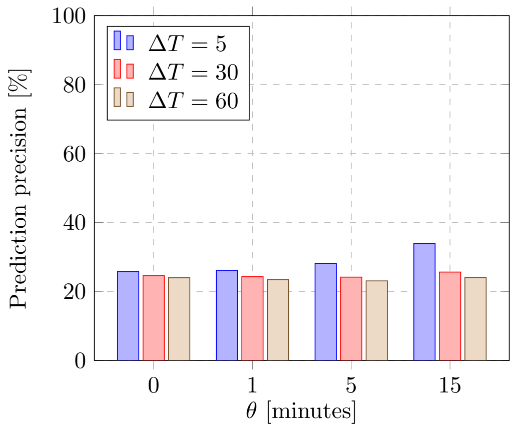
We also explore the effect of error margin θ on the prediction precision. The prediction precision becomes lower for smaller error margins, however, it still exhibits similar behavior for the presented predictors, for different values of m and for all datasets. In Fig. 13, we demonstrate how different error margins affect the prediction precision of the Distributed NextPlace predictor with $m = 3$ for some values of interval ΔT. For the predictions even without the error margin ($\theta = 0$), we obtain the results close to the case where $\theta = 15$ minutes (which is the choice in Fig. 11).



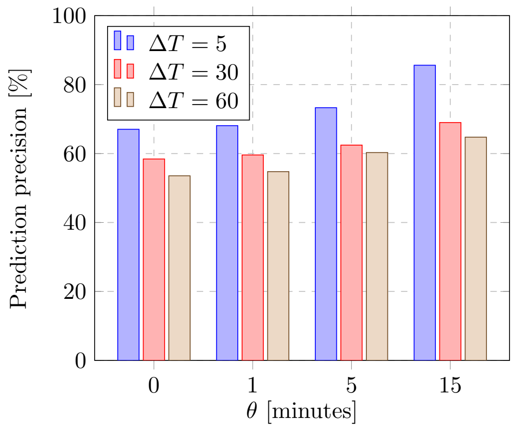
(a) Cabspotting



(b) CenceMe GPS



(c) Dartmouth WiFi



(d) Ile Sans Fils

Figure 13: Prediction precision of the Distributed NextPlace predictor with m = 3 as a function of error margin θ and for different values ΔT.

In Fig. 13c and Fig. 13d, higher error margin improves prediction precision for Dartmouth WiFi and Ile Sans Fils datasets. In Fig. 13a, we observe a decreasing performance for $\theta = 15$ minutes (900 seconds). This is because more locations can be seen at the time interval $[T_P - 900, T_P + 900]$ and picking the one which can exactly match with the real location of time T_P becomes harder. Since "average number of significant locations per user" is higher in Dartmouth WiFi dataset, we can also expect to see such decreasing behavior for the error margins larger than 15 minutes in this dataset. We also observe that a higher error margin improves the prediction quality in CenceMe GPS dataset, especially for $\Delta T = 5$ minutes. For this dataset, some fluctuating values for $\Delta T = 30$ and $\Delta T = 60$ minutes may arise from the averaging by a small number of users. Note also that, "average number of significant locations per user" is 4 in this dataset as compared to 1 in Ile Sans Fils dataset which exhibits regular increasing behavior in Fig. 13d.

Overall, our evaluation demonstrates that nonlinear predictor is able to achieve reasonable predictive performance after the long period and even at the arbitrary time picked uniformly at random from the interval [0, 86400]. Considering spatial aspects of recurrent patterns in significant locations turns out to be useful for both short-term and long-term estimations. Even after $\Delta T = 480$ minutes (8 hours), we achieve a prediction precision of at least 15% in all datasets as shown in Fig. 11. Finally, at the arbitrary time T, we obtain a prediction precision of at least 23% when all datasets are taken into account (see Fig. 10).

4.5 Speedup and Efficiency

To obtain better runtime, serial algorithms are parallelized on multiple processors. In this case, execution overhead is divided by several processors, concurrently. The runtime performance of a parallel algorithm is determined by its *speedup*. Let $T(n, 1)$ be the runtime of the serial algorithm and let $T(n, p)$ be runtime of the parallel algorithm executed on p processors, where n is defined as the size of the input. Then the speedup of the parallel algorithm is specified as [Bentz]

$$S(p) = \frac{T(n,1)}{T(n,p)}$$

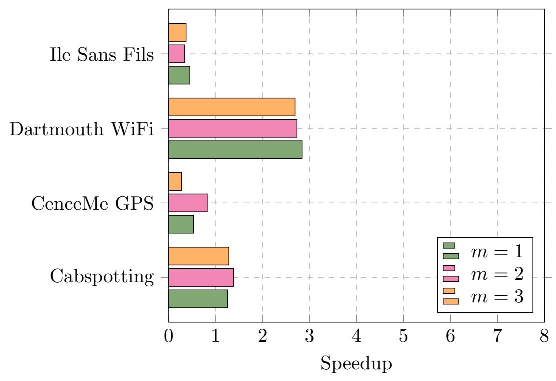
which is the ratio of serial execution runtime to the parallel execution runtime. Obtaining ideal speedup is unusual. It occurs when $S(p) = p$.

Another measure to examine runtime performance of a parallel algorithm is called *efficiency*. It is defined as [Bentz]

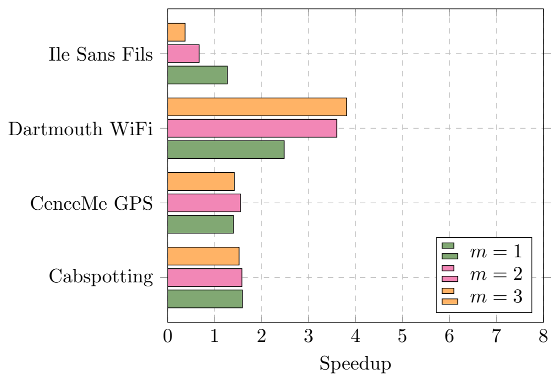
$$E(p) = \frac{T(n,1)}{p \cdot T(n,p)} = \frac{S(p)}{p}$$

When p is fixed at some value, the speedup and efficiency differ by a factor of p.

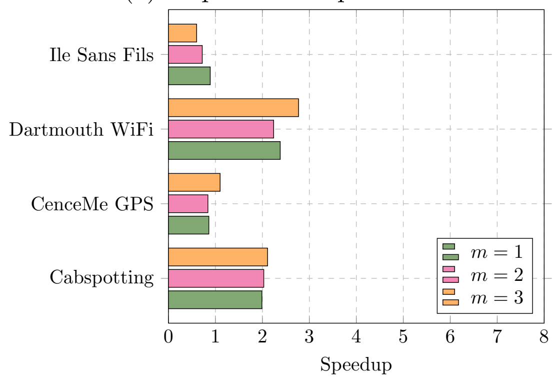
Distributed NextPlace algorithm improves the runtime of original NextPlace algorithm through parallelization. We report the speedup of the Distributed NextPlace algorithm with respect to the serial NextPlace for different datasets in Fig. 14. All experiments are performed on the same computer with *Intel(R) Core(TM) i7-3610QM* processor using Apache Spark framework [Zaharia et al., 2012]. For the distributed processing, environment provides 8GB of RAM with 8 CPUs at the frequency of 2.3 GHz. Parallelism level p (number of processors or CPUs for execution) is taken as 8. Number of partitions is defined by the number of users. In other words, the parallelization is at the user level. For each user, we perform different number of predictions at $T + \Delta T$ for each value of embedding dimension ($m = 1, 2, 3$) and average the respective executions times to obtain average speedup of the Distributed NextPlace algorithm. Efficiency of the Distributed NextPlace algorithm can be obtained similarly.



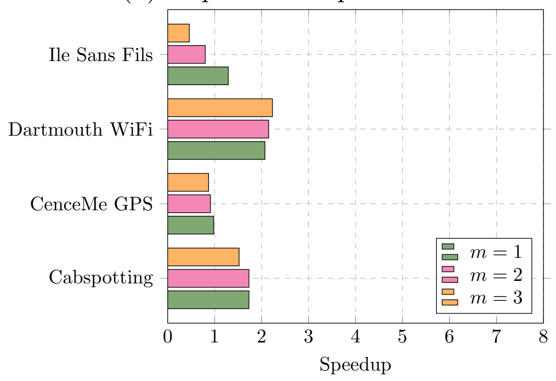
(a) 20 predictions per user



(b) 50 predictions per user



(c) 70 predictions per user



(d) 100 predictions per user

Figure 14: Average speedup of the Distributed NextPlace algorithm as a function of number of predictions per user for differenct datasets and for different values of embedding dimension m.

In Fig. 14, we observe that the speedup is below 1 for CenceMe GPS and Ile Sans Fils datasets in most cases. This means that the serial algorithm runs faster than the parallel algorithm. This can be explained by three facts: *i)* number of users in these datasets is small, such that at user level parallelization, communication time between processors may dominate the core runtime of parallel algorithm, *ii)* it is possible that with small number of predictions, the algorithm does not find a real location at $T + \Delta T$ and ends the procedure earlier, *iii)* maximum number of visits performed by a user in each dataset is around 3500 which can be explored faster by the serial algorithm to generate predictions.

For Dartmouth WiFi dataset, the Distributed NextPlace runs approximately 3 times faster than the original algorithm as reported in Fig. 14. For Cabspotting dataset, it runs up to 2 times faster. Note that, there are 5477 ("usable") users in Dartmouth WiFi dataset and 480 users in Cabspotting dataset. A maximum number of visits performed by a user in each dataset is 49429 and 38125 respectively. We notice that when number of predictions per user changes from 20 to 50 (in Fig. 14a and Fig. 14b) speedup increases. However, increasing the number of predictions further makes the runtime of the parallel algorithm be close to the runtime of the serial algorithm. Hence, speedup drops accordingly. This is because predictions are performed for each user in a sequential manner, therefore more distributed sequential predictions make the parallel environment have runtime close to the serial environment. Note also that, number of CPUs of our machine is 8, therefore natural parallelism level is taken as 8. We expect that when the execution is distributed in more processors (e.g. where the number of processors is close to the number users), such a drop in speedup is not likely to occur.

In the datasets we utilize, the number of users and the length of a movement trace of each user are not very large. In real-world applications, a huge number of location predictions are performed for millions of users with years of movement trace. For such a case, our evaluation indicates that the Distributed NextPlace will outperform the original NextPlace up to a million times or above.

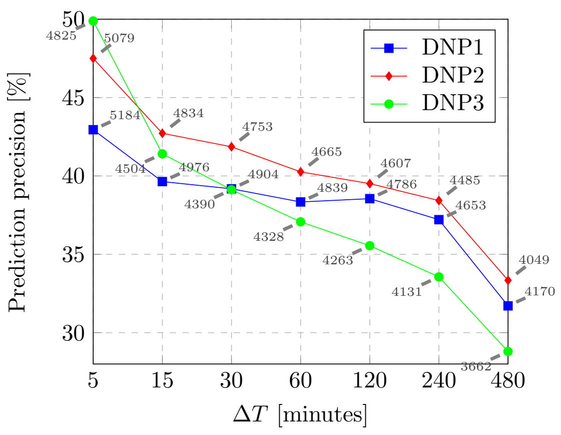
We have also tested the Distributed NextPlace by incrementing a future step K, instead of doubling it. However, this degrades the runtime performance. Our experiments have also indicated that the incrementation of the future step K does not yield a predictive performance which is worth for consideration.

4.6 Utilizing Semantic Names For Location Predictions

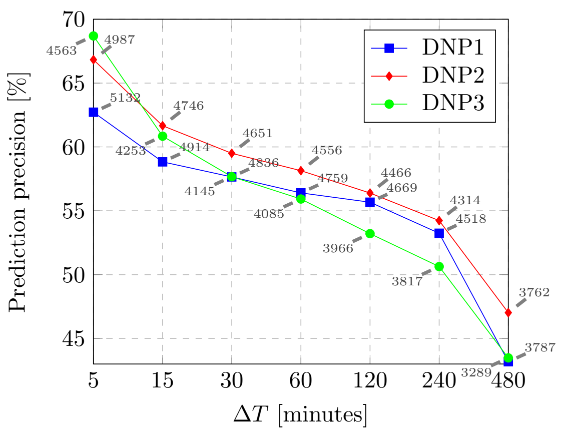
In this section, the results of the prediction precision are reported for a case where semantic location names are used in predictions. A semantic location or a point of interest (POI) is a particular point location that is useful or interesting for a user [Gale]. The term can refer to schools, distinctive buildings, shopping centers, movie theaters, tourist attractions or any other spots used in navigation systems [Gale].

In Cabspotting and CenceMe GPS datasets, GPS points do not have labelings which may indicate that they belong to some point of interests. In Ile Sans Fils dataset, significant locations have names, but they are anonymized and cannot be exploited for semantically enhanced predictions. In comparison, Dartmouth WiFi dataset has meaningful location names. Their structure is as follows: *AcadBldg*AP*, AdmBldg*AP*, AthlBldg*AP*, LibBldg*AP*, ResBldg*AP*, SocBldg*AP*, and OthBldg*AP** which means access points of the academic building, administrative building, library building, residential building, social building, and other building respectively. The sign * takes some numbers to indicate the number of the building and the number of the access point. Note that, the significant locations in WiFi-based datasets are WiFi access point locations. For each significant (access point) location, there is a corresponding visit history.

As an example, we consider locations *AcadBldg2AP1, AcadBldg2AP2, AcadBldg4AP3, AcadBldg4AP4, AcadBldg10AP2, AcadBldg10AP3* which have their own visit histories. Our semantic abstraction is simple. We perform two levels of abstraction and check how the prediction precision is affected. In the first level of abstraction, we consider access points of the same building as one significant location, i.e. *AcadBldg2, AcadBldg4, AcadBldg10* by merging corresponding visit histories together. In the second level, we take the access points of the building of the same type as a significant location, i.e. *AcadBldg*, again by merging the visit histories.



(a) Abstraction Level 1



(b) Abstraction Level 2

Figure 15: Prediction precision of Dartmouth WiFi dataset for different abstractions of semantic location names and for different embedding dimensions. Error margin is $\theta = 900$ seconds. Pins of the coordinates represent the number of users which can be predicted after ΔT interval.

For $\Delta T = 5$ minutes, the first level abstraction yields prediction precision of 50% in Fig. 15a and the second level abstraction yields 69% of prediction precision in Fig. 15b. Both levels provide better prediction quality than 40% of the original dataset in Fig. 11c. Even for $\Delta T = 480$ minutes, the prediction precision improves from 18% to 28% and 43% respectively. In both levels, we obtain a similar behavior as in the case where we decrease the number of significant locations (see Fig. 12). In fact, both abstractions decrease "average number of significant locations per user" of the Dartmouth WiFi dataset which makes nonlinear predictor pick a visit from a smaller set of predicted visits possibly occurring at $T+\Delta T$. Hence, the prediction quality improves. A prediction precision of the second level abstraction is higher than the first level. The reason is that the second level provides less number of significant locations per user. Note that, there are only 7 building types in Dartmouth WiFi dataset for the second abstraction level. We conclude that we may still expect lower prediction quality in the datasets with years of user trace where a lot of location types can be distinguished. Because, during multiple years and in different periods of user trace, the uniqueness of time-of-day information of visits to each significant location is more likely to diminish.

For GPS datasets without semantic location names, GPS points can be clustered to significant locations by matching each GPS point to a possible semantic location from GIS (Geographical Information System) data (e.g. such data can be obtained from OpenStreetMap [Haklay and Weber, 2008]). Visits to GPS points which belong to one semantic location can then be used to create a corresponding visit history. Visit histories of different semantic locations of GPS points can then be used for location prediction.

5 Related Work

Location prediction applications are based on the analysis of patterns in the human movement traces. Initial models on human movements are studied thoroughly in [Mishra et al., 2003, Balazinska and Castro, 2003] which analyze movement traces to understand the possible patterns in human movements. Other papers such as [Chaintreau et al., 2007, Eagle and Pentland, 2006] study patterns in human movements and network connectivity. They approximate intercontact time between mobile users as power-law distributions. They utilize these distributions to create more effective opportunistic forwarding algorithms [Jain et al., 2004]. In [Eagle and Pentland, 2006], temporal patterns in human movements are investigated in order to identify the time and daily activity relationships and significant locations of mobile users. Studies performed in these papers characterize statistical aspects of temporal patterns of user groups. However, we focus on the prediction of individual users in our nonlinear method.

Different location prediction methods (which do not predict arrival time and residence time of the next location) are evaluated in [Song et al., 2004]. Evaluation of the NextPlace with a custom dataset is presented in [Chon et al., 2012]. Prediction method relying on spatial and temporal aspects of collective movement trajectories is proposed by [Monreale et al., 2009]. This method predicts the next location of a mobile user by comparing his trajectory to a set of globally frequent trajectories [Scellato et al., 2011]. It is broader because it also identifies relationships between visits performed to different locations. In comparison, our nonlinear method focuses on the time-of-day aspect of human movements such that predictions are also generated for users who do not act out in a global behavior. Other location prediction methods are Markov based. For example, in [Ashbrook and Starner, 2003] a prediction method based on GPS traces of users is proposed. This method utilizes a first-order Markov predictor to estimate possible transitions between significant locations. However, it does not consider temporal patterns of user behavior during the prediction [Scellato et al., 2011]. In [Liao et al., 2006] the authors obtain significant locations by using a discriminative relational Markov network. They then use a generative dynamic Bayesian network to study possible transportation patterns. BreadCrumbs [Nicholson and Noble, 2008] is another framework which utilizes a second-order Markov model to predict future network connectivity. This framework estimates the next location of a user, but it cannot predict the future residence time and the transition time between two successive future visits. The method Predestination [Krumm and Horvitz, 2006] uses Markov models to predict the destinations (future visit locations) of drivers by analyzing partial trajectories of their vehicles. The original NextPlace paper [Scellato et al., 2011] evaluates the predictive performance of the Markov based methods in comparison to the serial NextPlace. Their empirical study reveals that Markov methods can generate accurate predictions only for the close future since their nature is memoryless.

6 Conclusion

In this thesis, we have proposed the Distributed NextPlace which is a novel method to predict future user locations in parallel by considering spatial, temporal and semantic aspects of location history data. It is based on nonlinear time series analysis of start times and residence times of visits of users to different significant locations. It predicts not only the next user location, but also future arrival time and residence time for that location. It improves the serial NextPlace [Scellato et al., 2011] in several aspects: it checks visits at time $T + \Delta T$ in a more robust manner by the help of date components, adds probabilistic nature to pick a visit from a set of visits which happen at a specific time, parallelizes the algorithm to enhance runtime performance, and utilizes semantic location names in predictions.

In our empirical evaluation, we compared the serial NextPlace and Distributed NextPlace in terms of predictive and runtime performance in four datasets. For some datasets, we obtained a slight improvement in the prediction quality because of the probabilistic nature of the Distributed NextPlace and speedup in the runtime which is up to 4 (where the perfect speedup is 8). On the other hand, we evaluated how different distance functions and train-test splits can affect the predictions generated by different visit histories. We also evaluated how the number of users in each dataset reacts to different train-test splits in terms of predictability error and neighborhood density of visit histories. Finally, we checked how the number of significant locations and semantic abstraction of the significant locations can affect location prediction accuracy.

For future work, a number of enhancements can be performed. Firstly, our empirical study demonstrated that a user can be predicted to be in different locations at the same time. In this case, a more robust probabilistic method can be pursued to find a "next place" among several promising location predictions. Secondly, weekly or monthly human rhythms can be investigated to check whether the prediction quality can be improved. Finally, the parallelization scheme can be enhanced; our parallelization scheme in the Distributed NextPlace algorithm is naive, but it still exhibits a reasonable level of speedup even for hundreds and thousands of users with thousands of visits. Speedup can be further improved by more sophisticated strategies. For example, instead of a user level parallelization, a visit history level parallelization can be utilized. In this case, different visit histories of each user can be predicted concurrently and predictions can be aggregated together to find a user location at the specific instant of time. If no location becomes available, either concurrent procedure can be repeated by doubling a future step K or the algorithm can end by stating that a user will not be in any significant location.

7 References

Example of cluster countours generated by matlab. URL https://it.mathworks.com/help/stats/fitgmdist.html. (visited on 2018-12-15).

Daniel Ashbrook and Thad Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous computing*, 7(5):275–286, 2003.

Magdalena Balazinska and Paul Castro. Characterizing mobility and network usage in a corporate wireless local-area network. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 303–316. ACM, 2003.

Jonathan Bentz. Parallel computing. URL http://mathworld.wolfram.com/ParallelComputing.html. (visited on 2019-01-21).

Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 6(6):606–620, 2007.

Chris Chatfield. *The analysis of time series: an introduction*. CRC press, 2016.

Yohan Chon, Hyojeong Shin, Elmurod Talipov, and Hojung Cha. Evaluating mobility models for temporal prediction with high-granularity mobility data. In *Pervasive computing and communications (PerCom), 2012 IEEE international conference on*, pages 206–212. IEEE, 2012.

Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

Gary Gale. Location vs. place vs. poi. URL https://www.vicchi.org/2010/11/16/location-vs-place-vs-poi/. (visited on 2019-01-25).

Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *nature*, 453(7196):779, 2008.

Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *Ieee Pervas Comput*, 7(4):12–18, 2008.

Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 187–201. ACM, 2004.

Sushant Jain, Kevin Fall, and Rabin Patra. *Routing in a delay tolerant network*, volume 34. ACM, 2004.

josliber. k-nn computational complexity. Cross Validated. URL https://stats.stackexchange.com/q/219664. (visited on 2018-12-05).

William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

Jong Hee Kang, William Welbourne, Benjamin Stewart, and Gaetano Borriello. Extracting places from traces of locations. In *Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 110–118. ACM, 2004.

Holger Kantz and Thomas Schreiber. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004.

Minkyong Kim, David Kotz, and Songkuk Kim. Extracting a mobility model from real user traces. In *INFOCOM*, volume 6, pages 1–13, 2006.

David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). Downloaded from https://crawdad.org/dartmouth/campus/20090909, September 2009.

John Krumm and Eric Horvitz. Predestination: Inferring destinations from partial trajectories. In *International Conference on Ubiquitous Computing*, pages 243–260. Springer, 2006.

Michael Lenczner, Benoit Grégoire, and François Proulx. CRAWDAD dataset ilesansfil/wifidog (v. 2007-08-27). Downloaded from https://crawdad.org/ilesansfil/wifidog/20070827, August 2007.

Lin Liao, Donald J Patterson, Dieter Fox, and Henry Kautz. Building personal maps from gps data. *Annals of the New York Academy of Sciences*, 1093(1):249–265, 2006.

Emiliano Miluzzo, Nicholas D Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B Eisenman, Xiao Zheng, and Andrew T Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

Arunesh Mishra, Minho Shin, and William Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *ACM SIGCOMM Computer Communication Review*, 33(2):93–102, 2003.

Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 637–646. ACM, 2009.

Mirco Musolesi, Kristof Fodor, Mattia Piraccini, Antonio Corradi, and Andrew Campbell. CRAWDAD dataset dartmouth/cenceme (v. 2008-08-13). Downloaded from https://crawdad.org/dartmouth/cenceme/20080813, August 2008.

Anthony J Nicholson and Brian D Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 46–57. ACM, 2008.

Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from https://crawdad.org/epfl/mobility/20090224, February 2009.

Douglas A. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, 2009.

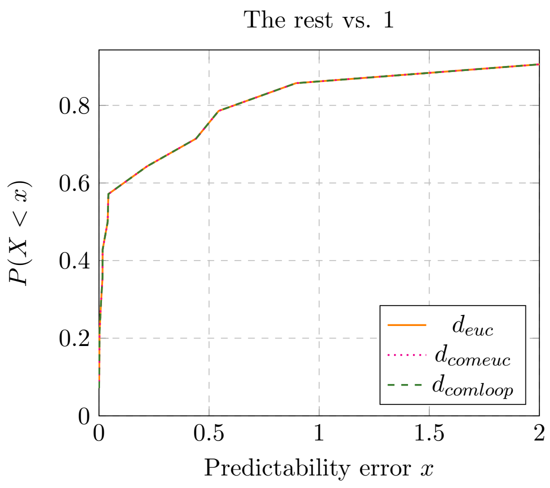
Salvatore Scellato, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Andrew T Campbell. Nextplace: a spatio-temporal prediction framework for pervasive systems. In *International Conference on Pervasive Computing*, pages 152–169. Springer, 2011.

Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive wi-fi mobility data. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1414–1424. IEEE, 2004.

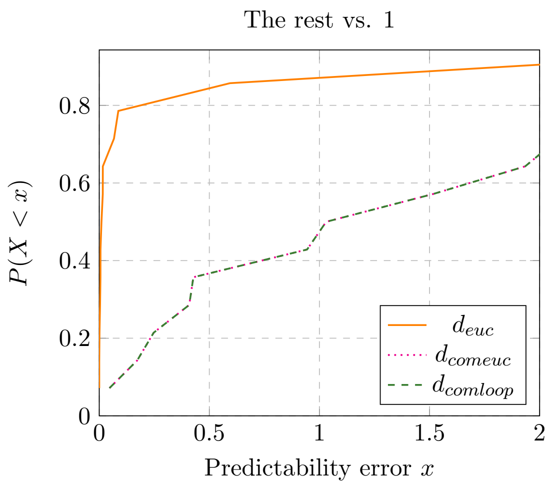
Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M Franklin, Scott Shenker, and Ion Stoica. Fast and interactive analytics over hadoop data with spark. *USENIX Login*, 37(4):45–51, 2012.

Changqing Zhou, Dan Frankowski, Pamela Ludford, Shashi Shekhar, and Loren Terveen. Discovering personally meaningful places: An interactive clustering approach. *ACM Transactions on Information Systems (TOIS)*, 25(3):12, 2007.

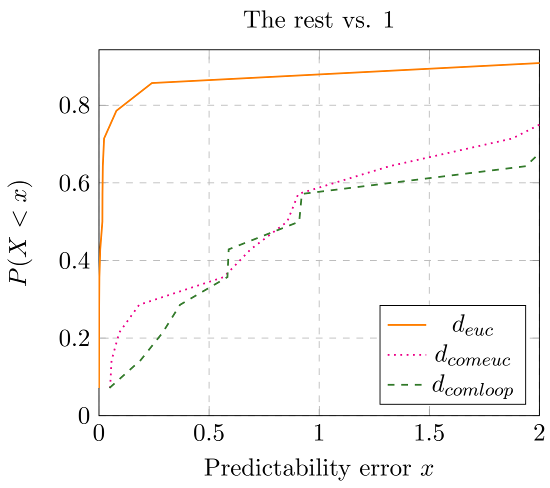
A Predictability Errors With Respect to Different Distance Functions



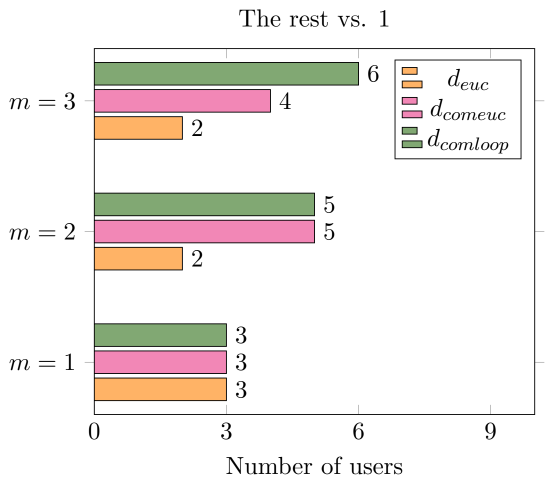
(a) Embedding dimension $m = 1$



(b) Embeddig dimension $m = 2$

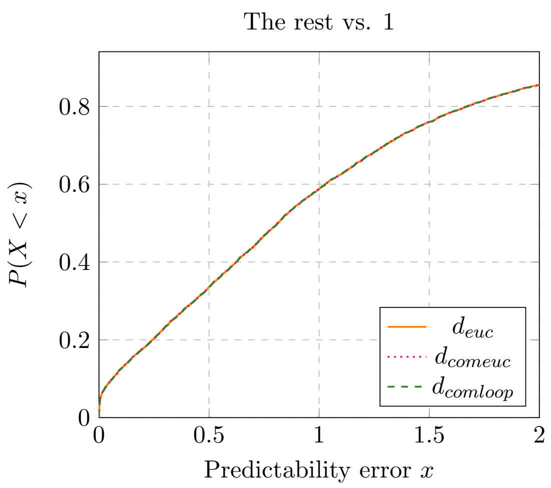


(c) Embedding dimension $m = 3$

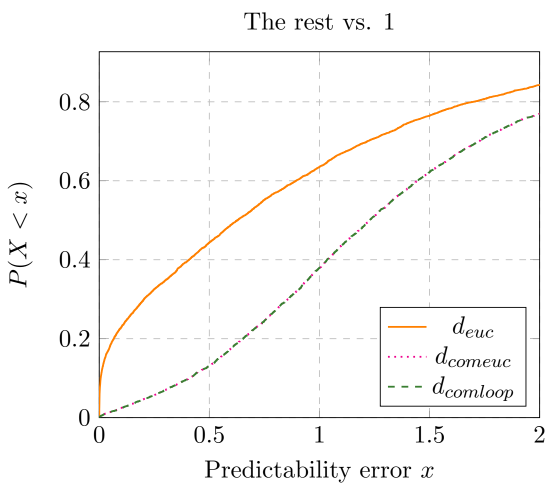


(d) $pe \geq 2.0$

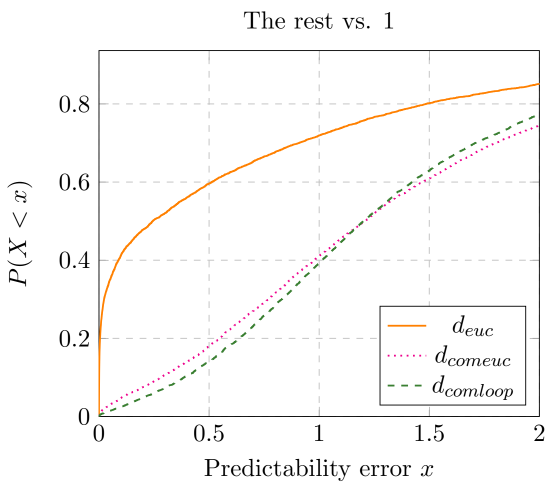
Figure 16: CDF of 1-step ahead predictability errors for CenceMe GPS dataset with respect to different embedding dimensions and different distance functions. 14 common users out of 17 are selected such that none of the users exhibits empty ϵ-neighborhood in all its visit histories. In (a), $d_{euc} = d_{comeuc} = d_{comloop}$. In (b) and (c), d_{euc} outperforms d_{comeuc} and $d_{comloop}$. In (d), we show how all 17 users react to different metrics. For example, when $m = 3$, d_{comeuc} and $d_{comloop}$ cause 6 and 4 users out of 17 to have $pe \geq 2.0$, respectively. This kind of routine also exists when $m = 1$ and $m = 2$. However, for d_{euc} almost stable number of users have $pe \geq 2$. Number of lost users ($|U_\epsilon| = 0$), is similar for all distance functions for $m = 1, 2, 3$.



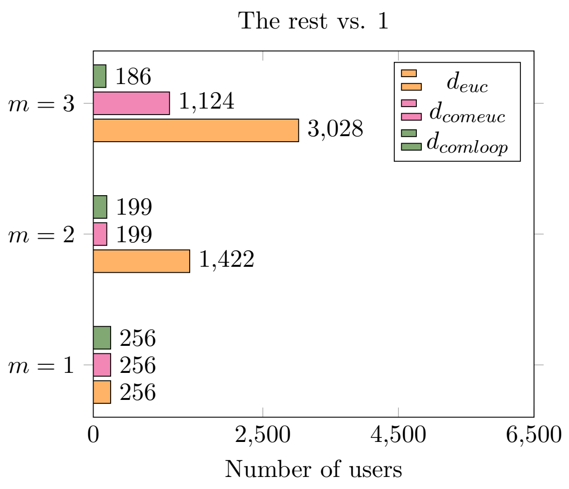
(a) Embedding dimension $m = 1$



(b) Embeddig dimension $m = 2$



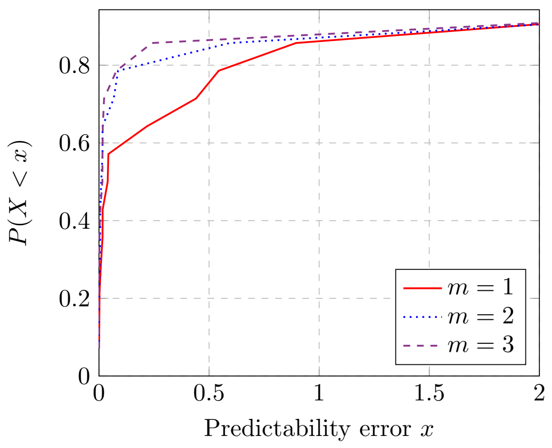
(c) Embedding dimension $m = 3$



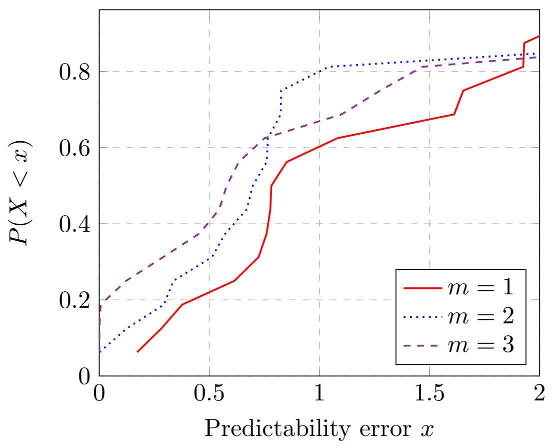
(d) All visit histories have $|U_\epsilon| = 0$

Figure 17: CDF of 1-step ahead predictability errors for Dartmouth WiFi dataset with respect to different embedding dimensions and different distance functions. Experiments are performed on 6003 common users out of 9041 users such that none of the users exhibits empty ϵ-neighborhood in all its visit histories which can make prediction impractical. In (a), when the embedding dimension $m = 1$, $d_{euc} = d_{comeuc} = d_{comloop}$. In (b), when $m = 2$, d_{euc} shows better performance and $d_{comeuc} = d_{comloop}$. In (c), d_{euc} still shows better performance, d_{comeuc} and $d_{comloop}$ show similar performance. In (d), we show how all 9041 users react to different metrics. For example, when $m = 3$, d_{euc} causes 3308 users out of 9041 to be lost such that they have empty ϵ-neighborhood in all visit histories. This kind routine also exists for d_{euc}, when $m = 1$ and $m = 2$. However, for d_{comeuc} and $d_{comloop}$, almost stable number of users are lost for $m = 1, 2, 3$. Number users whose $pe \geq 2.0$, is similar for all distance functions for $m = 1, 2, 3$.

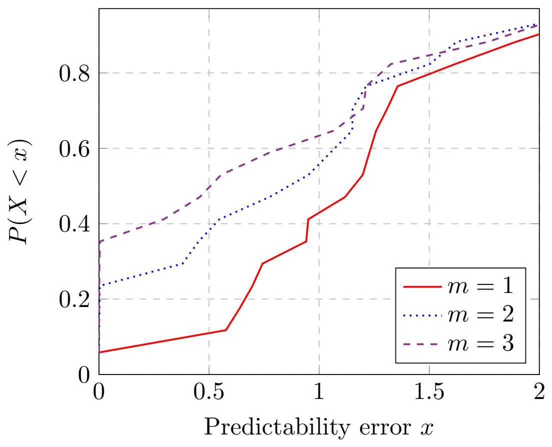
B Predictability Errors With Respect to Different Train and Test Splits



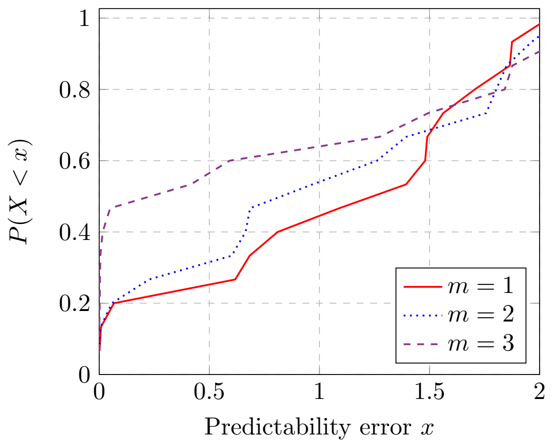
(a) The rest vs. 1



(b) 90% vs. 10%

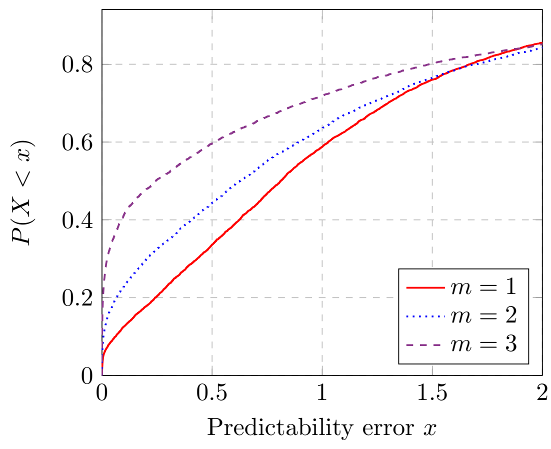


(c) 70% vs. 30%

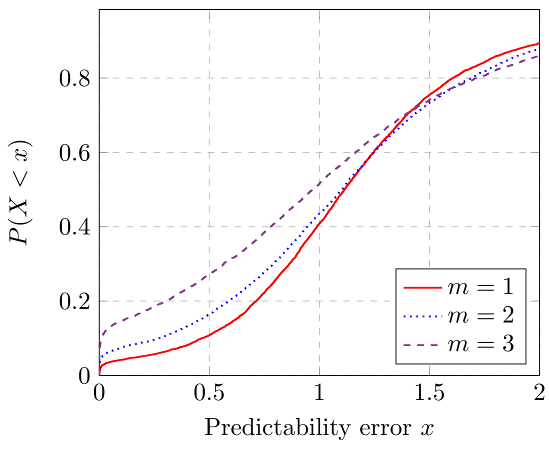


(d) 50% vs. 50%

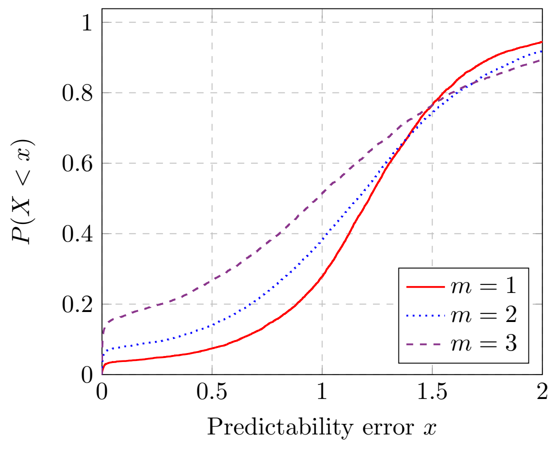
Figure 18: Cumulative Distribution Function of predictability errors for CenceMe GPS dataset with respect to different embedding dimensions and different train and test splits.



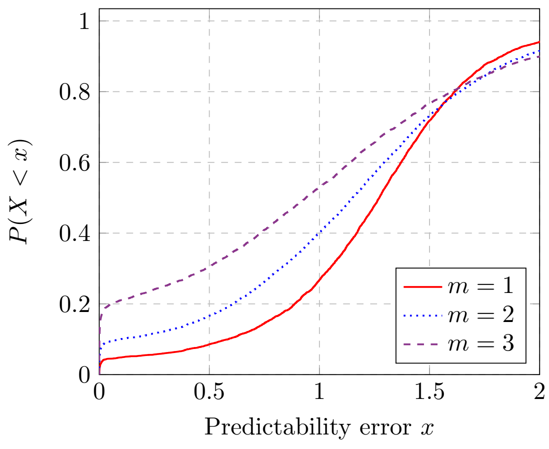
(a) The rest vs. 1



(b) 90% vs. 10%



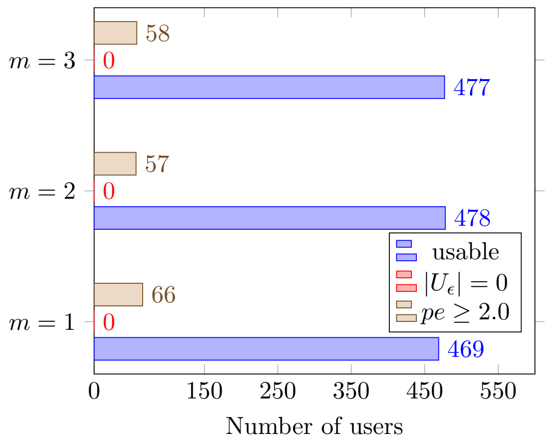
(c) 70% vs. 30%



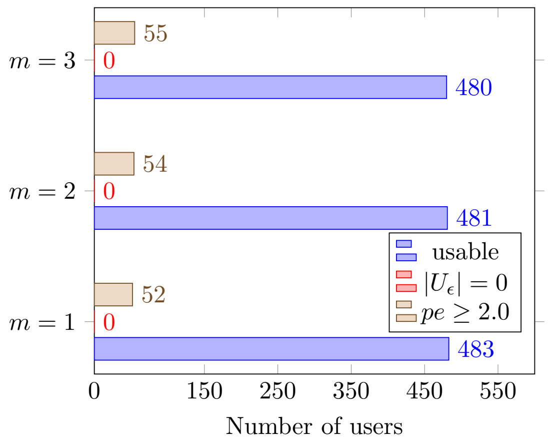
(d) 50% vs. 50%

Figure 19: Cumulative Distribution Function of predictability errors for Dartmouth WiFi dataset with respect to different train and test splits.

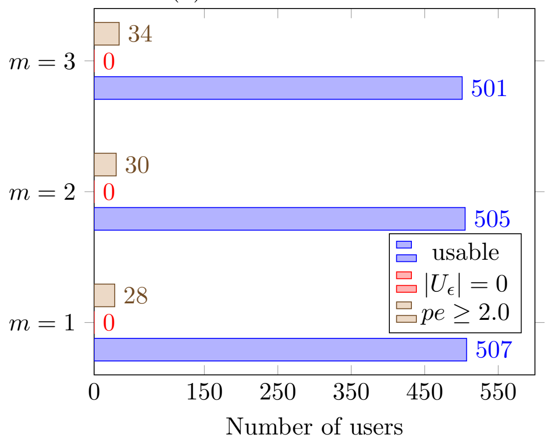
C Number of Users Affected by Different Train-test Splits and Embedding Dimensions



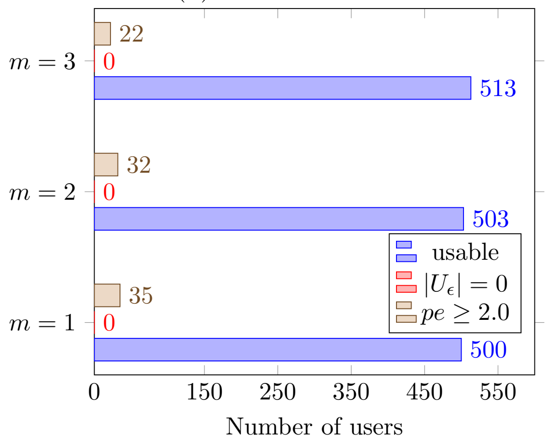
(a) The rest vs. 1



(b) 90% vs. 10%

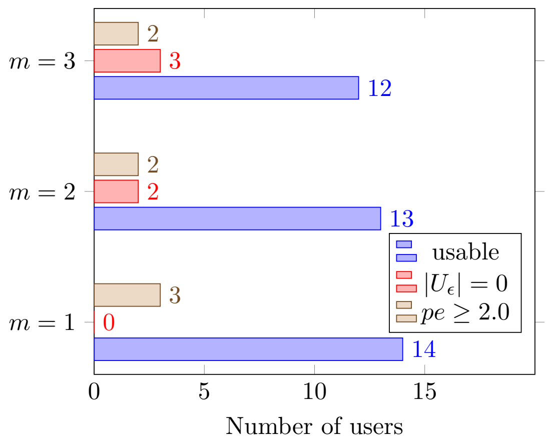


(c) 70% vs. 30%

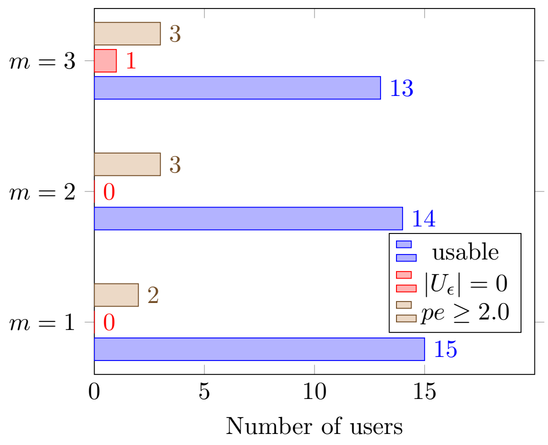


(d) 50% vs. 50%

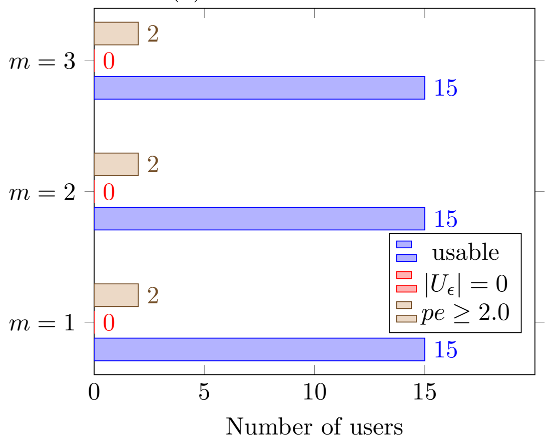
Figure 20: Number of users affected in Cabspotting dataset for different train and test splits. $|U_\epsilon| = 0$ represents the number of users which has empty neighborhood in all visit histories. $pe \geq 2.0$ represents the number of users with predictability error bigger than or equal to 2. If these quantities are subtracted from original number of users, then number of "usable" users are obtained.



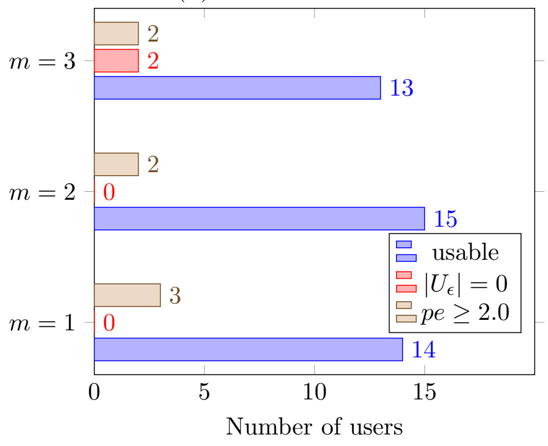
(a) The rest vs. 1



(b) 90% vs. 10%

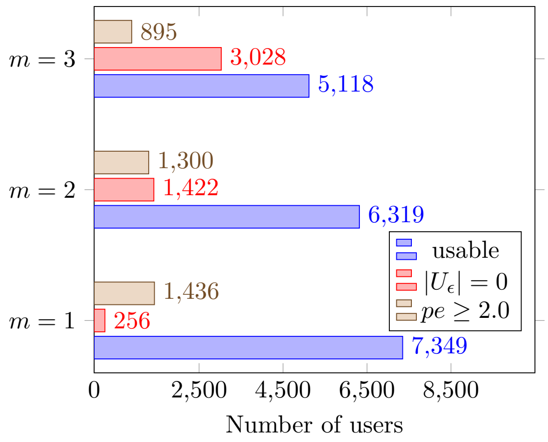


(c) 70% vs. 30%

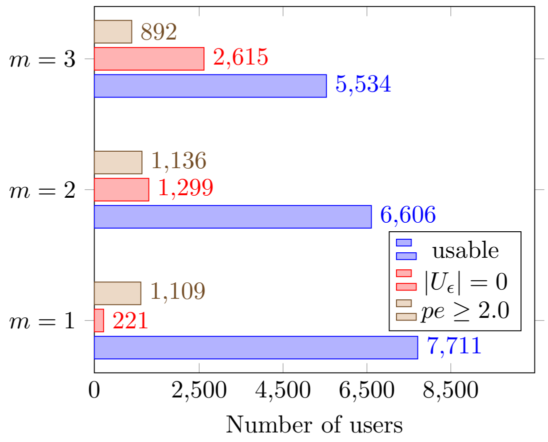


(d) 50% vs. 50%

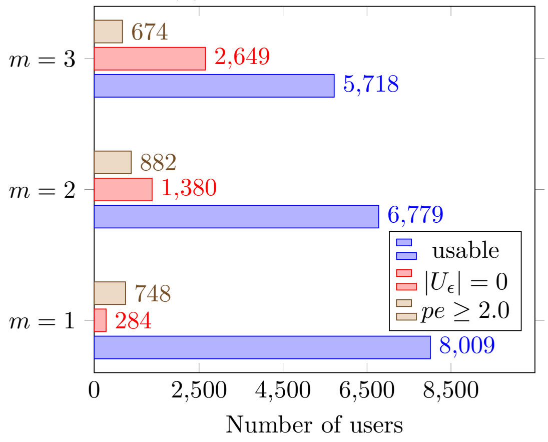
Figure 21: Number of users affected in CenceMe GPS dataset for different train and test splits. $|U_\epsilon| = 0$ represents the number of users which has empty neighborhood in all visit histories. $pe \geq 2.0$ represents the number of users with predictability error bigger than or equal to 2. If these quantities are subtracted from original number of users, then number of "usable" users are obtained.



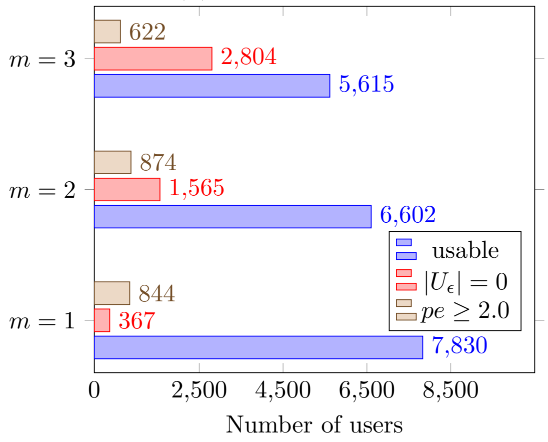
(a) The rest vs. 1



(b) 90% vs. 10%

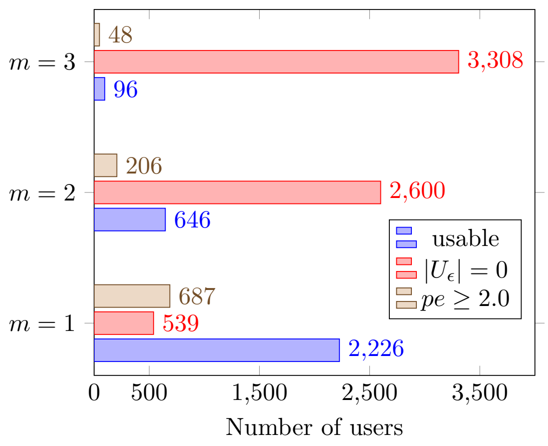


(c) 70% vs. 30%

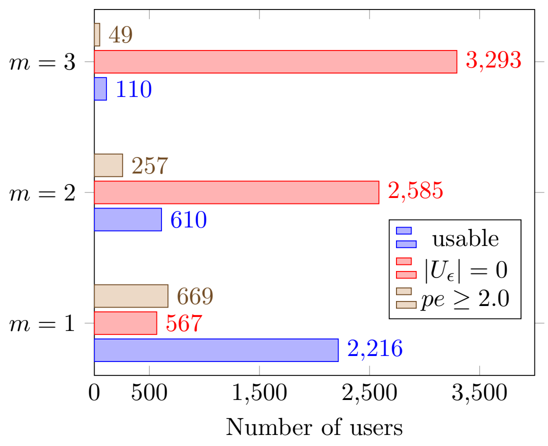


(d) 50% vs. 50%

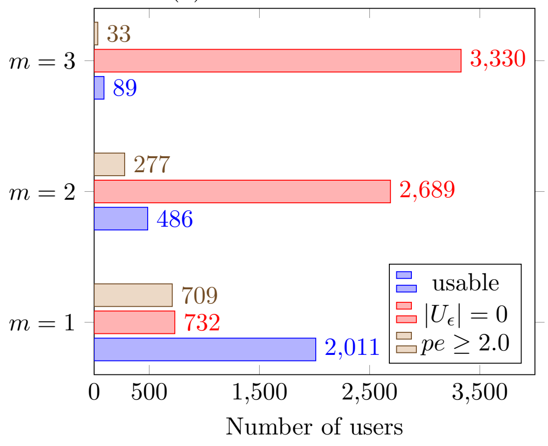
Figure 22: Number of users affected in Dartmouth WiFi dataset for different train and test splits. $|U_\epsilon| = 0$ represents the number of users which has empty neighborhood in all visit histories. $pe \geq 2.0$ represents the number of users with predictability error bigger than or equal to 2. If these quantities are subtracted from original number of users, then number of "usable" users are obtained.



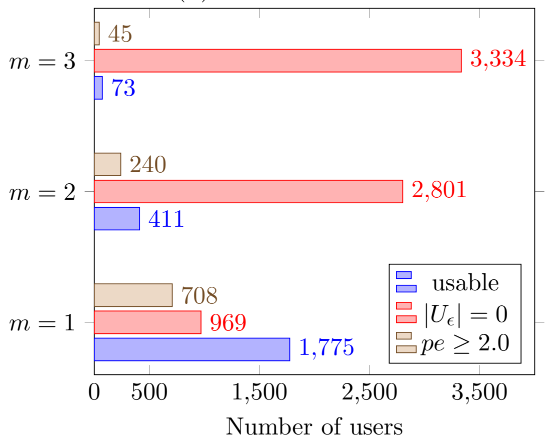
(a) The rest vs. 1



(b) 90% vs. 10%



(c) 70% vs. 30%



(d) 50% vs. 50%

Figure 23: Number of users affected in Ile Sans Fils dataset for different train and test splits. $|U_\epsilon| = 0$ represents the number of users which has empty neighborhood in all visit histories. $pe \geq 2.0$ represents the number of users with predictability error bigger than or equal to 2. If these quantities are subtracted from original number of users, then number of "usable" users are obtained.