Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Algorithms for time-space behaviour analysis of suspects

Bc. Ján Tkačík

Supervisor: Ing. Pavel Kordík, Ph.D.

12th January 2016

Acknowledgements

First and foremost, I would like to thank to Jana Čabaiová for extensive help with writing of this thesis as well as relentless support in critical times. My next thanks go to Jakub Žitný for consultations and TLV company for borrowing hardware for testing.

Last but not least I would like to thank my supervisor Ing. Pavel Kordík, Ph.D. for consultations and help during the whole study.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 12th January 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tkacik, Jan. *Algorithms for time-space behaviour analysis of suspects.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Mnoho studií prokázalo, že vzorce lidského pohybu mají vysoký stupeň jak prostorové, tak i časové pravidelnosti. Tento fakt nás ujišťuje, že je možné vytvořit model lidského chování pro extrakci vzorů chování a predikci pohybu. Detektivové vybaveni těmito modely budou schopní odhalit potenciální hrozby rychleji, stejně jako automaticky odhalit podezřelé chování sledovaných osob.

Implementovali jsme aplikaci umožňující detekci důležitých míst pro jednotlivce s následnou možností predikce pohybu mezi těmito místy jen z osobní historie polohy. Pro detekci důležitých míst byl navržen nový algoritmus AgarClust. Na predikci pohybu byl použit model založnený na rekurentní neuronové síti, Neural Turing Machine. Ukázali jsme, že prediktor založený na NTM je schopný modelovat mnohé vzorce pohybu s přesnosí blížící se maximální prediktabilitě. Vytvořená aplikace pomůže zefektivnit práci detektívů Policie České Republiky pri analýze dat o sledovaných osobách.

Klíčová slova Predikce mobility, Neural Turing Machine, Rekurentní neuronové sítě, Detekce zajímavých míst, DBSCAN, LDBSCAN

Abstract

Many studies have shown that human mobility patterns have a high degree of both spatial and temporal regularity. This fact assures us that it is possible to create a model of human behaviour for mobility pattern extraction and location prediction. Detectives empowered with such models will be able to detect potential threats faster as well as detect suspicious behaviour of suspects automatically.

We have implemented application for personally important place detection as well as mobility prediction just from personal location history. For place detection new algorithm AgarClust has been proposed. Sequence learning potential of recurrent neural networks has been used for mobility prediction. We implemented and used Neural Turing Machine to model person's behaviour and to predict his future locations. We showed that NTM predictor is able to model many mobility patterns with accuracy almost equal to maximum predictability. Created application will help detectives within Police of the Czech Republic to notice threats faster and make their work more efficient.

Keywords Mobility prediction, Neural Turing Machine, Recurrent neural networks, Important place detection, DBSCAN, LDBSCAN

Contents

List of Figures

List of Tables

Introduction

Motivation and objectives

There are not many applications on the market that can be used for personal behaviour analysis and we did not find any that provide ability to predict person‘s future locations automatically. Potential existence of such application can simplify and speed up work of detectives when trying to understand behaviour patterns of a suspect. We think creating such application is possible as movement of almost every person is regular to some degree, what means that this movement can be predicted. Gonzales in his work studied people movements, based on a sample of 100000 individuals over a six-month time period and showed that human mobility patterns have a high degree of both, spatial and temporal regularity [1]. From one‘s position history we can calculate degree of regularity and derive upper-bound for prediction accuracy [2]. There are also many studies showing that person mobility patterns are also centered around a small number of locations such as home, work, favourite bar or shop and so on. With this in mind we can imagine many applications of this prediction in various used-centered or crowd-centered applications. In our case we will try to use these predictions to model standard behaviour of suspects and extract places that are visited often or repeatedly as well as identify places which do not follow suspect‘s standard life cycle. These models will be used by detectives who will be able to understand suspect‘s behaviour patterns better and faster.

Problem definition

Our goal is to design, implement and test algorithms that are able to create easily understandable person‘s behaviour model from GPS and GSM-based location history. We need to be able to process data very sparse in time, inaccurate and noisy. Created software should also be as autonomous as possible

and should be able to predict person‘s future locations based only on location history.

Proposed solution

As we are interested in person‘s model that can be easily interpreted and understood by detectives we will first identify person‘s important places or points of interests as well as detect noise in data. Preprocessed locations will be used to create one‘s behaviour model. The model will be used for prediction as well as finding person‘s behaviour patterns.

Organisation of the thesis

The thesis is structured as follows: First a state-of-the-art methods for personally important place mining and sequence learning are summarized in Chapter 1 together with quick analysis of solutions currently available for similar purpose as ours. In the next chapter both place detection methods and mobility prediction methods are proposed. Their implementation details and proof of concepts are in Chapter 3. Chapter 4 contains experiments done, together with performance evaluations of all proposed methods.

Chapter 1

Background and analysis

1.1 State-of-the-art

In the next section we will look at related work in field of personally important place mining and human location prediction.

1.1.1 Personally important place mining

In mining important places there can be recognised two main families of methods. The first are geometry-based methods that use location data such as data from GPS, GSM or other sources as an input and produce coordinates, polygons or circles describing the significant places. On the other hand, fingerprint-based methods works with list of places where person was, but without no direct information about geographical location of these places. As we are going to work with people‘s location history in terms of GPS and GSM location we will now focus only on geometry-based methods.

1.1.1.1 Geometry-based methods

As mentioned before, geometry-based methods process data with exact location and time. This data may be obtained from various sources, which may significantly differ in accuracy. Quality of input data in terms of both spatial accuracy and time regularity is the most important factor when deciding what method to use for important places discovery.

One of the first works on important place extraction was Marmasse and Schmandt‘s system comMotion [3]. The system extracted indicated important place when the GPS signal was lost, because it should indicate that person is inside of a building. This approach cannot identify significant places that are outdoor (e.g. parks, stadiums). Another disadvantage of the method is that it is prone to produce false important places when signal is lost in a city as a result of street canyon effect. It also assumes that we do have information

about GPS signal strength and is unusable for data from other location data sources as it is dependant on properties of GPS satellite signals.

Another popular approach to the problem is to detect stay points from the data as a point or a region where the person stays for a specified amount of time. Various clustering methods are used on such stay points in order to detect important places. For example Ashbrook and Starner [4] used variant of well known k-means algorithm. The basic idea is to take a point and a radius. All points within the radius are marked and mean of the points is found. The mean becomes a new center point and the process is repeated until the mean stops changing. When it no longer moves, all points within its radius are marked as one cluster (important place) and they are removed from consideration in next iterations. The procedure repeats until there are no points left.

Density based clustering approaches are also very popular. DBSCAN was used by Ester, Kriegel and Xu in [5]. DBSCAN algorithm starts with an arbitrary point and retrieves all points within chosen region ϵ. If number of these points is sufficiently large with regard to other parameter which defines minimum points in one cluster, a cluster is started. All unvisited cluster point‘s ϵ-neighbours are then checked for number of points until the whole dense cluster is found. If number of points in ϵ-neighbourhood is smaller than minimum points in cluster the point is marked as noise. The main advantage of this approach is noise detection. On the other hand it is very sensitive to chosen parameter values and it can be unusable for larger data sets. However, there are many variations of DBSCAN which improves its worst case complexity.

Another variant of density based clustering algorithm used for spatial data clustering is DJ-Cluster, method proposed by Zhou [6]. This method is one of the attempts to overcome performance problems of DBSCAN. Basic idea of DJ-Cluster is similar to DBSCAN. Neighbourhood is calculated for each point as points within distance ϵ, under the condition that there are more points than *minpoints* parameter. If no such neighbourhood is found, the point is labeled as noise. Otherwise the points are created as a new cluster or joined with an existing cluster if any of neighbours is in existing cluster.

A bit different view of the problem was introduced recently by Pavan and his collaborators in [7]. For each important place they define three basic features as mapping from stay points. Area of important place is value which indicates the diagonal extension of the rectangular region that spans over all points from which stay point was created. Intensity is value which describes how many times person‘s position has been detected inside position area. Finally the feature frequency indicates how many times the person came back for another visit in that location. These three features are used to find the most important places for person.

All of the approaches using stay points clustering make one important hidden assumption about the input data. Stay points are series of measured points when person‘s position does not change within some region for a spe-

cified amount of time. In other words stay points are points when person‘s speed is very small or zero in perfect case. To calculate person‘s speed with reasonable accuracy we need input data that are accurate and the measurements need to be taken very frequently. If we cannot make these assumptions about the input data, previous approaches could fail.

1.1.2 Person movement prediction

There are many previous works that have discussed the problem of predicting personal mobility. We will focus on the personalized methods that look on every person independently.

Most of these works use Hidden Markov Models or some of its derivation for prediction. Work [8] shows that sequences of important places for person are Markovian once the data is clustered by the day of the week and time of the day. They used Hidden Markov Models for prediction in a way that the time has been observed dimension and for every important place there have been state defined, with one extra state defined as unknown position. Prediction is done by use of Viterbi‘s algorithm to define most likely sequence of states.

Similar approach have been taken by Yang, Zheng and Chen in [9]. They introduce variable order Markov Model which should be accurate as N-order Markov Model but without performance issues related with it.

These models assume that person does not change his movement patterns over time. However, it is clear that this assumption is not valid as person patterns vary over time significantly. People change their jobs, relationship statuses or even housings or just change patterns temporarily, for example for holidays or when sick. To address this issue usage of sliding window for learning was proposed in [10].

1.2 Clustering methods for place detection

In the next section we describe known clustering methods that will be implemented for important place detection in our system.

1.2.1 DBSCAN

DBSCAN is well known density based clustering algorithm proposed by Ester in [5]. This algorithm was designed to cluster data of arbitrary shapes in the presence of noise. Main idea of DBSCAN is that for each object in the cluster the neighborhood of given radius has to contain at least some given number of objects. The only two parameters of this algorithm are then radius ϵ and minimum number of points in neighborhood $MinPts$.

The complexity of DBSCAN is $O(n^2)$ if we do not use any accelerating index structure that is able to accelerate query on all points within radius. With this optimization we can achieve runtime complexity of $O(nlogn)$.

1.2.2 LDBSCAN

LDBSCAN is density-based clustering algorithm proposed in [11]. It is similar to DBSCAN but is able to cope with uneven density distribution in data space. We do not set overall threshold density for the whole data space. This threshold is in LDBSCAN dynamic with regards to local density. It takes advantage of the local outlier factor metrics proposed in [12] for noise detection.

1.3 Recurrent neural networks for sequence learning

Recurrent neural networks are able to model sequences very accurately and are able to learn and carry out very difficult data transformations over extended periods of time as they have ability to remember and process past information. Moreover, it is known that recurrent neural networks are Turing-Complete [13]. Despite these facts recurrent neural networks failed to become commonly used tool for sequence learning tasks due to the problems with efficient training. Recent studies show that with correctly defined models and carefully chosen optimization methods it is not impossible.

In [14] Alex Graves successfully used recurrent neural networks for both prediction of discrete sequential data (text prediction) and prediction of real-valued sequential data (handwriting prediction). For text prediction he used LSTM [15] trained with back-propagation through time algorithm.

Graves, Wayne and Danihelka later introduced model called Neural Turing Machine in [16]. Neural Turing Machine is recurrent neural network with architecture inspired by both Turing Machine and Von Neumann computer architecture. All of its parts are differentiable so it can be efficiently trained with gradient descent. First results show that Neural Turing Machine is able to infer simple algorithms such as copying, sorting or associative recall just from input and output examples.

Neural Turing Machine contains two basic components: controller and memory. Figure 1.1 shows how the controller is connected to memory. Input to the controller is external input from user and data read from memory from previous step. In every update cycle controller transforms input to external output for user and updates the memory.

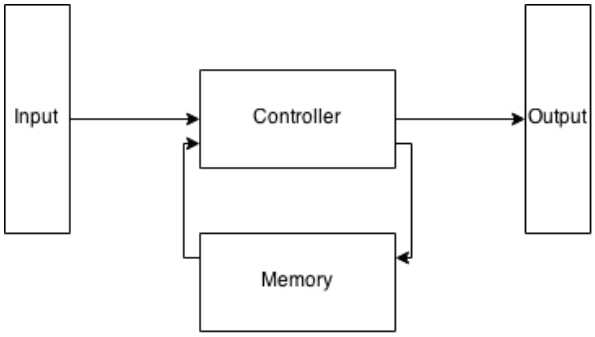


Figure 1.1: **Neural Turing Machine high level architecture.** During every update cycle the controller transforms data from input and data read from memory to output and memory update which is applied to memory afterwards.

The ability to remember data and to have dynamic state is the main attribute of recurrent neural networks. In Neural Turing Machine this ability is implemented with memory matrix with selective read and write operations. Basic overview of this implementation is shown on figure 1.2. Memory bank is matrix $N \times M$ where N is number of memory cells and M is size of cell, therefore memory cell is a vector. Every head is only an encapsulation of one part of controller output. For every head there are three vectors defined: add vector of size M, erase vector of size M and addressing vector. During every update cycle controller produces output which is interpreted as setting of one or more heads. For every head, addressing vector is calculated from head settings, addressing vector in last step and memory content. Addressing vector defines which memory cells will be read and updated. What will be read is defined only by memory content and what will be written to memory is defined by erase and add vector. Notice that every part of the Neural Turing Machine including addressing mechanism have to be differentiable. For more detailed description of Neural Turing Machine refer to [16].

1.4 Analysis of current solutions

There are not many applications on the market that can be used for personal behaviour analysis and we did not find any that provide ability to predict person‘s future locations automatically.

GeoTime5 [17] is one of the few in this area. It is great at integrating location data about a person from different sources like GPS logs, GSM cell records or even social networks. The main drawback of this solution is that it does not provide any automatic behaviour analysis, just extensive visualizations of data. Invalid (noise) point detection is done manually based just on map visualizations what can be overwhelming with large low quality data

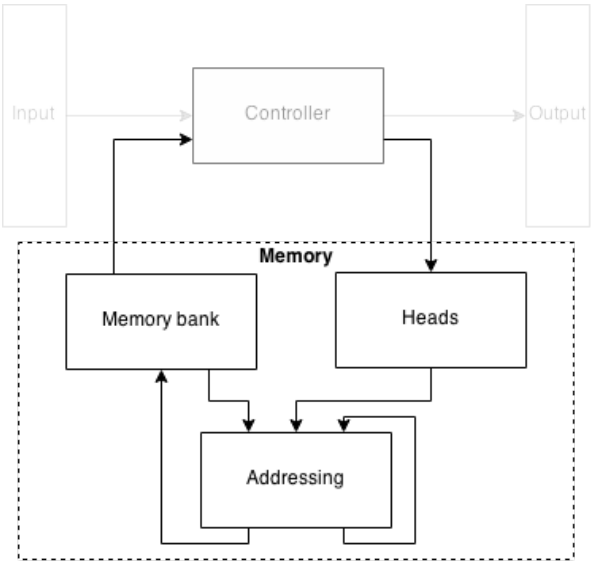


Figure 1.2: **Neural Turing Machine memory architecture.** During every update cycle the controller produces output which is interpreted as setting of one or more heads. For every head addressing vector is calculated from head settings, addressing vector in last step and memory content. This addressing vector defines which memory cells will be read and updated. What will be read is defined only by memory content and what will be written to memory is defined by erase and add vector.

sets. One of the automated features is so called detection of stationary events. Besides that, it provides many options of data visualization and reporting.

Another similar application is TrackerPAL [18] from TrackGroup company. It has very similar features like GeoTime5 but it works only with their proprietary tracking devices. This fact makes this solution unusable for our case.

For location data visualization almost any GIS software could be used. One for them is QGis [19]. It is great open source software for visualizations on map and for creating custom maps but it lacks any feature for automated behaviour analysis.

Chapter 2

Proposed solution

The main goal of this work is to implement system for human mobility prediction using sparse data about person‘s location with variable accuracy. Created model should be easily interpretable and as autonomous as possible. As shown in previous chapter, there is no application on the market that could achieve this goal currently. Our approach to this problem is to detect important places for a person and model standard person‘s movement between them. The model is then used for mobility prediction. Every feature will be available through simple user interface with comprehensive visualization of results.

2.1 Important places detection

If we do not want to predict exact locations in terms of latitude and longitude, we need to divide all possible locations to places with some meaning for the person. It is not an easy task to exactly define what is an important place and what is just movement between the places or noise.

Firstly we need to define what is a place. Place can be defined as area with exactly defined boundaries. For every location we are able to define if it belongs to the place. Area of these places can vary and large places can usually be divided into smaller places. Example of the large place can be a country. The country can be divided to districts, districts to departments, departments to cities and so on up to really small places like one house that can be split even more to rooms. The lower bound of precision we can use is given by accuracy of input data spatial component.

From all places visited by a person we need to choose only important ones. Place importance can be derived from the time component of the data. For each place we can tell approximate total stay time but also other features like visit frequency and regularity. Example of places visited regularly and frequently can be work, school or home.

All points that do not belong to any important place are marked as noise points. These points are typically movements between important places. For

the prediction purposes we will try to minimize count of these points as these points in discrete prediction means that we do not really know where the monitored person is.

In following sections we will describe algorithms implemented for important places detection in our system. The main requirements for such algorithms are:

- Ability to detect places of arbitrary shapes
- Noise detection capability
- Automatic detection of places count

We can notice two main groups of these algorithms which differ in input data. The first group takes only set of geographic positions with timestamp. The second group needs for each position also stay time in the position.

2.1.1 Definitions

In the next section we will define some basic concepts used in important place detection algorithms. Location is geographic position defined by latitude and longitude. Point is location with timestamp. It is a triplet containing latitude, longitude and timestamp. Point with stay time is a quadruplet which contains latitude, longitude, timestamp and stay duration in the point. Place is a set of points.

Equirectangular distance has been used for distance calculation between two points. We can see from algorithm 1 that it is just Pythagoras' theorem used on equirectangular projection. We chose this method because it is much faster than other methods like great-circle distance or spherical law of cosines with acceptable decrease of accuracy.

Algorithm 1: Equirectangular distance calculation

input : Two geographic locations a and b
output: Distance between a and b

1 **function** `EquirectangularDistance(`*Location a, Location b*`)`
2 $R \leftarrow$ Earth radius
3 $x \leftarrow (a.lon - b.lon) * \cos(\frac{a.lat - b.lat}{2})$
4 $y \leftarrow (a.lat - b.lat)$
5 **return** $\sqrt{x^2 + y^2} * R$

2.1.2 Weighted DBSCAN

We slightly modified DBSCAN algorithm for important place detection. The main difference is that there is point's weight defined on the input as stay

time in the point. The consequence is that in weighted DBSCAN we are counting total stay time of all points in given radius instead of total number of points. $MinPts$ parameter then changed to $MinTime$. Unlike original DBSCAN algorithm this version should be able to deal with data gathered with unstable sampling frequency. The other advantage of this method over standard DBSCAN is that parameter $MinTime$ can be interpreted easier as minimal time spent in some area needed for marking that area as a place.

2.1.3 AgarClust

Apart from DBSCAN and WDBSCAN we defined our own method for important place detection. AgarClust is iterative method meeting the requirements defined in previous section. The algorithm is based on moving and merging of Agar places. Agar place is set of points from input with one well defined position and weight. Position of a place can be anywhere, not necessarily on position of some input point. Weight of the place is simply the sum of approximate stay times of all points in place. The first step is to define Agar place for every input point with position defined by the point. Then iterations take place. One iteration of AgarClust consists of two steps. The first step is place merging. For every place we find all other places with distance from reference smaller than merge radius. New merged place has position of place with the largest weight and contains all points of merged places. The second step is place movement. For every place we calculate the force vector which is equal to the sum of all interactions between places. Every two places with distance smaller than max field range parameter interact. The magnitude of interaction force vector is equal to

$$F(p_1, p_2) = \frac{wp_1 * wp_2}{d(p_1, p_2)^2} \tag{2.1}$$

where wp is weight of a place and the force direction is simply heading from one place to the other. Every two places are attracted to each other. When all of the forces are calculated we can apply them to places. Every place changes its position according to formula

$$\Delta L = \frac{F * \epsilon}{wp^3} \tag{2.2}$$

where ΔL is place location update vector. These steps are repeated until there is no interaction between places. Resulting places are then filtered according to their weight. All places with weight smaller than noise threshold are marked as noise. Remaining places are important places detected. Main loop is shown in algorithm 2 and inner methods are shown in algorithm 3.

2.1.4 Hierarchical AgarClust

Hierarchical AgarClust is simple extension of AgarClust algorithm. It was designed for finding places with different areas. It only runs AgarClust clustering

with different settings of max field range parameter. In first step it runs AgarClust with large max field range parameter to find clusters with large areas. Then max field range is set to half of its previous value and the AgarClust is applied recursively on positions of every place found in previous step. This should find places with smaller area within the larger places. If no place is found place from previous step is used. If there is one or more place founded we apply AgarClust again on every place with halved max field range parameter. When max field range is smaller than minimum field range set the algorithm ends.

Algorithm 2: AgarClust main loop

input : Set of points D, Merge radius ϵ, Max field range $range$, Noise threshold κ
output: Places found

```
1 function AgarClust(Set of points D, Distance ϵ, Distance range,
  TimeSpan κ)
2     agarPlaces ← Create new agar place for each point in D on its
      position
3     repeat
4         forces ← GetForces(agarPlaces, range)
5         pointsMoved ← ApplyForces(agarPlaces, forces)
6         agarPlaces ← MergePlaces(agarPlaces)
7     until pointsMoved
8     return agarPlaces
```

Algorithm 3: AgarClust inner methods

```
1  function GetForces(List of agar places agarPlaces, Distance range)
2      forces ← empty list
3      foreach p in agarPlaces do
4          F ← 0
5          foreach p′ in agarPlaces do
6              d ← Distance(p, p′)
7              if p <> p′ AND d ≤ range then
8                  d ← Heading(p, p′)
9                  m ← (p.Weight ∗ p′.Weight) / d²
10                 Add new Force with magnitude m in direction d to F
11         Add F to forces
12     return forces
13 function ApplyForces(List of agar places agarPlaces, List of forces
   forces)
14     pointsMoved ← FALSE
15     foreach p in agarPlaces do
16         F ← forces on p 's index
17         δ ← (F.Magnitude ∗ STEPSIZE) / p.Weight³
18         if δ > 0 then
19             pointsMoved ← TRUE
20             Move p by δ in direction of force F
21     return pointsMoved
22 function MergePlaces(List of agar places agarPlaces, Distance ϵ)
23     mergedPlaces ← empty list
24     foreach p in agarPlaces do
25         mp ← empty list
26         foreach p′ in agarPlaces do
27             if Distance(p, p′ ≤ ϵ) then
28                 Add p′ to mp
29                 Remove p′ from agarPlaces
30         Add new place to mergedPlaces with position equal to position
           of place with max weight from mp and all points from all places
           in mp
31     return mergedPlaces
```

2.2 Mobility prediction

The only input to mobility prediction is person's location history. Firstly we extract important places from the history and then we use preprocessed data for prediction. The input to mobility prediction methods is then collection of places in time. The whole process is shown on figure 2.1. We have implemented two methods for mobility prediction. The first fast, based on K nearest neighbours and the second using Neural Turing Machine.

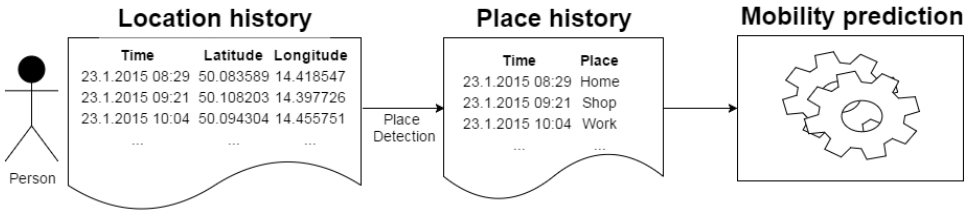


Figure 2.1: Mobility prediction process

2.2.1 K nearest neighbours predictor

K nearest neighbours predicts the person's position in future time just by finding similar times in history and using their places as prediction. The main challenge here is to define what are similar times in history.

Defining distance between two times as the absolute difference between them is not very suitable in our case as such k-NN predictor would always predict that person stays in his last known place or few last known places depending on k chosen.

We need to take advantage of the fact that behaviour of almost everyone is cyclic. The main cycle is daily cycle. Almost every person sleeps at night, gets up in the morning and has his own standard daily routine. This daily routine is often part of the weekly cycle and so on. Some people can have their own cycles or even live differently every day, but it is not common. Because of that we define separate features for time in day, day in week and absolute time. Moreover, to maintain time distances over midnight we map time in a day to two dimensional vector according to formulas:

$$x = \sin(2\pi * portionOfDay) \tag{2.3}$$

$$y = \cos(2\pi * portionOfDay) \tag{2.4}$$

As on the clock the distance between 23:00 and 01:00 is same as between 21:00 and 23:00. Using similar principle we encode day in a week so that distance between every consecutive pair of days is the same. Absolute time

is normalized to interval from minus one to one, so that the oldest historical position has absolute time minus one and newest has one.

The distance formula used for comparing times is defined as following.

$$d\alpha(p, p') = \alpha\sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2} \tag{2.5}$$

$$d\beta(p, p') = \beta\sqrt{(p.v - p'.v)^2 + (p.w - p'.w)^2} \tag{2.6}$$

$$d\gamma(p, p') = \gamma\sqrt{(p.z - p'.z)^2} \tag{2.7}$$

$$d(p, p') = d\alpha + d\beta + d\gamma \tag{2.8}$$

There are three parameters, time in day importance α, day in week importance β and absolute time importance γ. Optimizing these parameters together with number of neighbors k for every person independently radically improves performance of this method. In our implementation we used genetic algorithm to learn the best parameter settings.

2.2.2 Neural Turing Machine predictor

Neural Turing Machine predictor is based on Neural Turing Machine [16] trained to predict one‘s position one step after the last position presented to the network‘s input. This implies that we need to set quantization step in which historical data will be presented to the network to predict next positions. The network is trained with back-propagation through time algorithm with rmsprop weight optimizer. Neural Turing Machine itself has many number of parameters that significantly change its performance. The parameters are: size and type of controller network, size of memory and number of heads used for memory access. The only parameter that is fixed is type of controller network. Our implementation of neural turing machine uses simple feed-forward neural network with one hidden layer. This simple controller has been used as more complicated controller networks do not bring significant performance improvement according to results achieved in [16]. Number of neurons in that layer as well as other parameters can be set freely.

2.2.3 Input preprocessing

In order to match the prediction with time we need to re-sample the input sequence to have one predefined sample rate. If we choose sample rate to be 1 hour we will present 24 samples per day to the network and the output place predicted will be 1 hour after the actual input. In order to produce trustworthy predictions we need to choose this sample rate wisely with regard to original input data sample rate. For the re-sampling we use best known current place method as shown on the figure 2.3.

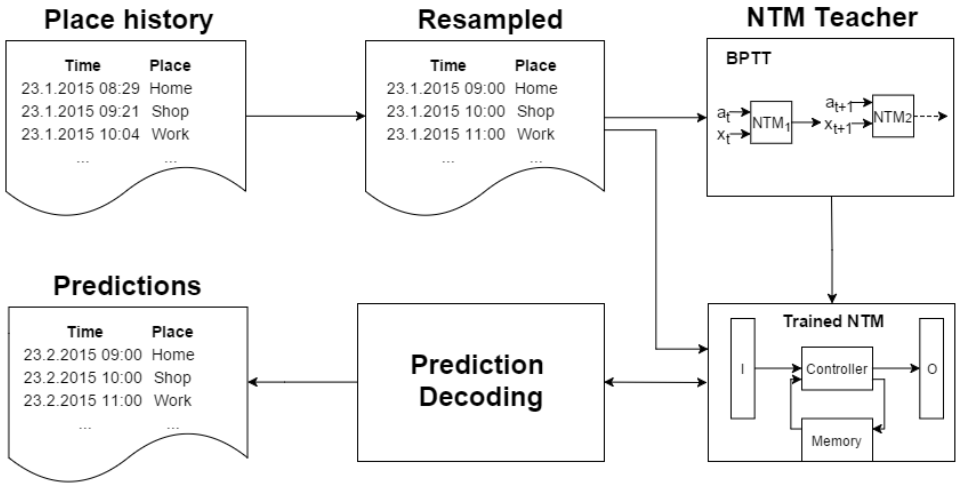


Figure 2.2: Neural Turing Machine predictor working schema

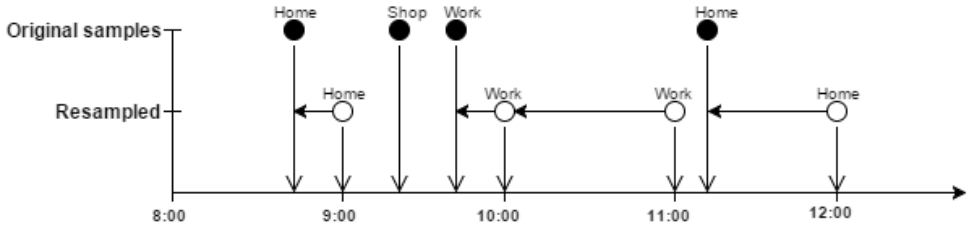


Figure 2.3: **Best known current place re-sampling method.** We can see that if we choose new sample rate higher than original we are speculating about points in between original samples (Work at 11:00). On the other hand, if we choose sample rate lower than original, we lose valuable information (Shop between 9:00 and 10:00).

We can see that if we choose new sample rate higher than original we are speculating about points in between original samples which can lead to untrustworthy predictions. On the other hand, if we choose sample rate lower than original, we lose valuable information.

Once again, it is very important to choose the new sample rate with regard to not only the original sample rate but also with other specific details of input data for example how the data has been gathered. If we used GPS monitoring unit which produces output only if the person is moving we would be able to choose higher sample rate even if there were large gaps between samples. In this case it would mean that person did not change his position and therefore the last best known position was valid.

On the input we have two pieces of information. Time and position of the person in terms of place in that time. Good encoding of the input is key to

the correct network operation. Time is split to two parts, time in a day and day in a week. The encoding is similar to the one used in KNN predictor. For the time we therefore have 4 input neurons. The place is encoded as "one-hot" input vector. That is, in our case that if there are N different places recognised for the person, and place n is on the input, then n^{th} element of input vector is set to one and all the other to minus one. To sum up, the input for one step is vector of length $4 + N$ where N is number of places that person visited in the available history. Example of the encoding is shown on figure 2.4.

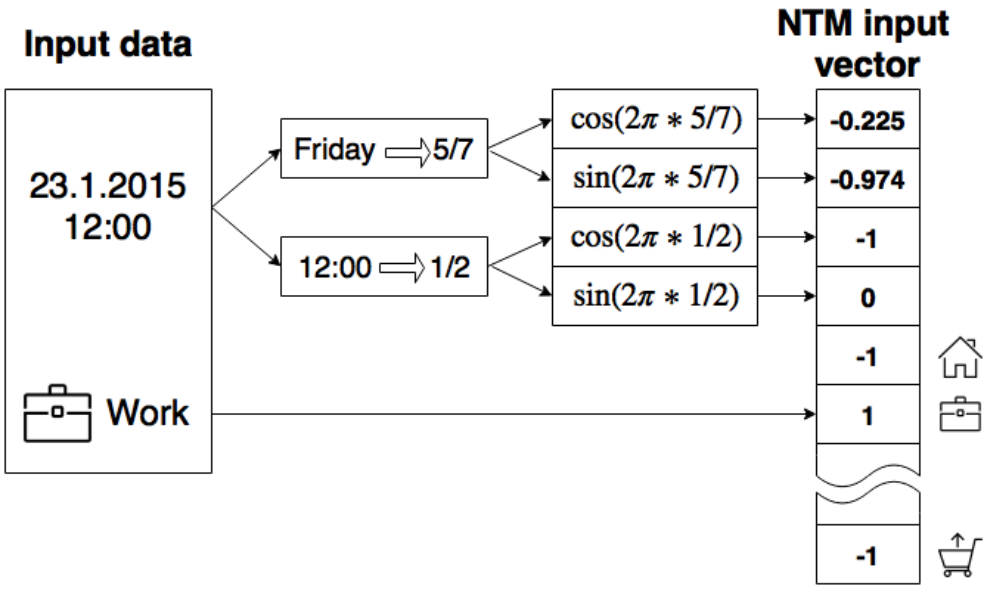


Figure 2.4: Neural Turing Machine predictor input vector encoding example

2.2.4 Learning

As the Neural Turing Machine is differentiable end to end, it can be efficiently trained by gradient descent. For training we are using method called back-propagation through time which is generalization of back-propagation method used for training simple feed-forward networks. As a weight optimization method rmsprop is used. Rmsprop is form of stochastic gradient descent where the gradients are divided by a running average of their recent magnitude. For update equations refer to [14]. The length of sequences presented to the network in one back-propagation through time training iteration can be set freely. The sequence length largely affects training speed and memory consumption. If one uses sequences with length 100 there will be 100 copies of the network in the memory what can be very limiting.

Data available for network learning are split into training and validation part. Validation part is taken from the end of the sequence and its length is

equal to length selected for training sequences. After every n iterations we try to predict the validation part and save the result. If there is no improvement on validation part for m iterations, the training is stopped and the machine with smallest validation error is used for later predictions. This is done to prevent network over-fitting.

Training recurrent nets can be very difficult without proper weights initialization at the start of learning. Weight initialization can significantly affect speed of convergence. Without good weight initialization we might also end up with learning nothing at all. Practically it means that two runs of training on the same data with the same network settings can end up with significantly different results. To face this problem we create more random initializations on the start of training and after every n iterations we discard one with the worst validation error till we end up with the best initialization found. This technique decreases variability of learned machines on the end so we end up with much more stable results.

The last training improvement is preferring to use newer sequences for learning. The sequence start in the training part of the input data is not taken from uniform random distribution but from beta distribution with parameters $\alpha = 1.1$ and $\beta = 1.0$. This setting slightly prefers newer sequences which helps when dealing with person's behaviour changes in the end of training sequence which leads to better adaptability. See figure 2.5 for comparison of beta distribution used for selecting learning sequence start with more commonly used uniform distribution.

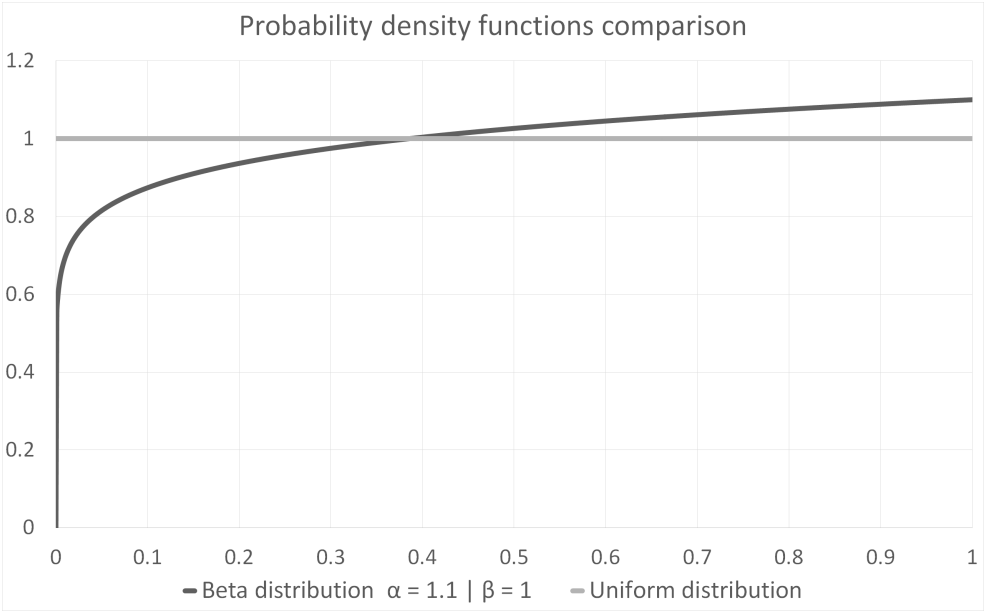


Figure 2.5: Comparison of beta and uniform distribution

2.2.5 Prediction decoding

Neural turing machine used for prediction has number of output neurons equal to number of possible places. Output value of the n^{th} neuron is considered to be likelihood of person being in place n. To calculate confidence of being in that place we simply divide likelihood by the sum of all likelihoods.

Predicting one step in the future is very simple. To predict more steps we are using values predicted before. If we used the place with highest confidence as the input for the next step we would end up with totally wrong prediction if the network was uncertain in the first prediction. To give our predictor better ability to recover from errors or uncertainties which are natural in human behaviour we propose algorithm to calculate prediction confidences differently.

The confidence of being in place n in the k^{th} step is defined as weighted sum of probabilities of all possible paths how person could get to place n. Calculating all possible paths is unrealistic as the number of paths explodes with increasing k so we prune the state space and calculate probabilities only for 100 most probable paths.

2.2.5.1 Prediction decoding example

In the next example we will show how exactly the prediction is calculated for more than one step to the future with Neural Turing Machine predictor. Figure 2.6 shows all possible paths for person from its current location to the third predicted step.

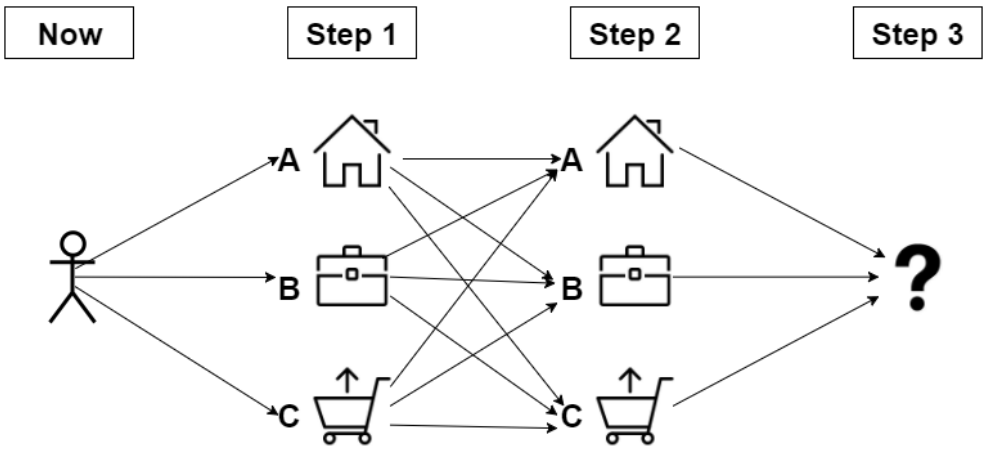


Figure 2.6: Possible paths for person.

Each of this possible paths will bring Neural Turing Machine used for prediction to different internal state. Different internal state means different prediction confidences in the next predicted step, therefore, we need to take all of these possibilities into account. Every path has its probability that is calculated as joint probability of every step in path. Summing confidences of every

Path	Path probability			Step 3 confidence			Step 3 probability		
	Step1	Step2	Total	A	B	C	A	B	C
AA	0.30	0.80	0.24	0.40	0.30	0.30	0.096	0.072	0.072
AB	0.30	0.20	0.06	1.00	0.00	0.00	0.060	0.000	0.000
AC	0.30	0.00	0.00	0.00	1.00	0.00	0.000	0.000	0.000
BA	0.60	0.90	0.54	0.60	0.00	0.40	0.324	0.000	0.216
BB	0.60	0.00	0.00	0.80	0.10	0.10	0.000	0.000	0.000
BC	0.60	0.10	0.06	0.20	0.50	0.30	0.012	0.030	0.018
CA	0.10	0.90	0.09	0.10	0.90	0.00	0.009	0.081	0.000
CB	0.10	0.10	0.01	0.00	0.00	1.00	0.000	0.000	0.010
CC	0.10	0.00	0.00	0.60	0.10	0.30	0.000	0.000	0.000
							0.501	**0.183**	**0.316**

Table 2.1: Long term prediction example calculation

possible path weighted by probability of the path gives us place confidences for the next step as shown in table 2.1. Figure 2.7 shows the most probable path for the person as predicted.

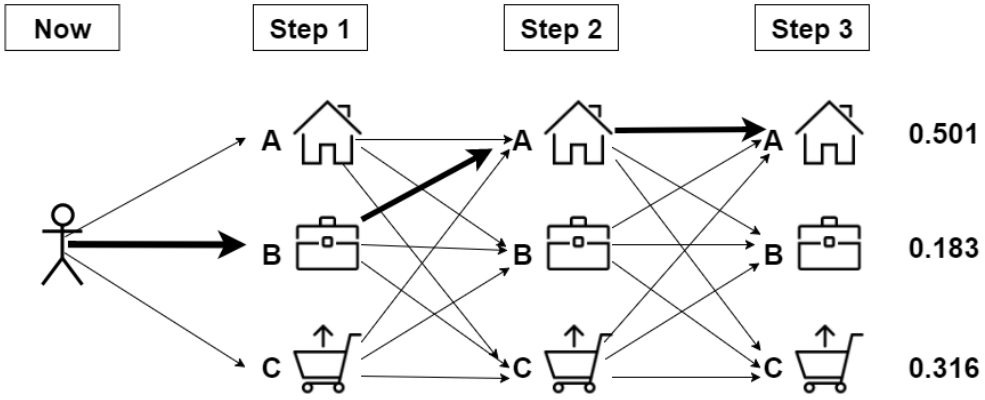


Figure 2.7: Most probable path predicted for the person

Chapter 3

Implementation and testing

Within the scope of this work we have implemented complete system for behaviour analysis of the person. It takes location history as an input and on the output we have important places for the person and ability to predict his mobility. The whole system is implemented in C#. The most difficult part was implementation of Neural Turing Machine library. This library is standalone and published in GitHub repository `github.com/JanTkacik/NTM`. In the next section we will describe its implementation in details.

3.1 Neural turing machine library

Neural turing machine was introduced by Alex Graves, Greg Wayne and Ivo Danihelka in their work Neural Turing Machines [16]. According to this paper we have implemented NTM together with learning algorithm. As a controller we chose feed-forward neural network for its simplicity. Using more complex controller like LSTM does not bring any significant performance improvement according to [16]. In our implementation we did not introduce any additional restrictions on NTM settings. One can choose any size of controller, memory and any number of heads.

The library is implemented in C# with use of .NET Framework. There are no other additional dependencies. Every part of NTM has its own class and is replaceable. It is very easy to make improvements or changes to the library. The main modules are controller, addressing and memory.

Implementation of controller and memory is straightforward with no changes compared to what have been proposed in the paper. As mentioned before we rely on our own implementation of feed-forward neural network as controller. The more interesting part of implementation is implementation of addressing mechanism. Overall view of addressing mechanism is on figure 3.1.

As a learning algorithm we chose back-propagation through time with rmsprop used for weight optimization. In our implementation any weight optimization method can be used with back-propagation. Back-propagation through

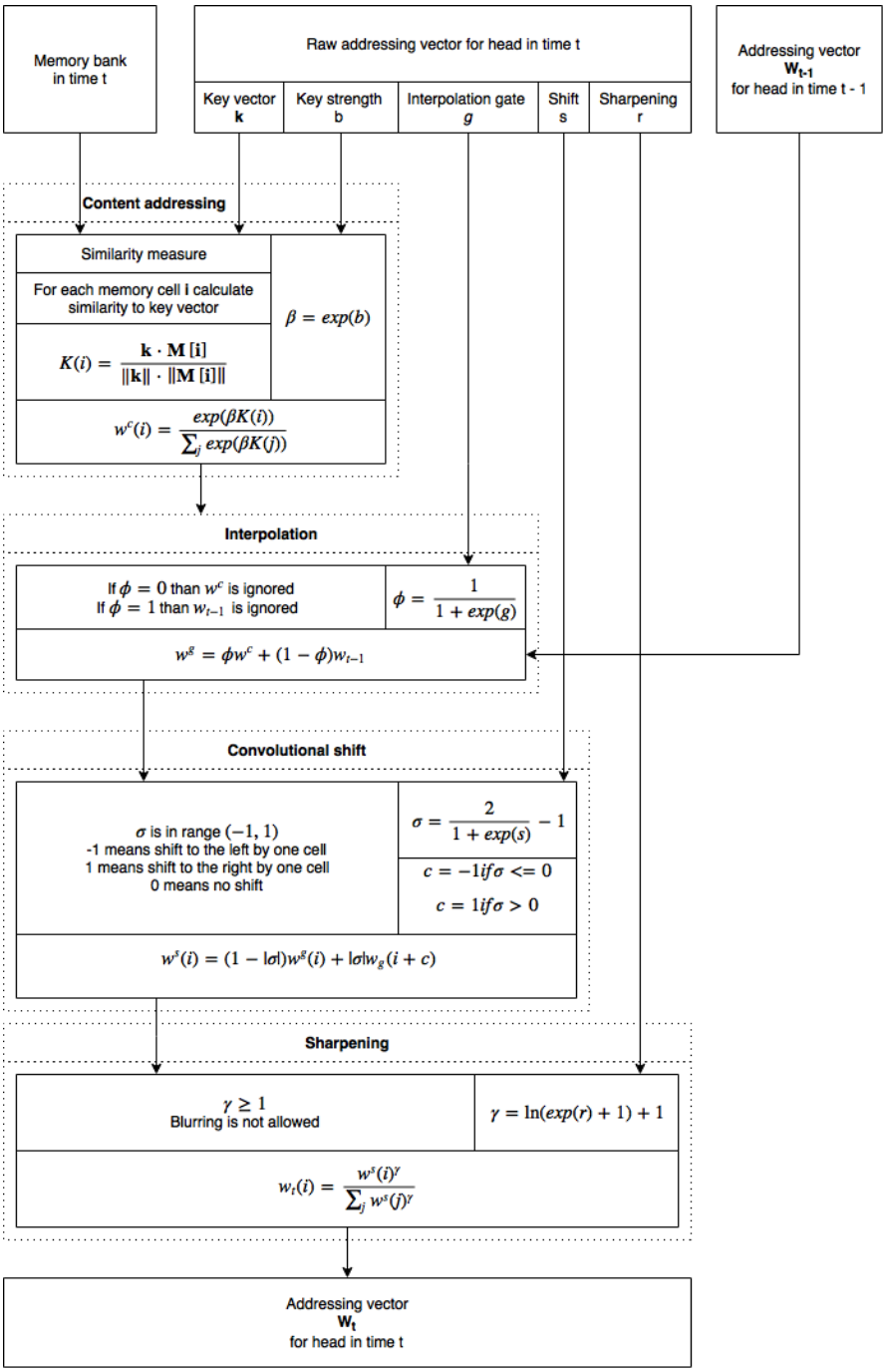


Figure 3.1: **Neural Turing Machine addressing mechanism as implemented in NTM library.** Content of memory bank, controller output represented by raw addressing vector and last addressing vector produces addressing vector for the next step. Convolutional shift implemented allows shifting at most by one cell.

time is not very efficient algorithm. For every learning step we need to make as many copies of underlying NTM as the length of the sequence. Because of that, learning NTM by using library provided is extremely demanding on memory. We did not do any optimizations on memory usage so learning large networks could fail.

We have tested library functionality by implementing learning tasks as proposed in [16]. We have successfully reproduced results on copy task, repeat copy task and n-gram task. Implementation of these tests is part of the library.

3.2 Behaviour analysis application

We have implemented software for behaviour analysis of a person. It has been written in C# with use of .NET Framework. For the GUI we used WPF which restricts its usage to operating system Windows. This is the only requirement of the application. Screenshots from the application are in appendix. Its basic features are:

- Data import from CSV
- Detection of important places for the person
- Ability to examine detected places in detail
- Person mobility prediction
- Map visualizations (Open Street Map)
- Installation and updates through Click-Once technology

3.2.1 Important place detection

All of the proposed methods for place detection are available in GUI. User can set all of the method parameters freely and experiment with them. Implementations of DBSCAN, WDBSCAN, LDBSCAN, AgarClust and Hierarchical AgarClust did not require any special libraries. It is very easy to add new method for important place detection and integrate it with map visualization environment. User is able to examine details for each cluster detected.

3.2.2 Mobility prediction

For the mobility prediction there are two options available. Fast KNN based predictor and slower but more accurate NTM predictor. Training of predictors can take a long time so it runs on background. User can watch predictor predictions for the validation set. User can again freely set any of the methods parameters. After the predictor is learned, user can choose number of steps to predict and show the predictions on heat-map and map.

KNN based predictor uses Accord-Framework library for KNN algorithm implementation as well as genetic algorithm. NTM predictor uses NTM library described in previous section.

Chapter 4

Experiments

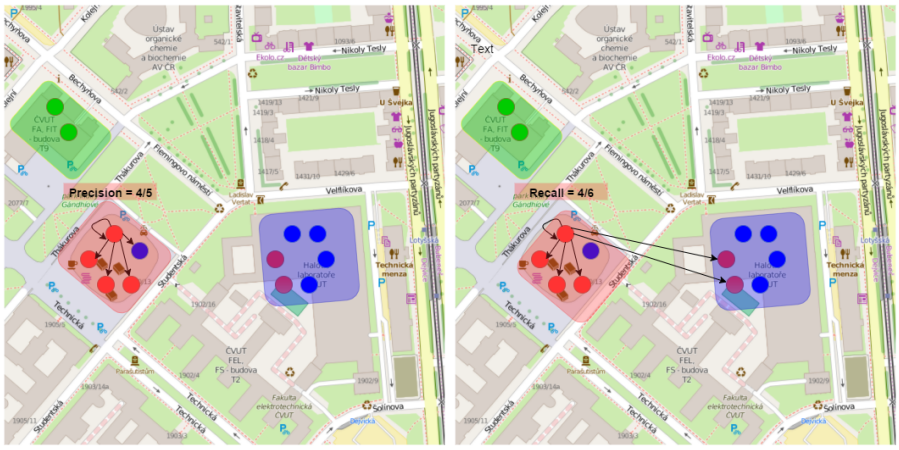
4.1 Important place detection

To validate proposed important place detection methods and to evaluate their performance, we have created a dataset containing location history for 6 people together with manually defined important places for every person. The data for every person are data taken from android location history service. People were asked to mark places important for them as polygons on the map with historical points displayed. Dataset is published on GitHub in repository `github.com/JanTkacik/personal-mobility-dataset`.

Performance evaluation of our methods is comparison of two clusterings. The one detected by the system with the ground truth defined by interviewed person. As the base evaluation metrics we chose BCubed metrics family proposed by Bagga and Baldwin in [20]. BCubed metrics defines precision and recall for each point in dataset. The point precision represents how many points from the same place detected belong to known place defined by interviewed person. Symmetrically, the recall represents how many points from known place appears in its detected place. For better illustration see figure 4.1. If the point has a high recall we will find most of the points from one known place in same detected place. On the other hand, if the point has a high precision there are only a few noisy points in the detected place. Both of these metrics are important for us because for later movement prediction we need detected places to be as clean as possible and we also want to cover as much points from known places as possible.

Another important thing to notice is that we assume that interviewed person does not remember every place visited but the places marked are marked correctly. This is the reason why we cannot measure quality of noise detection but only precision and recall of place detection. When calculating BCubed metrics we omit points that are not in any known polygon and points marked as noise by place detection algorithm.

As mentioned before, people tend to spend most of their time in a few



(a) Precision (b) Recall

Figure 4.1: Example calculation of BCubed precision and recall for one point.

places. This fact causes that using just BCubed metrics is insufficient as the difference between two place detections, one with only a few most frequent places detected correctly and the other with all places detected correctly will be very small. To face this issue we define mapping between places defined by interviewed person and detected places based on BCubed F-Score. For every place defined by interviewed person we chose zero or one place from detected places and vice versa. If there are no matching places detected it means that all points of place in one clustering are defined as noise in the other. In table 4.1 there are defined all possible mapping cardinalities and their interpretations in place detection performance evaluation context. Please notice that we are not able to recognise invalid places detected as we assume that interviewed people did not remember all of the places.

We further defined known places detection precision as

$$Precision = \frac{3 * Correct + 2 * Split + 1 * Merged}{3 * Marked} \tag{4.1}$$

This measure is invariant to point count distribution between places. It prefers split places over merged as for prediction purposes it is better to have place split than merged. If the place is split we can recognize it later and merge the places but we are not able to split merged place. Parameters of all proposed methods for place detection have been optimized by particle swarm optimization method with fitness function defined as average of BCubed F-Score and known places detection precision for all people in dataset. We did not optimize parameters for every person separately as we want to find settings that will work well for many people. Overall results are shown in tables 4.2 and 4.3.

(a) Correct (b) Split

(c) Merged (d) Mixed

(e) Not Found

Figure 4.2: **Place matching possibilities visualization.** Red places are defined by person, blue are detected by the system.

	Manually defined places count	Detected places count	Interpretation	Figure
Correct	1	1	Correctly found place	4.2a
Split	1	2+	One place detected as two or more	4.2b
Merged	2+	1	Two or more places detected as one	4.2c
Mixed	2+	2+		4.2d
Not Found	1	0	Two or more places detected as one	4.2e

Table 4.1: Place matching possibilities

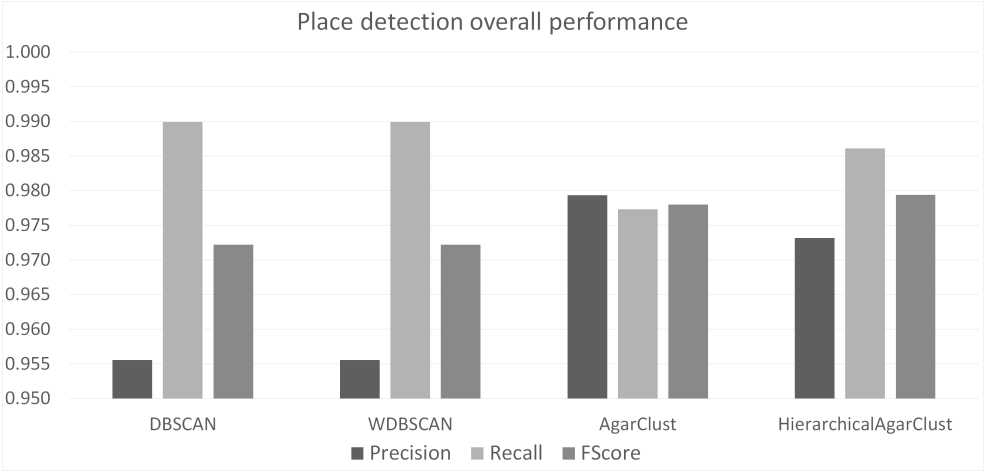


Figure 4.3: Overall place detection performance - BCubed metrics

	Person	Correct	Split	Merged	Mixed	Not found	Accuracy
DBSCAN	A	23	3	0	0	0	0.962
	B	16	0	0	0	0	**1.000**
	C	10	1	2	0	0	0.872
	D	3	1	0	0	0	0.917
	E	13	3	0	0	0	**0.938**
	F	14	5	2	0	0	0.857
WDBSCAN	A	23	3	0	0	0	0.962
	B	16	0	0	0	0	**1.000**
	C	10	1	2	0	0	0.872
	D	3	1	0	0	0	0.917
	E	13	3	0	0	0	**0.938**
	F	14	5	2	0	0	0.857
LDBSCAN	A	3	1	18	0	4	0.439
	B	2	2	12	0	0	0.458
	C	7	2	0	0	4	0.926
	D	4	0	0	0	0	**1.000**
	E	3	0	11	0	2	0.476
	F	5	0	15	0	1	0.500
AgarClust	A	24	2	0	0	0	0.974
	B	14	2	0	0	0	0.958
	C	12	1	0	0	0	**0.974**
	D	3	1	0	0	0	0.917
	E	13	3	0	0	0	**0.938**
	F	18	3	0	0	0	**0.952**
Hierarchical AgarClust	A	25	1	0	0	0	**0.987**
	B	15	1	0	0	0	0.979
	C	9	2	2	0	0	0.846
	D	3	1	0	0	0	0.917
	E	13	3	0	0	0	**0.938**
	F	18	3	0	0	0	**0.952**

Table 4.2: Overall place detection performance - place matching metrics

	Person	Precision	Recall	FScore
DBSCAN	A	0.918	0.978	0.947
	B	0.972	1.000	0.986
	C	0.949	0.998	0.973
	D	0.992	0.981	0.987
	E	0.988	0.992	0.990
	F	0.914	0.991	0.951
	Average	**0.956**	**0.990**	**0.972**
WDBSCAN	A	0.918	0.978	0.947
	B	0.972	1.000	0.986
	C	0.949	0.998	0.973
	D	0.992	0.981	0.987
	E	0.988	0.992	0.990
	F	0.914	0.991	0.951
	Average	**0.956**	**0.990**	**0.972**
LDBSCAN	A	0.872	0.948	0.909
	B	0.794	0.570	0.664
	C	0.929	0.853	0.889
	D	0.919	0.856	0.887
	E	0.962	0.989	0.975
	F	0.914	0.940	0.926
	Average	**0.898**	**0.859**	**0.875**
AgarClust	A	0.926	0.982	0.954
	B	0.991	0.923	0.956
	C	0.983	0.998	0.990
	D	0.992	0.981	0.987
	E	0.992	0.988	0.990
	F	0.992	0.992	0.992
	Average	**0.979**	**0.977**	**0.978**
Hierarchical AgarClust	A	0.925	0.980	0.952
	B	0.987	0.976	0.981
	C	0.950	0.997	0.973
	D	0.992	0.981	0.987
	E	0.993	0.990	0.992
	F	0.991	0.994	0.992
	Average	**0.973**	**0.986**	**0.979**

Table 4.3: Overall place detection performance - BCubed metrics

We can see that there is no difference between DBSCAN and WDBSCAN as in our dataset most of the data are sampled regularly. WDBSCAN performs better only when there is irregular time interval between samples and the samples are frequent enough so that we can approximate stay time in point better. Both of these methods perform very well overall. We can notice that almost all points that were marked by interviewed person were part of the some detected place but the places are more noisy than the places detected by AgarClust method. AgarClust method have similar overall performance as (W)DBSCAN and there is no noticeable tendency to merge places together and detect them as one. Hierarchical AgarClust did not bring any significant improvement over more simple AgarClust method. Result of Hierarchical AgarClust method together with LDBSCAN method proved that (W)DBSCAN and AgarClust are able to detect places of variable sizes without any modifications.

4.2 Movement prediction

Movement prediction was tested and evaluated on two different dataset. First dataset contains movement history of ideal artificial people. We created this dataset to test basic capabilities of our predictive models. Second dataset is data as described in previous section. This dataset should show us predictive power of our model on real world data as well as global performance of proposed system.

4.2.1 Performance testing details

To measure performance for each person we reserved last week for testing purposes. Predictors were learned on the remaining historical data. We used 1 hour sample rate. After learning predictors, we have simulated the following scenario. Every hour prediction is made for the next 24 hours, correct new point is saved to the predictor and the next 24 hours are predicted. This scenario should simulate one week of predictor usage without additional training but with new correct data gathered from the input system. The result is 144 measurements of prediction accuracy each for 24 hours.

We have defined following performance measures. AccSoft@n defines soft accuracy of prediction on the n^{th} predicted step. AccBest@n defines average number of matches between real true place of person and the predicted place with best confidence on the n^{th} step. Example calculation is shown in table 4.4.

	Step 1			Step 2			Step 3		
	A	B	C	A	B	C	A	B	C
Truth	1	0	0	1	0	0	0	0	1
Prediction	**0.8**	0.1	0.1	0.4	**0.6**	0.0	0.3	0.3	**0.4**
AccSoft@n		0.8			0.4			0.4	
AccBest@n		1			0			1	

Table 4.4: Example calculation of AccSoft@n and AccBest@n

AccSoft(n) defines average soft accuracy of prediction through n steps and AccBest(n) defines average number of matches between real location of person and the predicted place with best confidence through n steps.

$$AccSoft(n) = \sum_{i=1}^{n} \frac{AccSoft@i}{n} \tag{4.2}$$

$$BestSoft(n) = \sum_{i=1}^{n} \frac{AccBest@i}{n} \tag{4.3}$$

4.2.1.1 NTM predictor parameters

NTM predictor has 4 basic parameters that can affect its performance (controller size, memory width, memory length and number of heads). Rmsprop used for its weight optimization has another parameters that affect learning and another important parameter is sequence length that will be used in learning process. Moreover two runs with completely same settings may end up with different results as there is random initialization and random choice of sequences that will be presented to the NTM throughout learning process. It is clear that parameter optimization is not a simple task in this case.

Based on a experience with NTM learning we have decided to choose parameters of NTM predictor as shown in Table 4.5. We have also experimented with other NTM predictor settings as well but this one offers best ratio of prediction accuracy to learning speed.

Parameter	Setting
Learning sequence length	168
Controller size	75
Number of heads	1
Memory size	32 x 16
Learning rate	10^{-4}
Momentum	0.9

Table 4.5: NTM predictor settings used in experiments

4.2.2 Artificial dataset test

4.2.2.1 Dataset description

We have created dataset with location history with 7 simple artificial people. Each person should test some basic ability of predictors.

The first artificial person, Adam, is absolutely regular and therefore absolutely predictable. This person have 3 basic day cycles and 3 places. The first day cycle is used for workdays. Every workday Adam go to work exactly at 9am and come back at 5pm. The second cycle is used for Saturdays. On Saturday artificial person stays at home whole day. The last, Sunday cycle is similar to weekday cycle. The only difference is that Adam does not go to work at 9am for 8 hours but go to church at 10am for 2 hours. Adam should test that predictors are able to learn weekly and daily cycles that are completely predictable.

Behaviour of the second artificial person, Bob, is almost exactly the same as Adam‘s behaviour. The only difference between them is that on weekdays after work, between 6pm and 9pm, Bob goes to shop with 25% probability. Bob is therefore not completely predictable, but we can easily prove that his long term predictability is 97%. Bob should test the ability of predictors to deal with person‘s randomness. We will also look the accuracy of prediction and how it approaches the predictability limit.

The third artificial person, Carl, is very similar to Bob. The difference between them is the way how they visit shop after work. While Bob goes to shop and home totally randomly, Carl only goes to the shop at random time but when he is there he always stays there till 9pm. His predictability is 96%. Carl should test that predictor is able to learn the behaviour rule in form - person stays in place till specific time regardless of arrival time.

Dan, the fourth artificial person is very similar to Adam. The only difference between them is that Dan does not go to work exactly at 9am. The probability that he goes to work is 33% at 7am, 33% at 8am and 33% at 9am. The only thing that is stable is work time. Dan always stays at work 8 hours

and then goes straight home. Predictability of Dan is 98%. Dan should test that predictor is able to learn the behaviour rule in form - person stays in place for specific amount of time regardless of arrival time.

The fifth artificial person, Earl, behaved exactly like Bob for the one month, but then he moved and changed his behaviour and now he behaves exactly like Carl. Earl‘s predictability is now the same as Carl‘s. Earl should test that predictor is able to adapt to new behaviours and forget the old habits.

Another artificial person, Fred is absolutely predictable like Adam but he does not have weekly cycle. He lives in four day cycle and visits only 3 places. On the first day he works from midnight till noon. On the second day he works from noon till midnight. Later he spends his whole third day at home and on fourth day he goes to church from 9am till 11am. Then he repeats his 4-day cycle. Fred should test if predictors are able to learn other than weekly cycles.

The last artificial person is Greg. Greg is very strange person but again absolutely predictable. His only life cycle last for 7 hours. He is at home for the first 2 hours of his day then goes to work for 4 hours and then to shop for 1 hour. Then he repeats this cycle again and again. Greg should test if predictors are able to learn cycles with period other than 24 hours.

4.2.2.2 Results

Tables 4.6 and 4.7 show overall results of our predictors on artificial dataset described in previous section. Please notice that results in tables are rounded to 3 decimal places. We can see that both methods were able to model Adam and Greg perfectly. There were also no problems with modelling Bob and Carl. Prediction accuracy for both of them was close to the maximum achievable value. The first noticeable difference between these two methods was at modelling Dan. KNN predictor unlike NTM predictor was not able to count number of hours spent at work. NTM is also better at adapting to new behaviour. We can see it from Earl‘s prediction accuracy results. The biggest problem for both predictors have been Fred‘s 4-day cycle. KNN cannot learn cycles with period that is not multiply of 7. The NTM have done better job but Fred‘s results are not as good as Adam‘s or Greg‘s even through all of them are completely predictable. The problem is caused by the input encoding which makes learning more common weekly cycles easier for NTM but, on the other hand, makes learning cycles with other periods harder.

Method	Person	Predictability	AccSoft				
			1	3	6	12	24
KNN	Adam	1.00	1.000	1.000	1.000	1.000	1.000
	Bob	0.97	0.932	0.932	0.932	0.932	0.932
	Carl	0.96	0.945	0.945	0.945	0.945	0.945
	Dan	0.98	0.930	0.931	0.931	0.931	0.931
	Earl	0.96	0.911	0.820	0.763	0.777	0.827
	Fred	1.00	0.901	0.837	0.759	0.615	0.511
	Greg	1.00	0.973	0.991	0.995	0.998	0.999
NTM	Adam	1.00	1.000	1.000	1.000	1.000	1.000
	Bob	0.97	0.951	0.945	0.939	0.934	0.939
	Carl	0.96	0.952	0.945	0.942	0.934	0.924
	Dan	0.98	**0.972**	**0.956**	0.947	0.934	0.924
	Earl	0.96	0.897	**0.866**	**0.843**	**0.832**	0.834
	Fred	1.00	0.921	**0.904**	**0.894**	**0.872**	**0.858**
	Greg	1.00	**1.000**	1.000	1.000	1.000	1.000

Table 4.6: Artificial dataset overall prediction results - AccSoft metric

Method	Person	Predictability	AccBest				
			1	3	6	12	24
KNN	Adam	1.00	1.000	1.000	1.000	1.000	1.000
	Bob	0.97	0.932	0.932	0.932	0.932	0.932
	Carl	0.96	0.959	0.959	0.959	**0.959**	**0.959**
	Dan	0.98	0.932	0.932	0.932	0.932	**0.932**
	Earl	0.96	**0.911**	0.820	0.763	0.777	0.827
	Fred	1.00	0.932	0.865	0.785	0.638	0.520
	Greg	1.00	0.973	0.991	0.995	0.998	0.999
NTM	Adam	1.00	1.000	1.000	1.000	1.000	1.000
	Bob	0.97	**0.960**	**0.958**	**0.957**	**0.956**	**0.961**
	Carl	0.96	0.951	0.944	0.942	0.936	0.916
	Dan	0.98	**0.968**	0.948	0.938	0.921	0.907
	Earl	0.96	0.890	**0.861**	**0.837**	**0.826**	0.830
	Fred	1.00	0.952	**0.934**	**0.928**	**0.905**	**0.885**
	Greg	1.00	**1.000**	1.000	1.000	1.000	1.000

Table 4.7: Artificial dataset overall prediction results - AccBest metric

4.2.3 Real world dataset test

4.2.3.1 Dataset description

This dataset contains the same data used for place detection evaluation. For 6 real people there are 3 month history with manually marked places. Other works mostly use Geolife dataset [21] for similar purposes. It contains recorded trajectories for 182 people. The only problem is that it does not contain true information about personally important places. The same problem is with potential usage of other data sets, for example the one used in [22].

4.2.3.2 Results

Overall results are shown on tables 4.8 and 4.9. We can see that for most cases NTM predictor is significantly more accurate than KNN predictor. For real data it is very complicated to calculate predictability or regularity of a person. The results, however, correlate with what we know about the people. For example people A and E are both university students with part-time job. Most of the time they obey some schedule, in school or work, what results in high expected predictability. On the other hand interviewed person F is running his own business and likes to travel a lot in his free time. He works with almost no fixed schedule what results in low expected predictability.

Figure 4.4 shows typical NTM predictor accuracy trend. We can see that prediction accuracy tends to go down rapidly for the first 5 predicted steps but remains relatively stable afterwards.

Method	Person	AccSoft				
		1	3	6	12	24
KNN	A	**0.877**	**0.829**	0.771	0.752	0.747
	B	**0.842**	0.783	0.693	0.607	0.632
	C	0.568	0.566	**0.564**	**0.563**	**0.561**
	D	**0.801**	**0.566**	0.396	0.316	0.293
	E	0.735	0.739	0.739	0.740	0.741
	F	0.322	0.326	0.328	0.332	0.353
NTM	A	0.813	0.797	0.785	**0.776**	0.753
	B	0.815	0.764	**0.719**	**0.675**	0.632
	C	**0.704**	**0.600**	0.537	0.490	0.460
	D	0.457	0.451	**0.450**	**0.437**	**0.397**
	E	**0.827**	**0.795**	**0.781**	**0.770**	**0.762**
	F	**0.630**	**0.584**	**0.527**	**0.462**	**0.442**

Table 4.8: Real world dataset overall prediction results - AccSoft metric

Method	Person	AccBest				
		1	3	6	12	24
KNN	A	0.877	0.829	0.771	0.744	0.736
	B	0.842	0.774	0.689	0.664	0.670
	C	0.568	0.566	0.564	0.563	**0.561**
	D	**0.801**	**0.566**	0.396	0.316	0.293
	E	0.815	0.813	0.817	0.829	0.832
	F	0.322	0.326	0.328	0.332	0.353
NTM	A	0.863	**0.865**	**0.849**	**0.837**	**0.816**
	B	**0.863**	**0.831**	**0.801**	**0.767**	**0.721**
	C	**0.884**	**0.801**	**0.694**	0.579	0.474
	D	0.452	0.445	**0.445**	**0.434**	**0.395**
	E	**0.863**	0.831	0.833	0.828	0.819
	F	**0.699**	**0.628**	**0.575**	**0.522**	**0.497**

Table 4.9: Real world dataset overall prediction results - AccBest metric

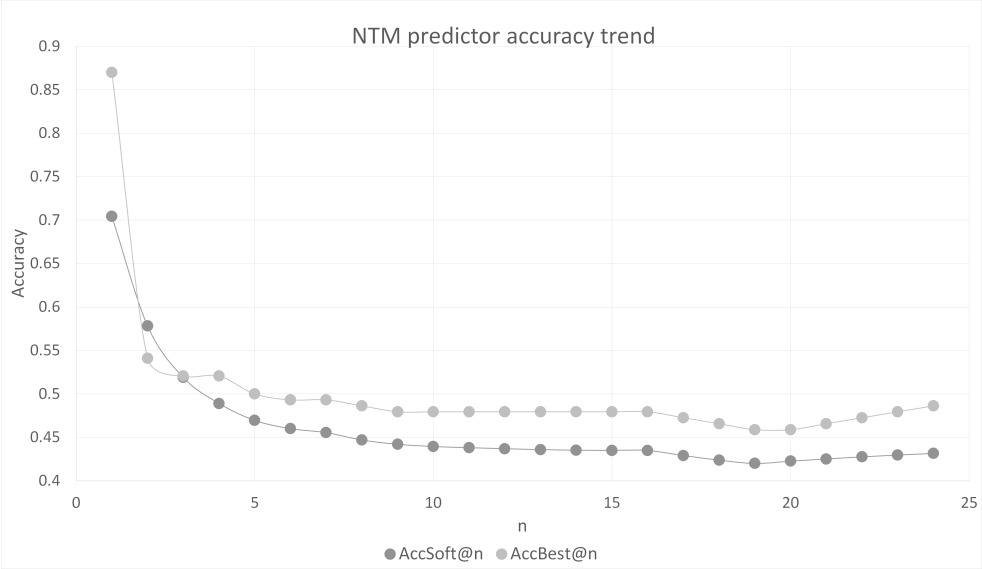


Figure 4.4: Neural Turing Machine predictor prediction accuracy trend.

Chapter 5

Application

Implemented application will be tested by detectives within the Police of the Czech Republic. Currently, no automated system is used for the behaviour analysis of suspects. Analysis is done only manually with help of GIS software used for data visualization or with text reports of raw positional data. It should help them to find interesting behaviour patters of suspects faster and easier than today.

The detectives should be able to use gathered location data and get interesting insights within a few minutes. The application supports data usage from multiple source systems and is optimized for large location histories.

As the whole process from data preprocessing to visualization of results is available through simple and comprehensive graphical user interface we expect that it should shorten the time from data gathering to its effective usage and allow detectives to see more in the data.

Conclusion

In the thesis we designed and implemented system for behaviour analysis of individuals that can be easily interpreted and understood by detectives and therefore used for automated analysis of suspects. The system is able to identify important places of an individual as well as detect noise in data automatically and create behaviour model of an individual suspect. This model can be used for prediction which reveals one‘s behaviour patterns. The results of every part of this process can be visualized on map.

To achieve person‘s model that can be easily interpreted and understood by detectives, we firstly identify person‘s important places and detect noise in data. Preprocessed locations are used to create a behaviour model of the person.

For the place detection we have proposed new algorithm AgarClust and compared it with DBSCAN and LDBSCAN. The algorithm was able to do place detection with almost 98% precision on provided dataset. This dataset is one of the contributions of the thesis as there was no public dataset available with interesting places marked down before.

We have proposed two methods for personal mobility prediction. The first uses KNN algorithm with advanced adaptive mechanism to fit every person‘s mobility patterns individually. The second method uses Neural Turing Machine — recurrent neural network with random access memory. We validated functionality of these methods on artificial dataset to understand their predictive power and also on real world dataset to evaluate overall system performance. We have shown that NTM predictor is able to learn many mobility patterns as well as adapt to changes in these patterns.

Another contribution of this thesis is implementation of Neural Turing Machine library in C#. It was later used to implement NTM predictor. The library is published on GitHub.

There are many possibilities for future work. When in comes to place detection as preprocessing step, it may be beneficial to look at the semantics of detected places. This knowledge could be used to more precise person‘s mo-

bility modeling and could help predictors to find a whole new level of mobility patterns as well as it can help predictors to adapt earlier to new patterns in personal mobility. As for NTM predictor and NTM itself, there are possibilities to investigate other options for NTM learning as well as other input encodings for the mobility prediction task.

Bibliography

[1] Gonzalez, M. C.; Hidalgo, C. A.; Barabasi, A.-L. Understanding individual human mobility patterns. *Nature*, volume 453, no. 7196, 2008: pp. 779–782.

[2] Chaoming Song, N. B. A.-L. B., Zehui Qu. Limits of Predictability in Human Mobility. *Science*, volume 327, 2010: pp. 1018–1021.

[3] Marmasse, N.; Schmandt, C. Location-Aware Information Delivery with-ComMotion. In *Handheld and Ubiquitous Computing*, *Lecture Notes in Computer Science*, volume 1927, edited by P. Thomas; H.-W. Gellersen, Springer Berlin Heidelberg, 2000, ISBN 978-3-540-41093-5, pp. 157–171, doi:10.1007/3-540-39959-3_12. Available from: `http://dx.doi.org/10.1007/3-540-39959-3_12`

[4] Ashbrook, D.; Starner, T. Learning significant locations and predicting user movement with GPS. In *Wearable Computers, 2002. (ISWC 2002). Proceedings. Sixth International Symposium on*, 2002, ISSN 1530-0811, pp. 101–108, doi:10.1109/ISWC.2002.1167224.

[5] Ester, M.; Kriegel, H.-P.; Sander, J.; et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, 1996, pp. 226–231.

[6] Zhou, C.; Frankowski, D.; Ludford, P.; et al. Discovering Personal Gazetteers: An Interactive Clustering Approach. In *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems*, GIS '04, New York, NY, USA: ACM, 2004, ISBN 1-58113-979-9, pp. 266–273, doi:10.1145/1032222.1032261. Available from: `http://doi.acm.org/10.1145/1032222.1032261`

[7] Pavan, M.; Mizzaro, S.; Scagnetto, I.; et al. Finding Important Locations: A Feature-Based Approach. In *Mobile Data Management (MDM), 2015*

16th IEEE International Conference on, volume 1, June 2015, pp. 110–115, doi:10.1109/MDM.2015.11.

[8] Alvarez-Lozano, J.; García-Macías, J. A.; Chávez, E. User Location Forecasting at Points of Interest. In *Proceedings of the 2012 RecSys Workshop on Personalizing the Local Mobile Experience*, LocalPeMA '12, New York, NY, USA: ACM, 2012, ISBN 978-1-4503-1639-2, pp. 7–12, doi: 10.1145/2365946.2365949. Available from: http://doi.acm.org/10.1145/2365946.2365949

[9] Yang, J.; Xu, J.; Xu, M.; et al. Predicting Next Location Using a Variable Order Markov Model. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-3139-5, pp. 37–42, doi: 10.1145/2676552.2676557. Available from: http://doi.acm.org/10.1145/2676552.2676557

[10] Alvarez-Lozano, J.; García-Macías, J. A.; Chávez, E. Learning and User Adaptation in Location Forecasting. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-2215-7, pp. 461–470, doi:10.1145/2494091.2495978. Available from: http://doi.acm.org/10.1145/2494091.2495978

[11] Duan, L.; Xiong, D.; Lee, J.; et al. A Local Density Based Spatial Clustering Algorithm with Noise. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 5, Oct 2006, pp. 4061–4066, doi:10.1109/ICSMC.2006.384769.

[12] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; et al. LOF: Identifying Density-based Local Outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, New York, NY, USA: ACM, 2000, ISBN 1-58113-217-4, pp. 93–104, doi: 10.1145/342009.335388. Available from: http://doi.acm.org/10.1145/342009.335388

[13] Siegelmann, H. T.; Sontag, E. D. On The Computational Power Of Neural Nets. *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, volume 50, no. 1, 1995: pp. 132–150.

[14] Graves, A. Generating Sequences With Recurrent Neural Networks. *CoRR*, volume abs/1308.0850, 2013. Available from: http://arxiv.org/abs/1308.0850

[15] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.*, volume 9, no. 8, Nov. 1997: pp. 1735–1780, ISSN 0899-7667,

doi:10.1162/neco.1997.9.8.1735. Available from: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[16] Graves, A.; Wayne, G.; Danihelka, I. Neural Turing Machines. *CoRR*, volume abs/1410.5401, 2014. Available from: http://arxiv.org/abs/1410.5401

[17] Uncharted Software Inc. GeoTime for Law Enforcement. Available from: http://www.geotime.com/Product/GeoTime.aspx

[18] TrackGroup. Platforms & Applications. Available from: http://www.trackgrp.com/products-services/platforms-applications/

[19] QGIS. A Free and Open Source Geographic Information System. Available from: http://qgis.org/en/site/

[20] Bagga, A.; Baldwin, B. Algorithms for Scoring Coreference Chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, 1998, pp. 563–566.

[21] Zheng, Y.; Zhang, L.; Xie, X.; et al. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-487-4, pp. 791–800, doi:10.1145/1526709.1526816. Available from: http://doi.acm.org/10.1145/1526709.1526816

[22] Chon, Y.; Talipov, E.; Shin, H.; et al. SmartDC: Mobility Prediction-Based Adaptive Duty Cycling for Everyday Location Monitoring. *Mobile Computing, IEEE Transactions on*, volume 13, no. 3, March 2014: pp. 512–525, ISSN 1536-1233, doi:10.1109/TMC.2013.14.

Appendix A

Acronyms

BPTT Back-propagation through time

GIS Geographic information system

GUI Graphical user interface

KNN K nearest neighbours

LSTM Long short-term memory

NTM Neural Turing Machine

WPF Windows presentation foundation

Appendix B

Application screenshots

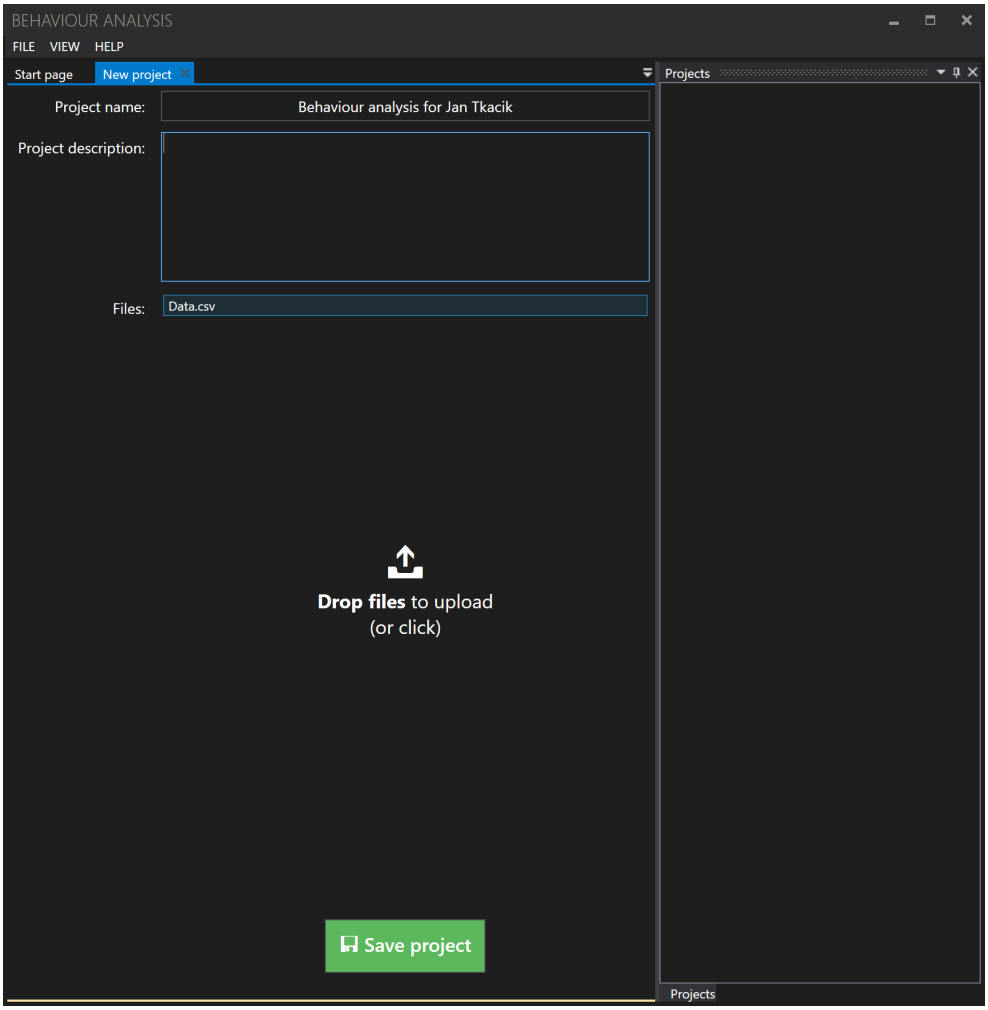


Figure B.1: **Project creation and data import view.** Firstly user need to create project and select import files that will be used for behaviour analysis.

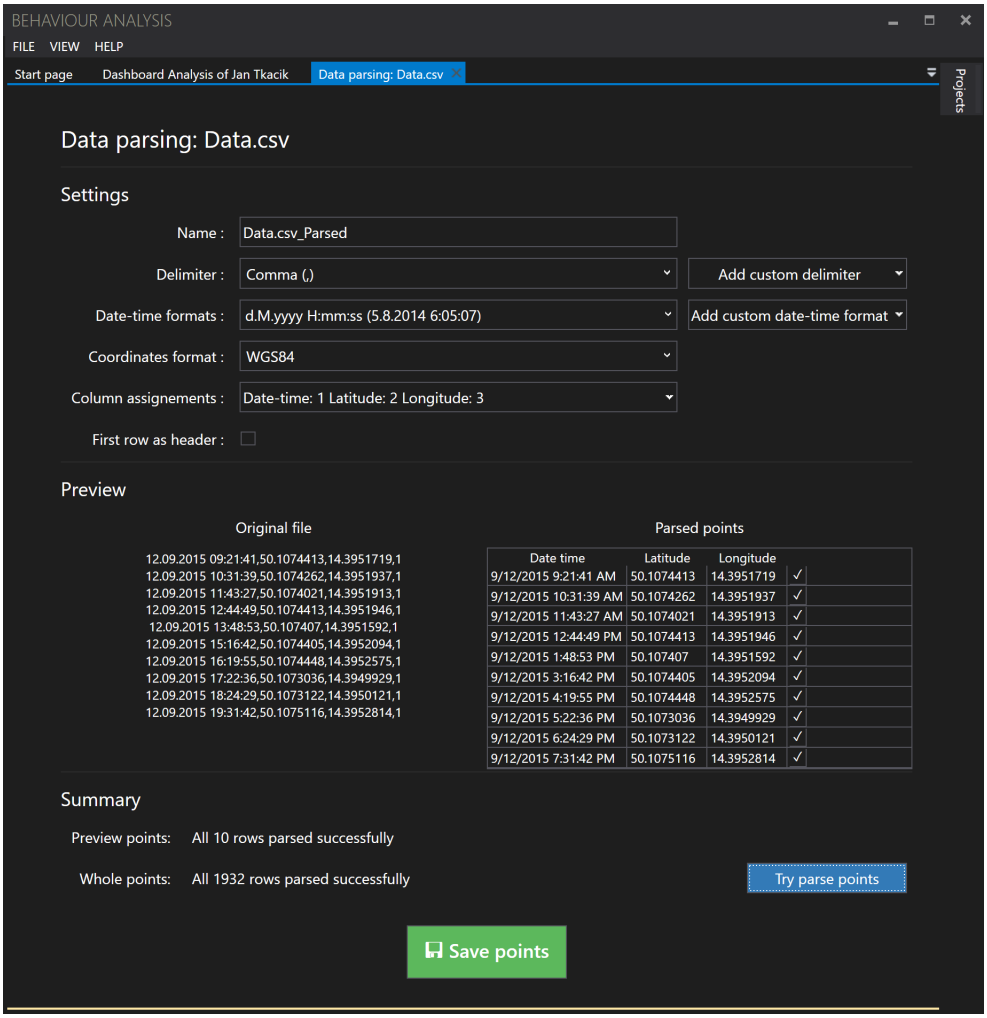


Figure B.2: **Data parsing view.** Every input file need to be parsed for further usage. User can set delimiter, date-time format, coordinates format and columns that will be used as input. Online parsing preview is shown for faster settings.

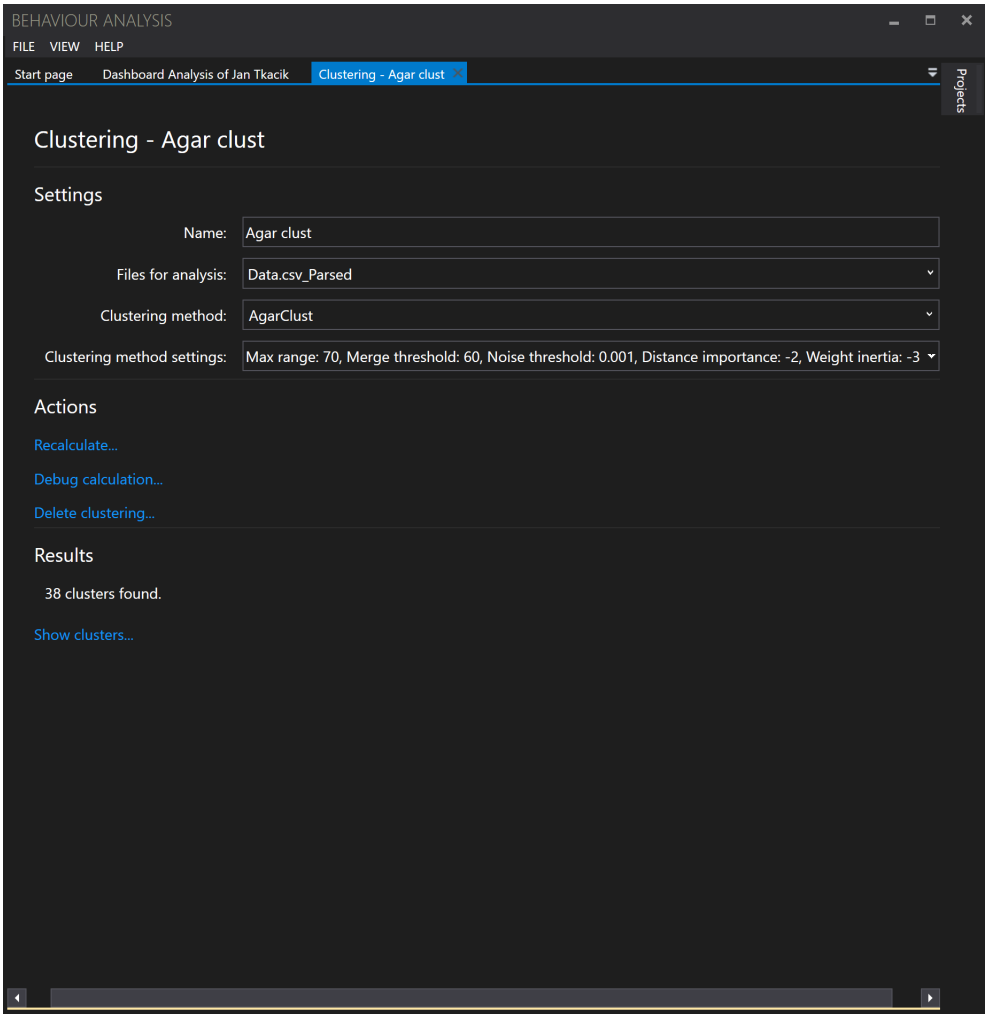


Figure B.3: **Place detection settings view.** On this view user can choose place detection method, its settings and run the calculation. After calculation is done user see number of detected places and can show places on map.

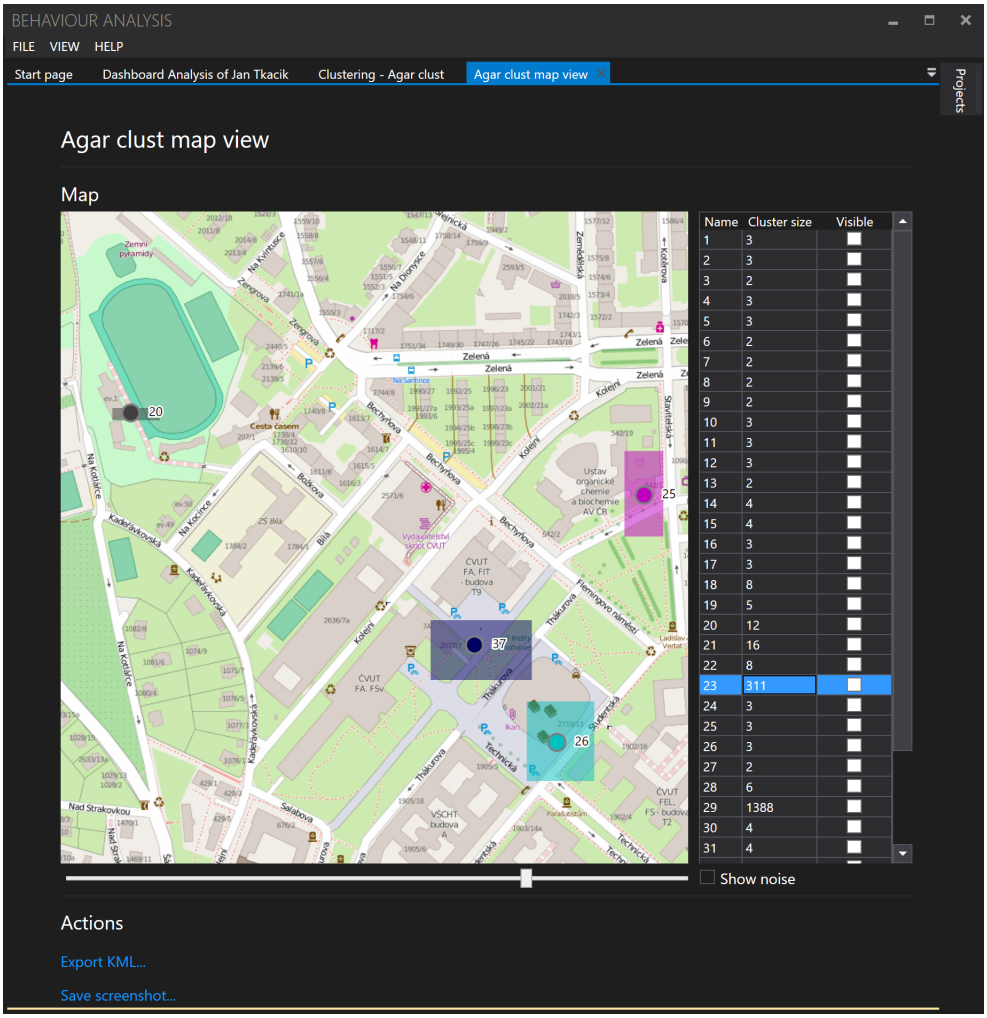


Figure B.4: **Place detection map view.** Places detected can be visualized on map. Open street maps are used for visualization. Double click on detected place shows place detail view.

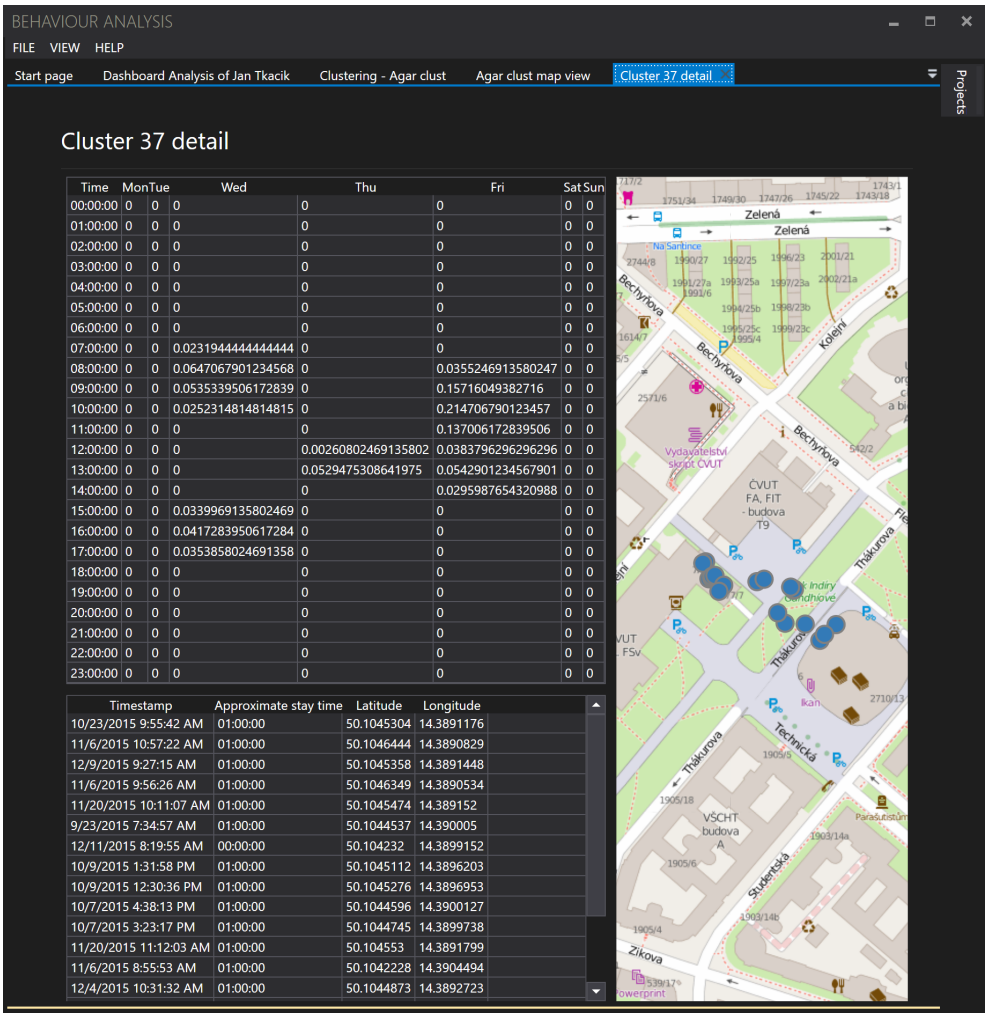


Figure B.5: **Place detail view.** On place detail view user can see all points that are in place on map as well as in table with approximate stay times. Contingency table provides further insight on how person visit the place.

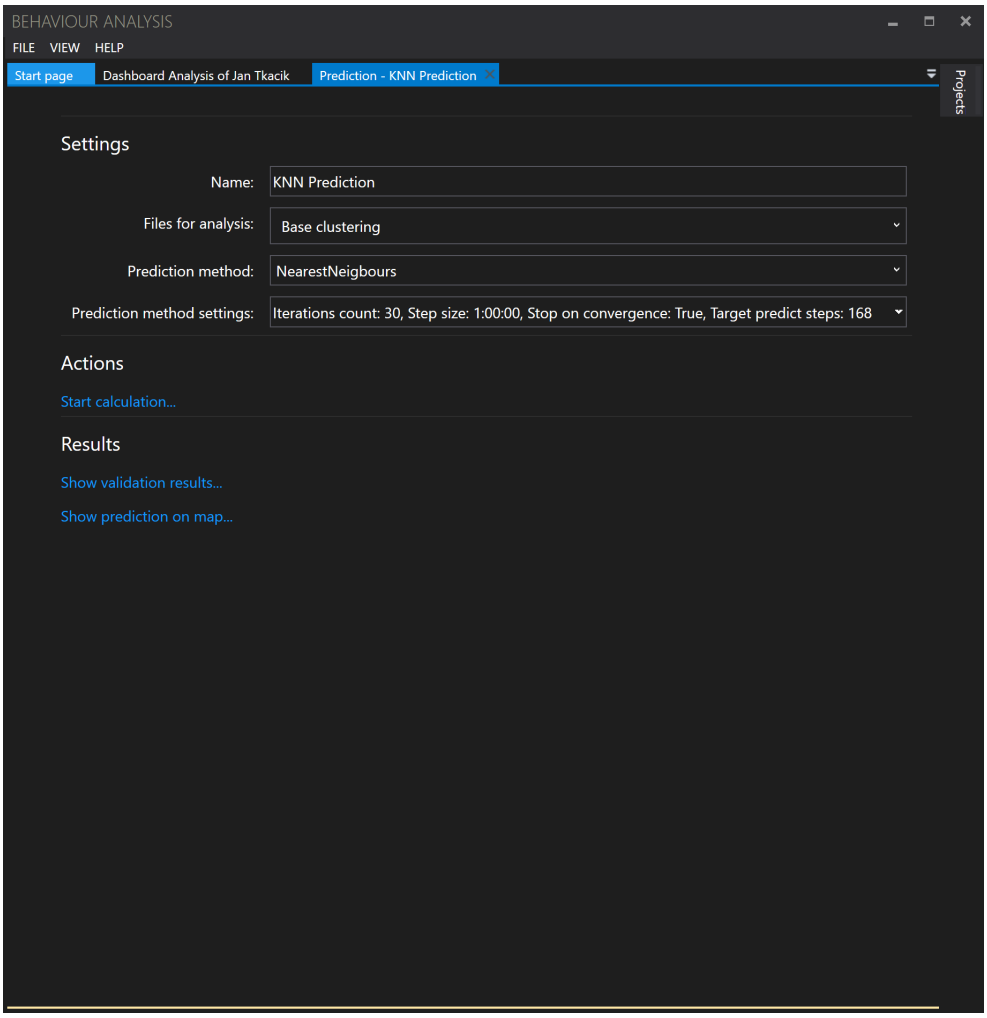


Figure B.6: **Prediction settings view.** On prediction settings view user can choose predictor that will be used. After calculation is done predictor can be used.

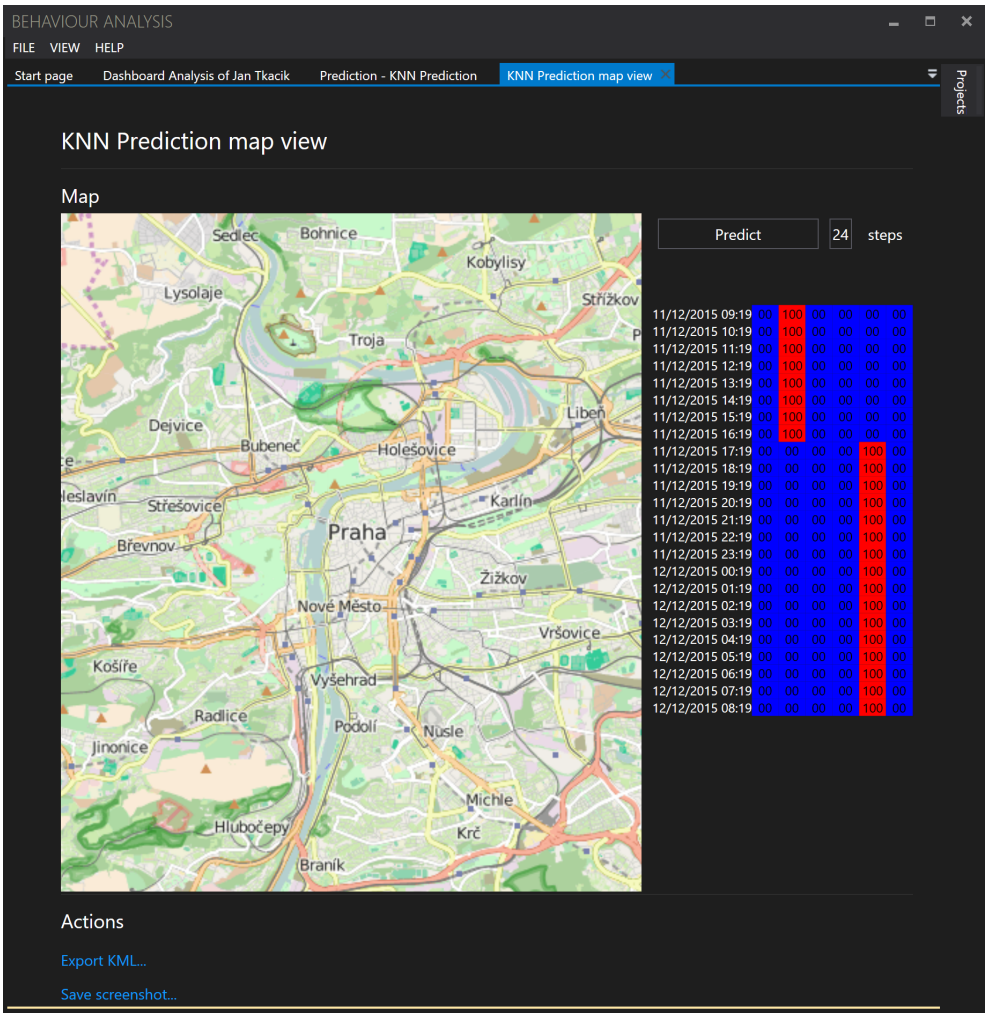


Figure B.7: **Prediction results view.** Prediction results are shown as heatmap with places as columns and predicted times as rows.

Appendix C

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory with source codes
  - `ntm` ..... implementation sources
  - `thesis` ..... the directory with LaTeX source codes of the thesis
- `dataset` ..... directory with dataset used
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format