Information Systems 98 (2021) 101452

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

# How meaningful are similarities in deep trajectory representations?

Saeed Taghizadeh [*], Abel Elekes, Martin Schäler, Klemens Böhm

*Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany*

ARTICLE INFO

*Article history:*
Received 31 December 2018
Received in revised form 23 September 2019
Accepted 25 September 2019
Available online 11 October 2019
Recommended by Ioannis Katakis

*Keywords:*
Trajectory similarity
Trajectory embedding models
Moving object databases
Trajectory databases
Trajectory clustering
Deep learning

ABSTRACT

Finding similar trajectories is an important task in moving object databases. However, classical similarity models face several limitations, including scalability and robustness. Recently, an approach named t2vec proposed transforming trajectories into points in a high dimensional vector space, and this transformation approximately keeps distances between trajectories. t2vec overcomes that scalability limitation: Now it is possible to cluster millions of trajectories. However, the semantics of the learned similarity values – and whether they are meaningful – is an open issue. One can ask: How does the configuration of t2vec affect the similarity values of trajectories? Is the notion of similarity in t2vec similar, different, or even superior to existing models? As for any neural-network-based approach, inspecting the network does not help to answer these questions. So the problem we address in this paper is how to assess the meaningfulness of similarity in deep trajectory representations. Our solution is a methodology based on a set of well-defined, systematic experiments. We compare t2vec to classical models in terms of robustness and their semantics of similarity, using two real-world datasets. We give recommendations which model to use in possible application scenarios and use cases. We conclude that using t2vec in combination with classical models may be the best way to identify similar trajectories. Finally, to foster scientific advancement, we give the public access to all trained t2vec models and experiment scripts. To our knowledge, this is the biggest collection of its kind.

## 1. Introduction

Advances in mobile computing have led to the generation of massive trajectory data [1]. A trajectory of a moving object usually is a sequence of GPS points. Finding similar trajectories is an elementary but important task in moving object databases (MODs). It is the foundation for downstream tasks such as trajectory clustering [1–5], movement pattern mining [6], finding popular routes [7] and discovering moving groups [8]. Different kinds of similarity models have been proposed, including longest common subsequence (LCSS) [9], edit distance on real sequences (EDR) [10], edit distance with real penalty (ERP) [11], model-driven assignment (MA) [12], and DISSIM [13]. All of them have been shown to be useful for some of the aforementioned tasks.

However, all these classical models face several limitations. A very important one is scalability [14]. The classical models hardly scale to datasets larger than several thousand trajectories [4,15–17]. In real-world scenarios however, the number of trajectories usually is in the millions.

It also has been shown in [14] that similarity computation among trajectories with the classical models has three other drawbacks in terms of robustness. This is mostly due to their local matching property, which matches the points of two trajectories that are closer to each other than a threshold. We illustrate these drawbacks with the following scenarios: First are scenarios with non-uniform sampling rates. This causes the GPS records to be dense in some time periods and sparse in others. With the existing models, it is hard or even impossible to identify similar trajectories if the sampling rates are different. The second scenario is when the sampling rate is low. Geo-tagged tweets or online check-ins are examples of such data [18]. The third scenario is when the data is noisy. Noise could come from different sources such as "privacy" (i.e., intentionally perturbing GPS data) or accidentally, due to, say, connection problems between the GPS device and the satellites.

To overcome these limitations, the authors of [14] adopted the idea of a similarity preserving embedding of a dataset (trajectories in this case) into a d-dimensional vector space. The approach is named t2vec (trajectory to vector) and is based on deep representation learning and sequence-to-sequence models [19]. This means that, with the use of neural networks, t2vec represents GPS-point-based trajectories as d-dimensional vectors so that vectors close to each other correspond to trajectories which are similar.

[*] Corresponding author.
*E-mail addresses:* saeed.taghizadeh@kit.edu (S. Taghizadeh), abel.elekes@kit.edu (A. Elekes), martin.schaeler@kit.edu (M. Schäler), klemens.boehm@kit.edu (K. Böhm).

https://doi.org/10.1016/j.is.2019.101452
0306-4379/© 2019 Elsevier Ltd. All rights reserved.

This idea is well known, for example, in natural language processing (NLP). Word embedding models such as word2vec [20] and Glove [21] have gained lots of attention. They have recently become key tools for many NLP tasks like part of speech (POS) tagging [22], named entity recognition (NER) [23], image annotation [24] and machine translation [19,25,26].

There is further research such as GRAIL [27] in the area of time-series analysis which addresses the problem of representation learning. They focus on time-invariant shifts in time series and are not directly applicable to noisy datasets.

These approaches show the usefulness of embedding models in general. But for t2vec we are not aware of any research on the semantics of the similarity values of the learned trajectory vectors. In this paper, we aim at evaluating these semantics. This is required to work with t2vec in the future in meaningful ways. This includes evaluating research questions such as: What do similarity values coming from this new embedding model mean? How do the parameters of the embedding model change the meaning of similarity values? For example, is it possible that in one model two trajectories which are 0.5-similar should be considered similar, and in another model trained with different parameter settings the same value implies dissimilarity? – These questions are building blocks for downstream tasks like trajectory clustering. Answering them helps to understand the notion of similarity in t2vec and ultimately to answer how good an embedding-based trajectory similarity model is.

On the other hand, classical similarity models have exact definitions of similarity, allowing humans to understand the logic behind the definitions. But when dealing with neural networks in general, the trained model is not readily perceivable by humans. So it is necessary to compare non-embedding (definition-based) classical models and embedding-based models. In the comparison we are evaluating the following important questions: What types of relationship exist between similarity values coming from an embedding model and a non-embedding model? What are the causes of the similarities and differences between them? Can we find mappings between similarity values of embedding models and non-embedding ones? Which model yields better results in downstream tasks, such as clustering of trajectories?

To our knowledge, there is not any prior work in the area of moving object databases which addresses these questions. Work most related to ours is in the area of word embedding models [28]. However, their method is not directly applicable to trajectory embedding models, as there are fundamental differences between word embeddings and trajectory embeddings, which we will point out in the related work section.

**Challenges.** Several issues arise when studying trajectory similarity models: Since the authors of [14] choose the parameters of embedding model experimentally, it remains to be investigated on how to choose the parameters of the new embedding model, and how this will affect the similarity values of the model. How can we compare models with different parameters, and how can we compare different models in general? Finally, how to choose an unbiased evaluation dataset?

Since the t2vec model highly depends on the parameter settings, we investigate the various combinations of its parameters to figure out how they change the similarity values. This helps us to understand their semantics and to prepare for the next step where we evaluate t2vec in downstream applications.

To our knowledge, there does not exist any trajectory-similarity benchmark or test set which let us quantify the quality of different models. This makes evaluating the goodness of t2vec and comparing it to other models challenging.

Regarding the choice of an unbiased evaluation dataset, different criteria may affect evaluation results adversely. For example, we have to use a training set where the trajectory distribution is similar to the entire dataset in every part of the full map. Otherwise, we should not expect the model to extract equally meaningful representations of trajectories on the whole map.

**Contributions.** Since embedding models such as t2vec are based on neural networks, it is not possible to understand the semantics of the learned vectors and their similarity directly by evaluating the underlying model. Our contribution is to propose a methodology to evaluate the meaningfulness of the deep trajectory representations in t2vec. It is two-fold: First, we evaluate how different parameter settings affect the similarity values of the t2vec model. This includes all parameters, such as the size of the training dataset and dimensionality of the deep representation vector. To do this, we first carefully build an evaluation set of trajectory embeddings, with different parameter settings, and then study the resulting distributions based on statistical tests. We use two well-known trajectory datasets, from Porto [29] and Beijing [30,31]. We conclude that the t2vec model is robust regarding parameterization, by showing that the similarity-value distributions are fundamentally very similar between models trained with different parameters.

We then turn to downstream tasks. In order to assess the goodness of t2vec, we compare it to baseline and state-of-the-art classical non-embedding models. Since in this paper we are employing high-quality trajectory data, we work with the LCSS [9] similarity model, which is the state-of-the-art model for such data [14]. We use Dynamic Time Warping (DTW) [32] as a baseline model. There are other classical similarity models such as edit distance with projections (EDwP) [18] that yield better results on non-uniform and low sampling rate trajectories, but dealing with such noisy trajectories is not the focus of this paper.

We evaluate the differences between t2vec and the classical models, such as their similarity-value distributions or the semantics of similarity used by them. By systematic qualitative evaluation, we identify different scenarios where the combination of the t2vec and classical models give more meaningful results than using them separately. The last part of the evaluation is quantifying the models in identifying similar trajectories, i.e., their capability of trajectory clustering. It turns out that t2vec is better when clustering trajectories while at the same time calculates trajectory similarity orders of magnitudes faster than classical models.

As a final contribution, we make all the created models and evaluation scripts publicly available on our website.[1] To our knowledge, this is one of the biggest collections of trajectory-embedding models trained with systematically different parameter settings. Building the models has taken more than a month using a modern machine; hence, it is a valuable resource for all researchers working in this area.

Lastly, our findings regarding deep learning frameworks in trajectory databases are not limited to this specific domain. In fact, our evaluation method can target any deep learning framework to provide better insights into the trained models. This is because we have not made any prior assumption concerning the underlying deep learning framework.

## 2. Fundamentals and notation

In this section, we first review classical similarity models, and then we introduce the t2vec model and its corresponding parameters. Finally, we define the default parameter setting which we use throughout this paper.

[1] http://dbis.ipd.kit.edu/2652.php.

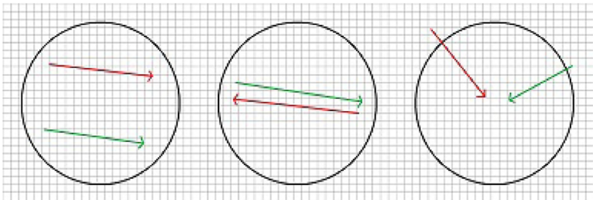

**Fig. 1.** Examples for ambiguous semantics of trajectory similarity.

### 2.1. Trajectory similarity in moving object databases

A Moving Object Database (MOD) is a database that can represent and query moving objects, i.e., objects whose position is changing over time. A trajectory $T \in \mathbf{T}$ in a MOD is defined as $T = ((p_1, t_1), \ldots, (p_n, t_n))$, where $p_k \in \mathbb{R}^2$ is a location in a 2-dimensional Euclidean space (i.e., map), and $t_k \in \mathbb{R}$ is a timestamp. A route $R \subset \mathbb{R}^2$ is a continuous curve in the Euclidean plane.

The question of how similar two trajectories $(T^1, T^2)$ in a MOD are, is not trivial, for several reasons. Fig. 1 shows three such examples in which it is not obvious whether we should deem the two trajectories similar or not. On the figures, the arrows represent the trajectories and the sphere represents the border of a city. For the first figure, the question is whether two parallel trajectories which are almost identical but shifted in space are similar or not. For the second one, it is whether two almost identical trajectories only going to the opposite direction are similar. Finally, are two trajectories both going from the side of the map into the center similar or not?

### 2.2. Basic trajectory similarity models

The naïve approach, to calculate the Euclidean distance (dist) at every point in time and use the sum of these values $\sum_{k\in\{1,2,\ldots,n\}} dist(p^1_k, p^2_k)$ as similarity, fails for several reasons. First, the points in time are not necessarily the same in the two trajectories, hence the location comparison may be biased. Second, and this is almost always the case in reality, if the lengths of the trajectories are different, one cannot use this formula at all.

To overcome these problems, warping-based distances have been proposed. They match the points of two different trajectories even when their timestamps are not equal, or their lengths are not the same. Well-known representatives are: DTW, LCSS, edit distance on real sequence (EDR) [11], and edit distance with real penalty (ERP) [10]. In this paper, we work with the DTW distance, a prominent trajectory similarity model.

*DTW fundamentals*. Dynamic Time Warping (DTW) is an algorithm used to measure the distance between two time-series. Trajectories of moving objects are sequences of GPS points and therefore they can be interpreted as time-series. To obtain the distance between two trajectories $T_i$ and $T_j$, DTW looks for the optimal match between the points of $T_i$ and $T_j$. The optimal match is the one with minimal cost, where the cost is the sum of the distances for each matched pair of GPS points.

Formally, the DTW operator is as follows:

$$DTW(T_i, T_j) = \begin{cases} 0 \quad \text{if } T_i \text{ and } T_j \text{ are empty} \\ \infty \quad \text{if } T_i \text{ or } T_j \text{ is empty} \\ dist(p^i_1, p^j_1) + \min(DTW(Tail(T_i), \\ Tail(T_j)), DTW(Tail(T_i), T_j)), \\ DTW(T_i, Tail(T_j)) \quad \text{otherwise} \end{cases}$$

where $Tail(T) = (p_2, \ldots, p_n)$ is the sequence of the last $n-1$ points in trajectory $T(p_1, p_2, \ldots, p_n)$.

Since DTW is a distance measure, rather than a similarity measure, the inverse of DTW distance is used to calculate the DTW similarity as follows:

$$DTW_{sim}(T_i, T_j) = \frac{1}{DTW(T_i, T_j)}$$

However, [33] has shown that these warping-based similarity models have other kinds of robustness issues as well. As stated in the introduction, these models consider the spatial proximity to align points. This makes them error-prone in certain cases which we explain in Section 4.

### 2.3. Advanced trajectory similarity models

To overcome these robustness issues, the authors of [34] have proposed an anchor-based model which aligns the trajectories into fixed points called anchor points. They also have employed advanced machine learning techniques together with historical trajectory data to transform the original trajectories into ones with a unified sampling rate. Hence, the similarity models mentioned previously can be employed without explicitly dealing with the sampling rate problem. However, the approach uses hidden Markov models in which every state only depends on the previous one. But this is not the case in real-world scenarios. Moreover, this model suffers from time-complexity issues even worse than the previous ones. This also is the case for other proposals such as EDwP, to tackle the problem of data with low sampling rates. EDwP is threshold-free, to dynamically interpolate the trajectories, by using the idea of projections.

There exist other distance metrics in the area of trajectory similarity which solely rely on the shape of trajectories. This includes the Frechet distance (FD) [35], symmetrized segment-path distance (SSPD) [36] and Hausdorff distance (HD). However, since in MODs the direction of the movement is important as well (rather than only the shape of the trajectory), these shape-based models are not the focus of this paper.

In this paper, we work with GPS data points sampled at a high rate (i.e., location updates every 15 s) using well-known trajectory datasets [29–31] and the popular advanced similarity model LCSS as the state-of-the-art. This is in line with previous work [14]. We now review LCSS and how it calculates similarity values for trajectories.

*LCSS Fundamentals*. LCSS is a variation of the edit distance (ED) which, in contrast to the original ED algorithm, allows some points of two trajectories to remain unmatched. Two parameters control the matching of the points between two trajectories, $\epsilon$ and $\delta$. $\epsilon$ controls the matching threshold, i.e., how far two points can be away from each other in space for the algorithm to match them. $\delta$ determines how far one can go forward in time in order to find a matching for a point. This parameter usually is defined as a percentage of the length of the trajectory in time. One defines $Head(T) = H(T) = (p_1, p_2, \ldots, p_{n-1})$ as the first $n-1$ points in trajectory $T(p_1, p_2, \ldots, p_n)$. The LCSS operator now is as follows:

$$LCSS_{\epsilon,\delta}(T_i, T_j) = \begin{cases} 0 \quad \text{if } T_i \text{ or } T_j \text{ is empty} \\ 1 + LCSS_{\epsilon,\delta}(H(T_i), H(T_j)), \\ \text{if } |p^i_{:,n} - p^j_{:,m}| < \epsilon, |m-n| < \delta \\ \max(LCSS_{\epsilon,\delta}(T_i, H(T_j)), \\ LCSS_{\epsilon,\delta}(H(T_i), T_j)) \quad \text{otherwise} \end{cases}$$

Based on the LCSS operator, the authors of [9] have suggested the following formula for LCSS similarity:

$$LCSS_{sim_{\epsilon,\delta}}(T_i, T_j) = \frac{LCSS_{\epsilon,\delta}(T_i, T_j)}{\min(n, m)}$$

Here, $n$ and $m$ are the length of $T_i$ and $T_j$ respectively. For LCSS parameters (i.e. $\epsilon$ and $\delta$) we follow the guidelines of the original

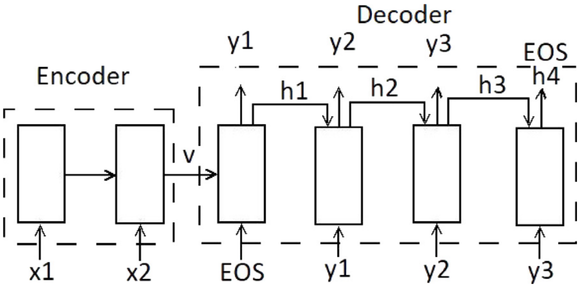

**Fig. 2.** General seq2seq model.

paper [9] regarding the default values, i.e., $\epsilon = 400$ meters and $\delta = 20\%$. This means that the points of two trajectories match if their distances in the x and y directions are less than 400 m, and they are at most one-fifth of the full trajectory length away from each other. For parameter $\epsilon$, the guidelines suggest to choose the standard deviation of distances between all pairs of points of two trajectories. We have calculated this number for 1000 trajectory pairs in the Porto dataset and set $\epsilon$ to the average of these numbers rounded to 100 m.

### 2.4. Trajectory embedding model

The t2vec model is similar to the conventional sequence to sequence (seq2seq) models. As introduced in [19], a seq2seq model consists of two parts: an encoder and a decoder, both of them being neural networks. As shown in Fig. 2, the encoder encodes the input sequence $x = \langle x_1, x_2, \dots, x_n \rangle$ to vector $v \in \mathbb{R}^{vdim}$, and the decoder decodes $v$ into the output sequence $y = \langle y_1, y_2, \dots, y_m \rangle$. $vdim$ is a predefined parameter of the algorithm denoting the dimensionality of the embedding space. $EOS$ is a special token which denotes the end of a sequence, and $h_t$ denotes the hidden state of the decoder at time $t$. In our case, the encoder encodes a trajectory as a vector $v$, and the decoder is vice versa.

To obtain a deep representation of the trajectories, t2vec first trains the neural networks. The training of t2vec works as follows. Having a route $R$ and a trajectory of a moving object $T$, the ultimate goal of the training is to maximize the probability of outputting the actual route when feeding trajectory $T$ into the neural network as input. This can be intuitively written as follows (we explicitly describe the loss function later in this section.):

$$Maximize\ P(R|T)$$

However, since a route consists of an infinite number of points, maximizing this probability is not feasible. In contrast, if we have a good representative of a route (i.e., a trajectory sampled at a high rate), one can train the model, with the help of noise contrastive estimation, as follows. Suppose that $T_b$ is a trajectory sampled at a high rate and $T_a$ is one with a low rate. The training strategy can be modified, to maximize the following conditional probability:

$$Maximize\ P(T_b|T_a)$$

Having a trajectory $T_b$ sampled at a high rate, by injecting two types of noise, namely downsampling ($ds$) and distortion ($dist$), we generate corresponding low-quality trajectories. The procedure is as follows: We have two sets of noise rates, one for downsampling ($S_{ds}$), and the other for distortion ($S_{dist}$). For every combination of these two sets, say $r_1 \in S_{ds}$ and $r_2 \in S_{dist}$, we do the following. We drop each GPS point of $T_b$ with the downsampling probability $r_1$ and we keep the point with probability $1-r_1$. We distort each remaining point with a distortion probability $r_2$ and do not distort it otherwise. Distortion means that we inject Gaussian noise in the $x$- and the $y$-dimension of the GPS points according to the following formula:

$$x_{new} = x_{old} + C \cdot d_x,\ d_x \sim \mathcal{N}(0, 1)$$
$$y_{new} = y_{old} + C \cdot d_y,\ d_y \sim \mathcal{N}(0, 1)$$

Here, $(x_{new}, y_{new})$ is the distorted position of the point $(x_{old}, y_{old})$. $C$ is a constant which is the magnitude of the distortion, and $d_x$ and $d_y$ are two standard normal distributions. Therefore, if $|S_{ds}| = n$ and $|S_{dist}| = m$, for each trajectory $T_b$ we have $(n * m)$ low-quality trajectories $T_{(a,1)}, \dots, T_{(a,n*m)}$ which we can use for training. Then the objective is to maximize the joint conditional probability:

$$Maximize \prod_{i=1}^{N} \prod_{j=1}^{K} P(T_b^i | T_{(a,j)}^i)$$

Here $N$ is the size of the training set and $K = n * m$.

As neural networks work with finite inputs and outputs, one has to transform GPS points, which have an infinite domain, to discrete tokens by gridding the Euclidean space. This means that we divide the city into grid cells with the help of the cell size ($cs$) parameter of the model. Since the number of cells usually is large, and hence some cells contain only a few GPS locations, one can limit the cells used to the ones which contain a predefined minimum number of GPS locations. We set this threshold with the minimum cell frequency ($mcf$) parameter. When the cells used (i.e., the vocabulary) are identified, we give the sequence of cells representing each trajectory as input to the seq2seq model. In order to update the weights of the neural networks, i.e., train the model, we need a loss function which should be minimized. In conventional seq2seq models a baseline loss function is to minimize the negative log-likelihood ($NLL$) product as follows:

$$L_1 = -log \prod_{t=1}^{m} P(y_t | y_{1,\dots,t-1}, x)$$

For each input trajectory sequence $x$ and corresponding output trajectory $y = \langle y_1, \dots, y_m \rangle$ we update the weights of the neural network based on this minimum. However, $NLL$ does not take into account the spatial proximity of the cells, i.e., the loss is the same if the corresponding cells are neighbors or they are on the other side of the map, which is obviously not reasonable. To overcome this issue, we can use a weighted loss function in which the weight of each cell is reversely proportional to the distance from its corresponding target cell:

$$L_2 = -\sum_{t=1}^{|y|} \sum_{u \in V} w_{uy_t} \log \frac{\exp(W_u^T h_t)}{\sum_{u \in V} \exp(W_u^T h_t)},$$
$$w_{uy_t} = \frac{\exp(-\|u - y_t\|/\theta)}{\sum_{v \in V} \exp(-\|v - y_t\|/\theta)}$$

where $w_{uy_t}$ is the weight of cell $u$ when decoding target $y_t$. $W^T$ is the matrix which projects $h_t$ into the vocabulary space and $W_u^T$ is the $u$th row of this matrix. Operator $\|.\|$ denotes the Euclidean distance between the centroid of the corresponding cells and $V$ is the set of all available vocabularies. The parameter $\theta > 0$ is used for scaling. If $\theta \rightarrow 0$ then the loss function will be equal to $NLL$. The bigger $\theta$ is, the heavier the loss function penalizes distance error. In order to decode each $y_t$, we need to summarize over the whole vocabulary twice which makes this loss function very inefficient. To overcome this, we are using the following approximate loss function which limits the number of used cells in the sum operation to a fixed number $K$, i.e., only the $K$ nearest neighbors of $y_t$ will be considered in the summarization:

$$L_3 = -\sum_{t=1}^{|y|} \sum_{u \in N_k(y_t)} w_{uy_t}(W_u^T h_t - \sum_{v \in NO} \exp(W_v^T h_t)),$$

$$w_{uy_t} = \frac{\exp(-\|u - y_t\|/\theta)}{\sum_{v \in N_k(y_t)} \exp(-\|v - y_t\|/\theta)}$$
$$NO = N_k(y_t) \cup O(y_t)$$

$N_k(y_t)$ is the set of $K$-nearest-neighbors of $y_t$ and $O(y_t)$ is a randomly sampled data from $V - N_k(y_t)$ containing 500 cells.

So far in the neural network, the input nodes have represented the grid cells of the vocabulary. This is not a suitable choice because such we ignore the spatial proximity among cells, hence it takes extra time to train the neural network. Neural networks allow to use any number of input nodes, hence we can extend the cell representations to multiple nodes for each cell, which lets the neural network to be better optimized. For example, representing each cell with its centroid solves the spatial proximity issue, but restricts the cell representation to two-dimensional vectors, i.e., two input nodes in the neural network. Using the idea of the skip-gram [37] model we create the context $c(u)$ for each cell $u \in V$. This means we sample cells which are usually near to $u$ in trajectory sequences. For $u' \in N_k(u)$ the probability of being sampled is:

$$p(u') = \frac{\exp(-\|u - u'\|/\theta)}{\sum_{v \in N_k(y_t)} \exp(-\|u - v\|/\theta)}$$

Then, using skip-grams with the negative sampling algorithm [20], we obtain a vector representation for each cell $u$ based on its context $c(u)$ by the following formula:

$$Maximize \sum_{u \in V} \log p(c(u)|g(u))$$

$g(u) \in \mathbb{R}^{cdim}$ is the representation of cell $u$, where $cdim$ is a predefined parameter denoting the dimensionality of the cell representation. Further discussion on cell representation algorithms can be found in [14]. Finally, after training the model with appropriate parameter settings, we have an encoder, which maps trajectories into vectors $v \in \mathbb{R}^n$ with the property that similar trajectories have close corresponding vectors. Summing up, the encoder is a function which embeds every trajectory of the MOD into a $vdim$ dimensional vector space, based on different parameters:

$$F(T, vdim, cdim, cs, mcf, S_{ds}, S_{dist}, C) \in \mathbb{R}^{|T| \cdot vdim}$$

### 2.5. Related work in embedding models

As mentioned earlier, we are not aware of any prior work on trajectory embedding models which investigates the semantics of similarity values. There is similar work such as [28] for word-embedding models. However, trajectory and sequence-to-sequence models are different for the following reasons:

- In word embedding models, words are the building blocks. In t2vec, the building blocks are grid cells, i.e., GPS positions mapped to cells. A sequence of words is a sentence, and a sequence of grid cells is a trajectory. Hence t2vec rather has analogies with a sentence-embedding model rather than with a word embedding model in NLP. On the other hand, as explained in the previous section, t2vec embeds the grid cells as well, which would correspond to a word embedding model.
- There is a fixed and finite number of words per language. Hence, words are common among different datasets, i.e., corpora. In contrast, the grid cells are not common among different datasets, so a t2vec model is only valid for a trajectory dataset on which it is trained.
- In word embedding models, word combinations (words found within a window of size n in a text) are only weakly correlated. In t2vec in turn, cell combinations are strongly correlated. This is because one can only move from one cell to a neighboring cell.

In the area of moving objects, there are other works which have applied embedding models for applications other than similarity search. These include human mobility identification [38], hierarchical reinforcement learning [39] and clustering [40]. All these approaches show the usefulness of embedding models in trajectory databases. But none of them deals with trajectory similarity, which is the focus of this paper.

## 3. t2vec parameter investigation

To understand the semantics of similarity values in t2vec and to evaluate the robustness of t2vec in this section, we investigate how the different parameters affect the similarity values among trajectories. First, we state two hypotheses regarding the similarity-value distribution of t2vec. Second, we present our experimental setup, introducing the dataset used and explaining how we evaluate the similarity distributions. Then we evaluate each parameter one-by-one in separate subsections.

### 3.1. Investigation objectives

The first contribution of this paper is to help to understand the semantics of the similarity values in the embedding space of t2vec. To this end, we evaluate how different parameter values of the trajectory embedding model affect the similarities. We will see that similarity values in t2vec models can differ significantly when trained with different parameters.

**Hypothesis 1.** The semantics of similarity values can be very different for t2vec models trained with different parameter settings.

We plan to confirm this hypothesis by showing that the similarity value distributions of models trained with different parameter settings have different statistical characteristics (mean values, maximal similarity values, etc.) which make general statements on the semantics of similarity values meaningless. We present two intuitive examples of such semantic differences between models in the following.

**Example 1.** Think of two models trained with different parameter settings, Model *A* with an average similarity between two trajectories of 0.0 and Model *B* with an average of 0.5. This means that the similarity value is negative for roughly half of the pairs in Model *A* and for hardly any pair in Model *B*. If one now assumed that a negative similarity value implied dissimilarity between the pair of trajectories, this assumption would have a highly different meaning for the two models.

**Example 2.** Again think of two models trained with different parameter settings. The highest similarity score of a trajectory pair is 0.8 in Model *A* and 0.5 in Model *B*. Saying that a pair with a similarity above 0.7 is definitively similar could be meaningful in Model *A*, but makes no sense in *B*. This is because there is not even a single pair with such a similarity value.

Although the similarity-value distributions of the models trained with different parameter settings can significantly differ in certain characteristics, we hypothesize that t2vec is robust to parameterization. This means that the distributions are all similar in shape, with only their means and standard deviations depending on the parameters. This leads to our second hypothesis:

**Table 1**
Statistics of the PORTO dataset used for training.

| # Points | # Trajectories | Mean | Median |
|---|---|---|---|
| 60,231,921 | 1,000,000 | 60.21 | 50 |

**Hypothesis 2.** While the parameters do influence the similarity-value distributions of the t2vec model, these distributions are almost identical when normalized.

We plan to confirm the hypothesis as follows. First, we standardize all distributions, so that they have 0 mean and 1 standard deviation. We then randomly draw 1000 values from all distributions and compare the samples pairwise by means of the two-sample Kolmogorov–Smirnov (K–S) test [41] with 99% confidence. This test checks if two samples are drawn from the same distribution. We will see that the K–S test cannot distinguish between the distributions, so they are very similar. Our main contribution in this section is that we conduct the evaluation systematically for all the parameters already introduced.

### 3.2. Experimental analysis

We now introduce the dataset used, details of the evaluation procedure and the hardware specification.

#### 3.2.1. Dataset description

We work with the Porto dataset, which is well known in the area of MODs [29]. This dataset is collected in the city of Porto for 19 months and contains more than 1,700,000 trajectories. In line with previous work [14], we remove all trajectories whose length is less than 30.

Table 1 shows statistics of the remaining dataset. From this remaining dataset, we choose the first one million trajectories as our training corpus.

#### 3.2.2. Experimental setup

As we will see in this section, t2vec highly depends on different parameters. To quantify the effects of the different parameters separately, we use a fixed default setting of the parameter values, and for each parameter, we evaluate how the model changes when only this parameter changes. The default parameter settings recommended in [14] are the following: $vdim = 256$, $cdim = 256$, $cs = 100$ m, $mcf = 50$, $S_{ds} = [0, 0.2, 0.4, 0.5, 0.6]$, $S_{dist} = [0, 0.2, 0.4, 0.6]$, $C = 50$ m, and using 80% of the trajectories from the Porto dataset as training data.

As explained in Section 2.4, we train the embedding models with two types of noise, downsampling noise, and distortion noise. We downsample the points of each trajectory $T$ with rates $r_1 \in S_{ds}$, and for each downsampled trajectory, we apply distortion noise with rates $r_2 \in S_{dist}$. So we have 20 noisy trajectories in the training set for each trajectory $T$. 50 m is the default value for the constant part C of the distortion noise parameter. When dividing Porto into grids, the default cell size is 100 m, and the minimum frequency for a cell to be considered in the vocabulary is 50. This means that cells with fewer than 50 GPS locations are not considered when training the model. Both the dimensionality of the embedding space and the cell representation is 256 by default.

The original paper uses Euclidean distance to quantify the similarity of trajectory vectors. In this paper, we use the cosine distance, for several reasons. First, the Euclidean distance is not upper-bounded and by definition heavily depends on dimensionality. This makes the comparison of similarity values between models ambiguous, for example when trained with different dimensionalities of the embedding space. This is not the case for the cosine distance, as its values are in the range $[-1,+1]$ for any space. Second, if we normalize the vectors, it is computationally cheaper to calculate the cosine distance between two vectors than the Euclidean distance. Given millions of trajectories, even a slight improvement in scalability saves a significant amount of time.

To use another similarity model instead of the one recommended by the authors we have compared the similarity values between the two models. It turns out that the similarity values are more than 99% correlated, and the biggest difference for any trajectory pair between the two models is less than 0.02. So which one we use does not make a difference in qualitative terms.

In the remainder of this section, we present the results for each parameter in the same way. For each parameter, there will be two figures. The first figure shows the similarity value distributions of the models. For these plots, we randomly select 10 000 trajectories from the full training set and calculate the similarity values of each pair. To obtain the frequency of different similarity values, we group the similarity values in 0.01 intervals in the range $[-1, +1]$, and we count the number of similarity values belonging to each group. In the figures, the $x$-axis shows the groups of similarity values and the $y$-axis the relative frequency.

The second figures are the results for k-nearest neighbor (KNN) queries. We again randomly select 10 000 trajectories $\{T_1, T_2, \ldots, T_{10000}\}$ from the dataset. For each trajectory $T_i$, we calculate the 1000 nearest neighbors $\{T_{i,1}, T_{i,2}, \ldots, T_{i,1000}\}$, with their respective similarity values $\{s_{i,1}, s_{i,2}, \ldots, s_{i,1000}\}$. i.e. $s_{i,j}$ is the similarity of $T_i$ and $T_{i,j}$. By definition, for any $0 < j < k \leq 1000$, $s_{i,j} > s_{i,k}$. We then calculate the average similarity value for every set of KNN, dubbed $avg\_sim(k) = average(s_{:,k})$. We plot the results with different values of $k$ on the $x$-axis and $avg\_sim(k)$ on the $y$-axis.

Since we group the similarity values for the first plots, and $k$ is an integer number in the second plots, we would have discrete plots, but for better visibility we connect consecutive values.

At this point, we are not trying to answer why different parameters affect the similarity values as they do; we are investigating how they affect the values. This means that we are not making qualitative statements. We are not making any statement that any model is better or worse than another one, but only to which extent they are different in their similarity semantics, and how robust the model generally is.

#### 3.2.3. System specification

Our code is written in Python and Julia. We use a modern machine with 16 CPU cores (2.4 GHz), 132 GB RAM, Nvidia GeForce GTX 1080 GPU (1.6 GHz, 8.23TFLOPS).

### 3.3. Training set size

We now evaluate how the size of the training set affects the similarity values and KNN query results in t2vec. We compare five models which are trained with differently sized subsets of the full trajectory dataset. Sampling is performed by randomly retaining different percentages of the full dataset. Then the corresponding training set is built by downsampling noise and distortion noise. To capture the effect of solely the training-set size on the models, we keep all other parameters fixed, using the default values.

Fig. 3a shows that the less training data we use, the narrower and more right-centric the distributions are, i.e., they have a smaller standard deviation and a higher mean. This means that, as the training-set size decreases, the vectors tend to be denser in one part of the embedding space. On the other hand, when using more trajectories for training, their vectors tend to fill the entire space. After standardizing the distributions to check whether they have the same fundamental shape, the K–S tests confirm that this

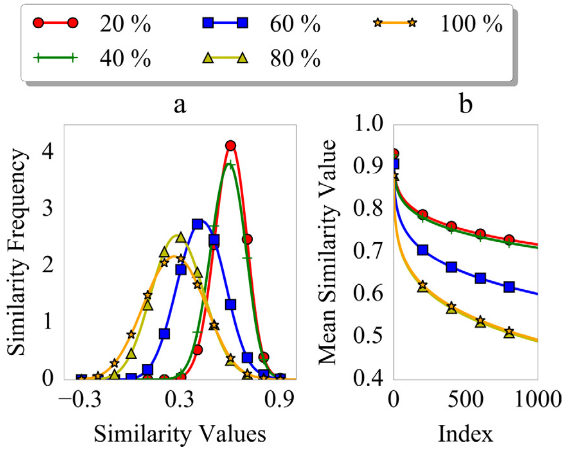

**Fig. 3.** Training set size similarity value distributions.

is indeed the case. Every $p$-value is higher than 0.01. This means that the statistical test cannot reject the null hypothesis which assumes that two sample sets come from the same underlying distributions. So we can deem the distributions very similar.

Regarding the KNN queries, Fig. 3b shows that models trained on smaller training sets generally have higher similarity values. This follows from the generally higher similarity values which we have seen in the first figure. Both the two models trained on the largest (80% and 100%) and the smallest (20% and 40%) training sets are almost identical in their KNN similarity values.

*Result interpretation.* We conclude that our hypotheses hold for the size of the training set, as the similarity-value distributions of the models are clearly different, but almost identical when normalized. This means that this parameter changes the semantics of the similarity values for t2vec significantly. However, the model is generally robust to the size of the training set.

### 3.4. Dimensionality of the embedding space

When measuring the similarity of the trajectory vectors, the number of dimensions they are embedded in is a parameter which should strongly influence the similarity values. In this section, we train the trajectory embedding model with hidden layers of different size, i.e., dimensionalities of the embedding space. Again, all other parameters of the model have the default values.

Fig. 4a shows that the lower the dimensionalities of the models, the narrower are their similarity distributions. Similarly to the evaluation of the size of the training set, the average similarity value increases with smaller dimensionality as well. In contrast to the visibly different distributions, we again show that the distributions are fundamentally similar, as the K–S test cannot distinguish the standardized distributions, with 99% confidence.

As shown in Fig. 4b, the higher the dimensionality of the embedding space, the lower are the similarity values. We have expected this, as vector spaces with lower dimensionality are denser when filled with the same number of trajectory vectors than those with higher dimensionality. This leads to closer trajectories and higher similarity values for the KNN queries.

*Result interpretation.* The experimental results on the dimensionality of the embedding space parameter confirm our hypotheses. The models are very different in their similarity distributions; however, the results of the statistical tests show that they are fundamentally very similar. Hence, the model is robust regarding this parameter as well.

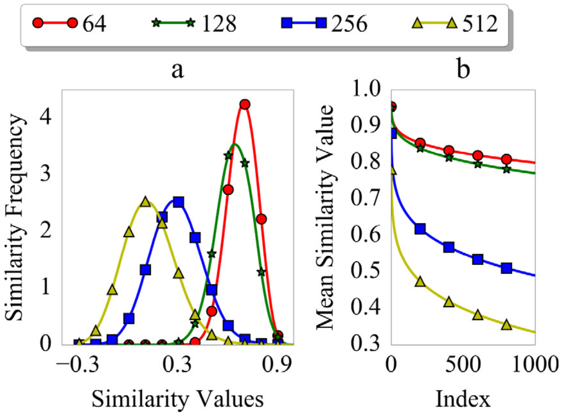

**Fig. 4.** Dimension size similarity value distributions.

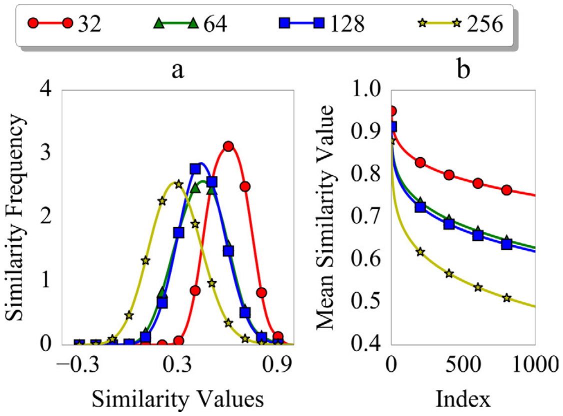

**Fig. 5.** Dimensionality of cell representation similarity value distributions.

### 3.5. Dimensionality of cell representation

As explained in Section 2.4, not only the trajectories but the cells as well are embedded into a vector space for better representation properties. We now investigate how the dimensionality of the cell representation affects the similarity values of the model.

Fig. 5a,b show very similar patterns as for the previous two parameters. Similarly, the average similarity value increases and the standard deviation decreases with smaller cell dimensionality. Again, the normalized distributions are very similar, as all p-values in the K–S tests are way above 0.01.

*Result interpretation.* The cell dimensionality parameter affects the similarity value distributions very similarly as the embedding dimensionality parameter. An exception is that in this case, the distributions corresponding to two middle parameter values are very similar, in contrast to the previous case, where they were more similar to the distributions corresponding to the extreme parameter values. Nonetheless, the models are very different in their similarity distributions, but very similar to each other when normalized. This confirms both hypotheses.

### 3.6. Grid cell size

When dividing the region into grid cells, we can choose different cell sizes according to our needs. The size of the grid cell directly influences the vocabulary size; the smaller the grids are, the larger the vocabulary tends to be. However, when the grids are very small, and the cell minimum frequency parameter is high, the vocabulary may be small as well. Larger vocabulary size

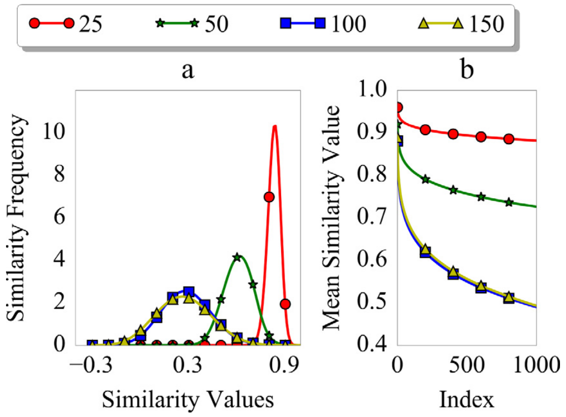

**Fig. 6.** Cell size similarity value distributions.

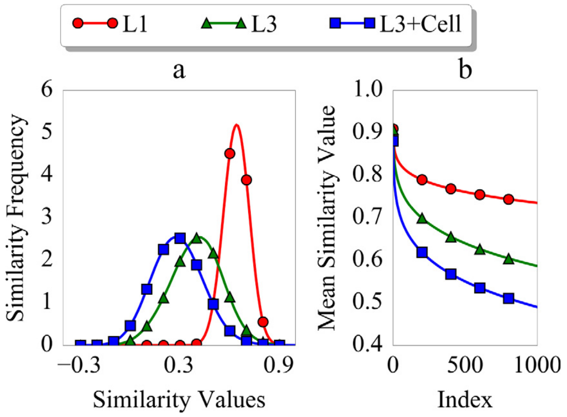

**Fig. 7.** Loss function similarity value distributions.

needs more training time. So our investigation of this parameter could help to understand the performance-accuracy tradeoff of the trajectory representations.

Fig. 6a shows the similarity-value distribution of the models trained with different cell-size parameters. It shows that, when the cells are small, the distribution has high average similarity and small standard deviation. This means that the vectors are dense in one part of the vector space. When the cell size, in turn, is high, the vectors fill a larger part of the space. Both Fig. 6a, b show that the distributions, as well as the KNN query values, change only slightly above the grid cell size of 100 m.

We test the normalized similarity distributions pairwise with the K–S test. Every $p$-value again is above 0.01. This means that the models are very similar.

*Result interpretation.* The cell size parameter affects the similarity-value distributions the most. However, this is only true below a certain value, which is roughly 100 m. Above this value, the distribution barely changes.

### 3.7. *Effects of loss function*

We evaluate the effect of the loss function in this section. Note that the original loss function without speed-up, introduced in Section 2.4, is not covered in our experiments since it does not converge in reasonable time on training sets of our size.

Fig. 7a,b shows that the models trained with different loss functions highly differ in their similarity values. The conventional $L_1$ approach embeds the trajectories much closer to each other, as we can see from its spiky distribution with high average similarity value. The two new approaches are quite similar in their distributions; however, the $L_3$ tends to have higher similarity values. The pairwise K–S tests confirm that the distributions are indeed very similar when normalized. When comparing the $L_1$ model with the other two however, the p-values are only slightly above 0.01 (0.018 and 0.023, respectively). So its distribution is somewhat different from the other two.

*Result interpretation.* The loss function affects the distributions of the similarity values very much. The approximate approaches embed the vectors more evenly in the space than the NLL based training, so the later tend to have higher similarity values.

### 3.8. *Summarizing parameter effects*

Our evaluation in this section has confirmed the two hypotheses from Section 3.1. We have shown that different parameter settings and loss functions indeed affect the value distributions of embedding models significantly; hence, the semantics of the similarity values can be very different for different models. This answers the question from the introduction of this paper as well: It is indeed possible that a similarity value of 0.5 can imply both similarity and dissimilarity, depending on the model. However, at the same time, all distributions have the same abstract bell shape. This remarkable robustness implies that the user can build a t2vec model with any parameter settings according to her needs, without having to worry about the model being distorted.

### 3.9. *Further experiments with the T-drive dataset*

The previous sections have confirmed the two hypotheses from Section 3.1 when training the model on the Porto dataset. Although this is useful for individuals working with this specific dataset, in order to generalize our findings, we aim to show that the results are independent of the training dataset. In this section, we show that our findings hold for other real-world datasets as well.

In order to do so, we use another well-known trajectory dataset to train the t2vec model on, the T-Drive dataset[2] [30,31]. This dataset contains all trajectories of 10357 taxis in Beijing during a one week period. The total number of GPS points is about 15 million, and the total distance of trajectories is about 15 million kilometers. The total number of trajectories during the week is 68 122. Out of these trajectories, we used 60 000 and 5 000 random trajectories as the training and validation set, respectively. In order to compare the results with the ones from the Porto dataset, we set the model parameters to their default values. This means that the cell size parameter is 100 m, resulting in 40 000 tokens (i.e., vocabularies). To be fair in our comparisons, we use the default parameter setting for t2vec as follows: $vdim = 256$, $cdim = 256$, $cs = 100$ m, $mcf = 50$, $S_{ds} = [0, 0.2, 0.4, 0.5, 0.6]$, $S_{dist} = [0, 0.2, 0.4, 0.6]$, $C = 50$ m, and we use 80% of the trajectories from the T-Drive dataset as training data. We conduct the same experiments with the model trained on the T-Drive dataset as in Section 3. The results are presented in Fig. 8.

Fig. 8 shows how the underlying dataset affects the similarity values of the corresponding models. The similarity values obtained from the Beijing dataset tend to be higher than the ones of the Porto dataset. However, the shapes are very similar. Applying the K–S test confirms that two distributions are almost identical when normalized. As for the KNN queries, we can see that the model trained on the Beijing data has higher average similarity

[2] Available at https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/.

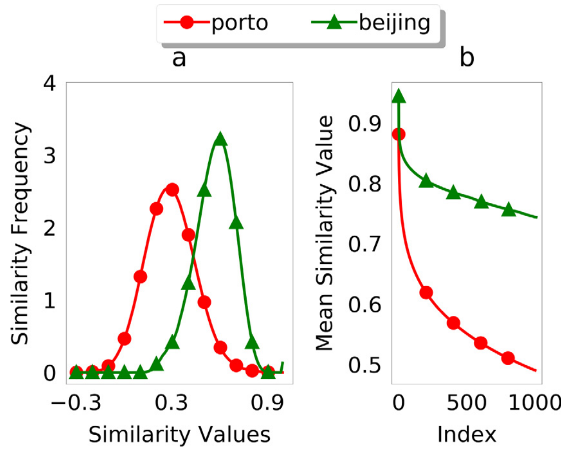

**Fig. 8.** Different datasets similarity value distributions.

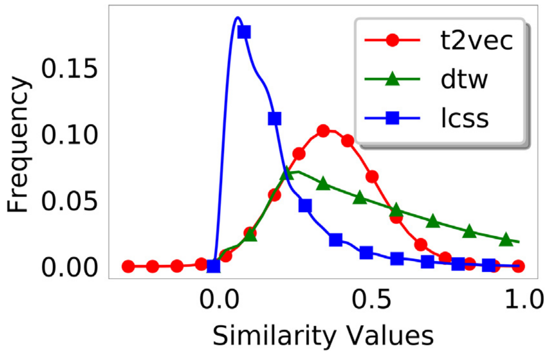

**Fig. 9.** LCSS and t2vec similarity-value distributions.

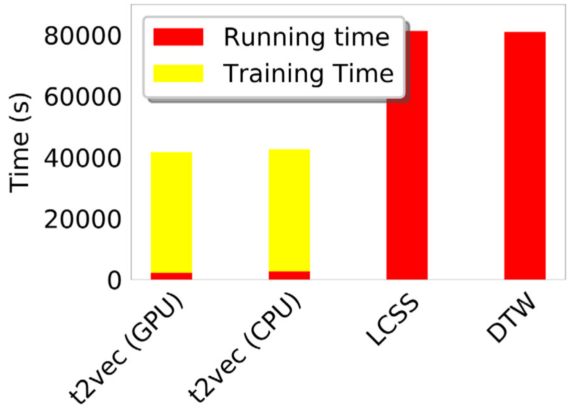

**Fig. 10.** Runtime for t2vec, LCSS and DTW models.

values. We think that these differences are consequences of the different training sizes of the two models. If we compare the model trained on the Beijing dataset to a model trained on the Porto dataset with a similar number of trajectories, the results are almost identical (see Fig. 3).

*Result interpretation.* Our results show that our hypotheses are independent of the underlying dataset, as the similarity values obtained from two different datasets are very similar. This means that the semantics of similarity in the t2vec model is robust against using different training datasets.

## 4. Comparing t2vec to classical models

In this section, we compare the new embedding-based trajectory-similarity model to classical models. To this end, we use the DTW and LCSS similarity models introduced in Sections 2.2 and 2.3. It has been shown that LCSS constantly performs better than DTW, especially when working with noisy data, and it is considered the current state-of-the-art similarity model in trajectory databases [9,14]. Hence, for qualitative comparisons, we only compare t2vec to LCSS. For general comparisons such as distributional differences and scalability, we compare t2vec to both LCSS and DTW.

It is not trivial how one can compare two similarity models since we are not aware of any existing similarity-value benchmark. Neither is there a method to quantify the goodness of a model. Hence, to learn the differences between t2vec and LCSS we do a three-fold comparison, explained in the following.

First, we compare their scalability and similarity-value distributions and compute their correlation. Second, we conduct a systematic qualitative comparison of the two models and study why the observed differences occur. Third, we cluster trajectories based on different models and evaluate the clustering results. Based on all these comparisons, we draw conclusions regarding the goodness and applicability of the models in different use cases and real-world scenarios.

### 4.1. Distributional differences

Now we compare the similarity-value distributions of t2vec, LCSS, and DTW. In Section 3, we have already evaluated how the parameters affect the t2vec model. It turns out that LCSS and DTW similarity distributions are different.

Fig. 9 shows the distributions. As a result of how LCSS calculates the similarity, most LCSS similarity values are exactly 0. This is not the case for t2vec, where the values are continuously distributed between [−1, +1]. This property lets us compare two trajectory pairs with low similarity scores in the t2vec case, but not in the LCSS case because their corresponding similarity is 0.0 with LCSS. This makes t2vec more robust in terms of the number of comparable trajectory pairs. Note that DTW does not have an upper bound by its formal definition. However, in order to make the comparisons more meaningful, we map DTW similarity values to the [0, 1] interval.

In general, higher t2vec scores should correspond to higher LCSS and DTW similarity. To quantify the similarity between the models, we calculate the Pearson correlation of the corresponding similarity values. These values are 0.381 and −0.006 for LCSS and DTW, respectively. This means that t2vec similarity is moderately correlated to LCSS, but not at all to DTW. DTW does not correlate with LCSS either (−0.016). Considering that LCSS has been proven to be a better trajectory similarity model than DTW, we conclude that DTW is just not suitable for noisy real-world data such as the Porto dataset.

### 4.2. Runtime comparison

In this section, we compare the models regarding scalability. Fig. 10 shows our runtime results. To obtain these results, we have randomly sampled 20 000 trajectories from the Porto dataset and calculated all pairwise similarity values using t2vec, LCSS, and DTW models. We calculate the similarities using both CPU and GPU for t2vec. For LCSS and DTW, because of the unequal length of the trajectories, GPU does not speed up the calculations.

The results show that t2vec calculates trajectory similarities 31–37 times faster than LCSS on average. This number can be even bigger with longer trajectories, since the runtime of LCSS is quadratic with the length, in contrast to t2vec with a linear

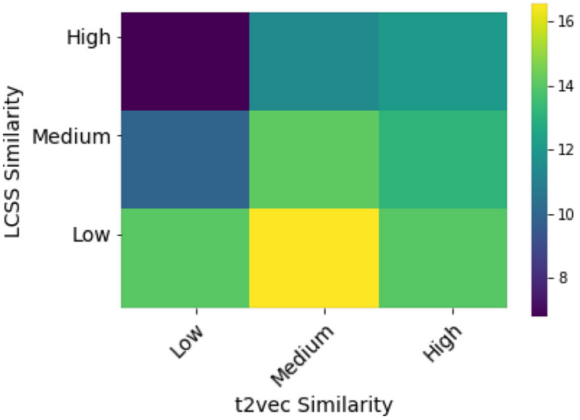

**Fig. 11.** Heatmap obtained by similarity groups: Low, Medium and High.

dependency. The same holds for the DTW model as well, since its runtime is almost identical to the one with LCSS. Using GPU it has taken 2184 s to calculate all similarity values with t2vec, with CPU 2645 s. These numbers are 79 122 and 77 986 for LCSS and DTW, respectively. However, there is an additional overhead for the t2vec model, which is to train the model. This has taken 37 341 s on average, of all trained models in Section 3. We can see that, even with this overhead, and even for such a small number of trajectories as 20 000 (almost 1% of all trajectories in the dataset), it has taken less than half the time to calculate the similarities with t2vec compared to LCSS and DTW.

The total runtimes are quadratic in the number of trajectories we want to compare. This means that the more trajectory similarities we calculate, the bigger is the runtime benefit for t2vec. For example, for 100 000 trajectories it would take about 27 times more time to calculate the similarities with the classical models. On the other hand, if we need to calculate the similarities of less than several thousand trajectories the classical models can do it faster, because of the overhead of model building.

### *4.3. Qualitative comparison*

In this section we give a systematic comparison of the LCSS and t2vec models.

In order to do so, we assign the similarity values of t2vec and LCSS models into similarity value groups "low", "medium" and "high". The "low" group contains the low similarity values ($[-1, 0.3]$ and $[0, 0.2]$ for t2vec and LCSS, respectively), the "medium" group includes the medium similarity values ($[0.3, 0.6]$, $[0.2, 0.5]$) and "high" group contains high similarity values ($[0.6, 1]$, $[0.5, 1]$) of each model. This divides the similarity space into a $3 * 3$ grid, see Fig. 11.

We count the number of trajectory pairs assigned to each of these 9 grid cells. A brighter cell color stands for a higher number of trajectory pairs. Note that we use the log scale of the numbers to color the cells.

In each of the following subsections we present one case of Fig. 11. We aim to understand why such trajectory pairs occur at such specific intervals of similarity values.

#### *4.3.1. Low LCSS, low t2vec similarity values*

In this case, both similarity values are low. It is easy to understand which kind of trajectory pair belongs to this group. As the examples in Fig. 12 show, they do not or only slightly intersect, their shape is different, and they go to different directions. Both models understand that such trajectory pairs should have low similarity scores.

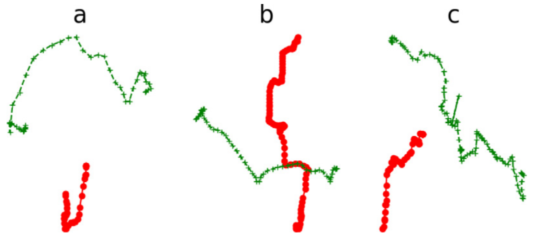

**Fig. 12.** Low LCSS, low t2vec examples.

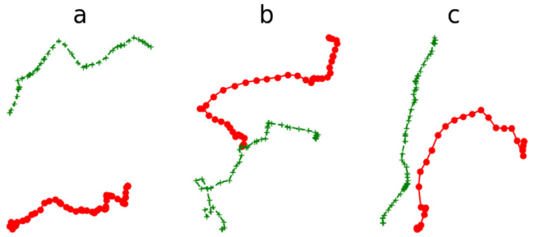

**Fig. 13.** Low LCSS, medium t2vec examples.

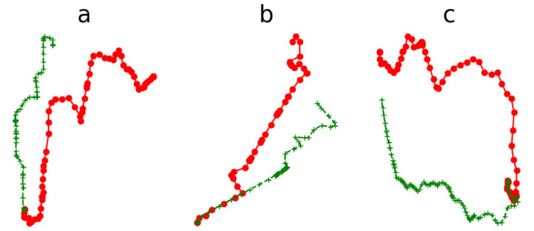

**Fig. 14.** Low LCSS, high t2vec examples.

#### *4.3.2. Low LCSS, medium t2vec similarity values*

The case when the LCSS score is low, but the t2vec score is in the medium range is much more interesting. We can see in Fig. 13 that again the two trajectories intersect only slightly if at all, but tend to *look* similar. With this, we mean that either their shape or their vector from start to endpoint is similar. This means that, in contrast to LCSS, the t2vec model is able to capture such information, even when the respective points are far from each other.

#### *4.3.3. Low LCSS, high t2vec similarity values*

When the LCSS score is low, and the t2vec score is in its highest interval, the examples show a specific pattern, see Fig. 14. These trajectory pairs have small intersections at one end of the trajectories, they look similar, and their other end is close to each other. This means that, if the t2vec score is high, *both* ends of the trajectories are close to each other.

#### *4.3.4. Medium LCSS, low t2vec similarity values*

In the case of medium-range LCSS values and low t2vec values the examples, such as the ones on Fig. 15, show a quite specific pattern. These are trajectory pairs which intersect on a long sub-trajectory but have substantially distinct sub-trajectories as well. In the meantime, their shape and start and endpoints are very different.

#### *4.3.5. Medium LCSS, medium t2vec similarity values*

In the middle of the grid, the same trend appears that we have seen in the section on low LCSS and medium t2vec scores, but this time with a substantial intersection between the two trajectories. This means that, as shown in Fig. 16, the trajectories not only

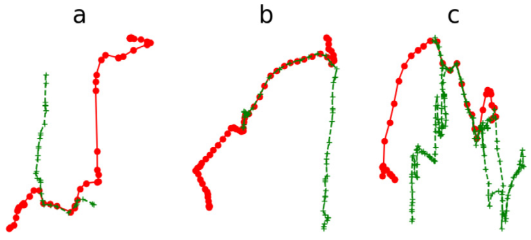

**Fig. 15.** Medium LCSS, low t2vec examples.

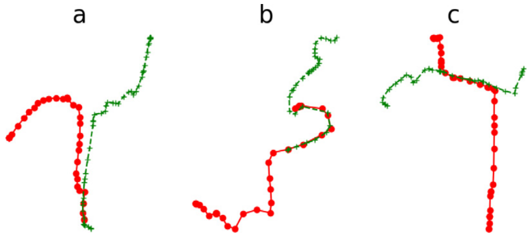

**Fig. 16.** Medium LCSS, medium t2vec examples.

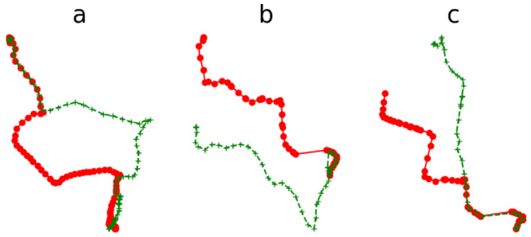

**Fig. 17.** Medium LCSS, high t2vec examples.

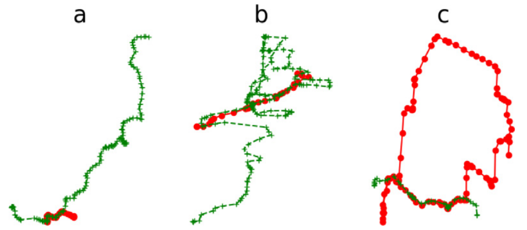

**Fig. 18.** High LCSS, low t2vec examples.

intersect but they look similar as well. However, their start and endpoint are not necessarily close to each other.

### 4.3.6. *Medium LCSS, high t2vec similarity values*

The description for the previous case holds for this case as well, with one difference. Here, not only the shapes are similar but the start and endpoint of the trajectories are both close to each other, if not the same. Again, as in the previous high t2vec score section, the intersections tend to be at the beginning or at the end of the trajectories, or at both ends. Fig. 17 features examples of this case.

### 4.3.7. *High LCSS, low t2vec similarity values*

In this case, when LCSS values are high and t2vec values are low, the trajectories look very differently from each other, and typically one trajectory almost fully contains the other one. See Fig. 18 for examples.

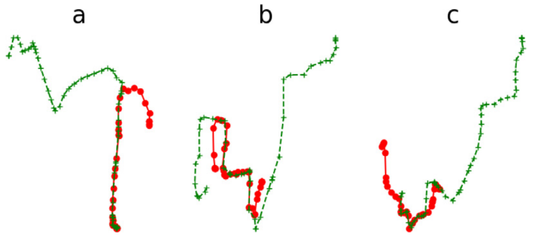

**Fig. 19.** High LCSS, medium t2vec examples.

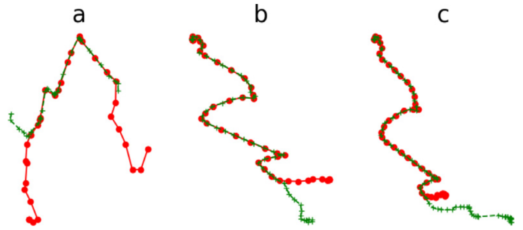

**Fig. 20.** High LCSS, high t2vec examples.

### 4.3.8. *High LCSS, medium t2vec similarity values*

Medium t2vec similarity values together with high LCSS values results in trajectory pairs with substantial intersection and similar looks. One path usually almost fully contains the other one, but in the meantime, their start and endpoint are closer, and their shape is much more similar than in the previous section. Examples for this case are shown in Fig. 19.

### 4.3.9. *High LCSS, high t2vec similarity values*

As we reach the other end of the spectrum for both models, we see a group of trajectory pairs, shown in Fig. 20, which are almost identical. This means that they have substantial common sub-trajectories, and their shape is very similar. They tend to have a slightly different start or endpoint, but these points are close to each other.

### 4.3.10. *Summarizing the qualitative differences*

Based on the previous evaluations of each case, we now give a summary of the differences of the LCSS and t2vec trajectory similarity models.

**Interpreting the T2vec model.** To start, we give an intuition how the t2vec model works. As we look through the cases one by one with fixed LCSS values and increasing t2vec values we find the same pattern: When the t2vec value is low, the trajectories have different directions, their start and endpoint are not close to each other, and their shape is different. In one word, they *look* differently. When the t2vec value is in the medium range usually either the start or endpoint of the trajectories are close to each other, or at least the two trajectories have the same general direction, and their shape is similar. However, at this point, the similarity of their shape usually means that the trajectories are mirrored or shifted versions of each other. Finally, when the t2vec value is high, this implies that both ends of the trajectories are close to each other, and they have similar shapes in between as well. The closeness of the two ends of the trajectories usually implies similarity in shape as well, but not in every case. In summary, we can say that the t2vec similarity captures a combination of both similarities in shape and similarity of the starting points.

**Interpreting the LCSS model.** Now we explain the specifics of the LCSS model that are responsible for the effects just described. In particular, we look at the cases with fixed t2vec but different LCSS values. One can see that the examples are similar between the cases, the difference is just that when the LCSS

**Table 2**
Similarity models and applications.

| LCSS | t2vec | Overlap | Shape | Direction | Distance |
|---|---|---|---|---|---|
| Low | Low | ✗ | ✗ | ✗ | ✗ |
| Low | Medium | ✗ | ✓ | ✗/✓ | ✗ |
| Low | High | ✗ | ✓ | ✓ | ✗/✓ |
| Medium | Low | ✗/✓ | ✗ | ✗ | ✗/✓ |
| Medium | Medium | ✗/✓ | ✓ | ✗/✓ | ✗/✓ |
| Medium | High | ✗/✓ | ✓ | ✓ | ✗/✓ |
| High | Low | ✗/✓ | ✗ | ✗ | ✗ |
| High | Medium | ✓ | ✓ | ✗/✓ | ✓ |
| High | High | ✓ | ✓ | ✓ | ✓ |

value increases, the intersection of the trajectory pairs becomes larger as well. However, when the t2vec value is small, one of the trajectories is a sub-trajectory of the other one. Namely, if the two trajectories have long common sub-trajectories, they inevitably tend to look more similar, hence the only way for the t2vec value to remain small is when one is much longer than the other. In summary, we can say that LCSS similarity models the extent of intersection between the trajectories. However, it is more than an algorithm that only compares the lengths of sub-trajectories. This is because of the variable grid cell settings, which let LCSS capture the similarity between close but not identical trajectories as well.

**Combining the two similarity models.** When looking at the two models in isolation, we have seen that one cannot perfectly describe what increasing values mean in terms of similarity semantics. If we group the examples from the previous sections based on only one of the models, for example, all the examples with medium t2vec values, we can see that it is hard to generalize their attributes. However, when explaining the cases in the grid, we were able to come up with descriptions how that group of trajectory pairs generally looks like. So we recommend using both models when one wants to find trajectory pairs of a certain kind. Table 2 summarizes our findings regarding the characteristics of similarity models and what kind of semantics of similarity one can expect in each case.

We can see that *Shape* and *Direction* are the characteristics which are present when t2vec yields high similarity values. These two attributes are consistently changing with the t2vec values. This is not the case for the *Distance* attribute: Here, different LCSS values imply different behavior when increasing the t2vec value. For example, when the t2vec value is low and the LCSS is medium, the trajectories can be close to each other, but this is never the case when the LCSS is high. This means that the Distance attribute depends on both t2vec and LCSS.

In the next paragraph, we give examples of use cases and real-world applications where finding trajectories from specific parts of the grid is relevant.

**Use cases and applications.** In Section 4.3 we have presented examples of each similarity group, i.e., of each cell of the 2-dimensional grid in Fig. 11. We argue that different applications call for different semantics of similarity. For example, for some applications such as ride sharing, the length of the overlapping part of two trajectories is of high importance, hence one may look for cases with high LCSS and medium or high t2vec similarity values. On the other hand for other scenarios, it suffices for the users only to look for trajectories that are spatially close rather than fully overlapping. This is the case, for example, in downstream applications such as monitoring big events [42], monitoring crowd behavior [42] or finding hurricane movement patterns [43]. In such cases, having a close source or destination (or both) is enough for the user to deem two trajectories similar. In this case, one may look for scenarios where the t2vec similarity value is high, whether this is the case for the LCSS or not. In other domains, such as when the problem is to find movement patterns in a city, i.e., trajectories that move in the same direction, but in different spatial locations, one may look for directional similarities without considering distance or overlap. In some "non-movement" applications such as finding similar handwritings, the only concern is whether the two handwritten trajectories are of the same shape. There are various other applications in which the aim is to find shape-based patterns. For example, animals have different search patterns [44] that can be used to determine whether they are random or planned. Other applications include determining similar fishing activities from trajectories of vessels [45] and sign-language recognition [46].

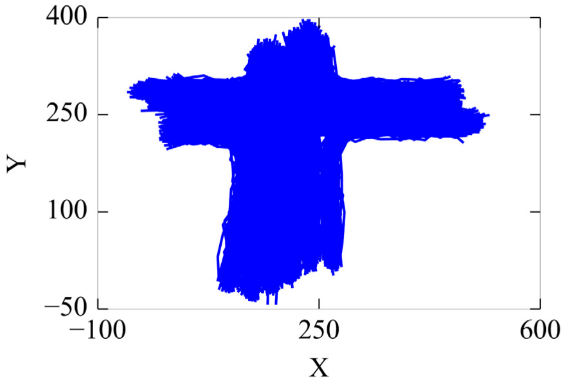

**Fig. 21.** CROSS dataset.

**Table 3**
Statistics of CROSS dataset.

| #P | #T | ML | #C[a] |
|---|---|---|---|
| 24,420 | 1900 | 13 | 19 |

[a] P = Points; T = Trajectories; ML = Mean Length; C = Clusters.

### 4.4. *Clustering*

To further check the validity of the similarity values of t2vec, we evaluate how appropriate they are for trajectory clustering. To this end, we study the differences on the CROSS [47] dataset. In contrast to the Porto dataset, the CROSS dataset is annotated; hence, one can quantify the quality of the clusters. The CROSS dataset contains the trajectories of moving objects in foursquare, as shown in Fig. 21.

It contains 1 900 trajectories which are annotated (19 clusters with 100 trajectories in each cluster) based on the lanes in which the cars come in and leave the square. The trajectories are generated from visual data (i.e., video recording), hence it uses pixels as base units of length. Statistics of the dataset are in Table 3.

*Clustering with LCSS.* It has been shown in [47–49] that spectral clustering combined with LCSS yields good results in surveillance environments. Morris et al. [47] have proposed a method to cluster the CROSS trajectories based on LCSS and spectral clustering [50]. We explain this in the following in some detail, because we use the same method in the next subsection to cluster with the t2vec model. So we can compare the effect of solely the similarity models on the clustering. The method consists of four steps:

- Deriving the LCSS similarity for all pairs of trajectories and building the similarity matrix $S = \{s_{ij}\}$ with the Gaussian kernel function

$$s_{ij} = e^{-\frac{LCSS(T_i,T_j)^2}{2\sigma^2}}$$

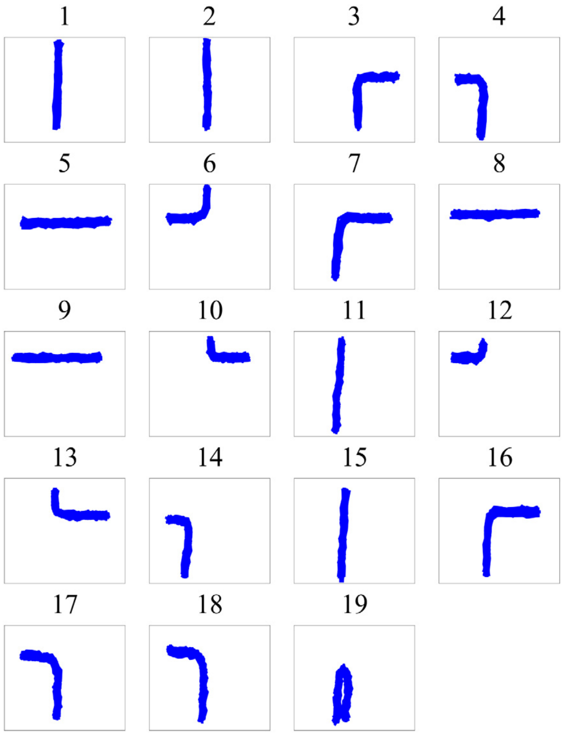

**Fig. 22.** The resulting clusters based on t2vec.

Here, $s_{ij}$ is the similarity of trajectories $T_i$ and $T_j$, and $\sigma$ is a normalization parameter to control the range of the values.

- Calculating the Laplacian matrix from the similarity matrix as follows:

  $L = I - D^{-\frac{1}{2}} S D^{\frac{1}{2}}$

  Here, $D$ is a diagonal degree matrix with the diagonal elements being the sum of the corresponding rows in $S$.
- Calculating the first $K$ eigenvectors of $L$ and building the matrix $U \in \mathbb{R}^{(N*K)}$ which contains these $K$ eigenvectors of $L$ as columns.
- Considering the rows of $U$ as representations of the trajectories, and clustering the rows using the fuzzy C-means (FCM) algorithm [51].

*T2vec vs. LCSS clustering comparison.* Since the number of clusters is not known a priori, we set the number of clusters to 30 initially for the FCM clustering in both cases. However, when validating the clusters we apply a "match and merge" procedure introduced in [47], to merge the clusters close to each other. For this, Dynamic Time Warping (DTW) [32,52] is used to find such clusters with an average distance smaller than $\epsilon$, or the number of their common points is higher than a threshold $T$. For more details, we refer to the original paper [47]. This yields 19 clusters in both cases; see Fig. 22. There are very similar clusters, such as four different vertical lines, but representing different lanes on the foursquare, hence they should not be clustered together.

Table 4 contains qualitative statistics regarding the two resulting clusterings. We see that t2vec yields better clusters than the LCSS model. This result is even more significant since we have used a clustering algorithm which has originally been developed for the LCSS distance.

**Table 4**
Clustering validation.

| | Hom. | Comp. | V | AMI | ARI[a] |
|---|---|---|---|---|---|
| t2vec | 0.981 | 0.982 | 0.981 | 0.981 | 0.971 |
| LCSS | 0.961 | 0.963 | 0.962 | 0.960 | 0.940 |

[a] Hom. = Homogenity; Comp. = Completeness; V = V-Measure;AMI = Adjusted Mutual Information; ARI = Adjusted Random Index.

### 4.5. Summary

We have shown in Section 4.2 that t2vec can calculate trajectory similarity values faster than classical models, even with the model-training overhead, especially when dealing with large datasets. In Section 4.3 we have shown that t2vec and LCSS tend to capture different spirits of similarity, hence we propose to consider using a combination of the two models. In Section 4.4 we compared the clustering ability of the two models and have concluded that trajectory clusters produced by t2vec are better.

All this means that we can work with t2vec without having to worry about any qualitative bias in the future. On the other hand, the scalability of t2vec opens up new possibilities, such as clustering of large real-world trajectory datasets, which has not been possible with classical models.

## 5. Conclusions and future work

Recently, a new trajectory-embedding model, t2vec, has been proposed that allows computing trajectory similarities efficiently. Since t2vec is built on a neural network which maps the trajectories into an embedding space and computes the similarities in this vector space, it is not trivial how one should interpret the semantics of its similarity values. In this paper, we propose a methodology which lets us evaluate the meaningfulness of the deep trajectory representations in t2vec.

Our first contribution is that we evaluate the semantics of similarity values in t2vec as well as the robustness of the model to parameterization. To this end, we have studied how the distribution of similarity values evolves when changing the parameters of t2vec. As a result, the semantics of similarity values highly depend on the parameters, i.e., the same value can represent different grades of similarity in different models.

Our second contribution is to evaluate the goodness of t2vec in downstream tasks. In order to do this, we compare the embedding model to a baseline and a state-of-the-art classical similarity model. We show that the t2vec similarity model captures different semantics of similarity than the classical models, hence they are the best when used together. Moreover, we show that t2vec is qualitatively better for clustering trajectories while calculating similarity values orders of magnitudes faster than classical models.

Our final contribution has been that we make all the trained embedding models publicly available. This is one of the biggest collections of its type available on the web.

In the future, we intend to create a benchmark for both similarity and clustering of trajectories as done for texts [53], since there is no such benchmark for large datasets such as Porto. We want to investigate as well whether the trajectory-embedding models can be seen as an operator, i.e., whether it is meaningful for the learned vectors to be added or subtracted from each other.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

[1] Y. Zheng, Trajectory data mining: an overview, ACM Trans. Intell. Syst. Technol. (TIST) 6 (3) (2015) 29.

[2] C.-C. Hung, W.-C. Peng, W.-C. Lee, Clustering and aggregating clues of trajectories for mining trajectory patterns and routes, The VLDB J.—Int. J. Very Large Data Bases 24 (2) (2015) 169–192.

[3] I.V. Cadez, S. Gaffney, P. Smyth, A general probabilistic framework for clustering individuals and objects, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2000, pp. 140–149.

[4] J.-G. Lee, J. Han, K.-Y. Whang, Trajectory clustering: a partition-and-group framework, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, ACM, 2007, pp. 593–604.

[5] Z. Li, J.-G. Lee, X. Li, J. Han, Incremental clustering for trajectories, in: International Conference on Database Systems for Advanced Applications, Springer, 2010, pp. 32–46.

[6] Z. Li, J. Han, M. Ji, L.-A. Tang, Y. Yu, B. Ding, J.-G. Lee, R. Kays, Movemine: Mining moving object data for discovery of animal movement patterns, ACM Trans. Intell. Syst. Technol. (TIST) 2 (4) (2011) 37.

[7] Z. Chen, H.T. Shen, X. Zhou, Discovering Popular Routes from Trajectories, IEEE, 2011.

[8] H. Jeung, M.L. Yiu, X. Zhou, C.S. Jensen, H.T. Shen, Discovery of convoys in trajectory databases, Proc. VLDB Endow. 1 (1) (2008) 1068–1080.

[9] M. Vlachos, G. Kollios, D. Gunopulos, Discovering similar multidimensional trajectories, in: Data Engineering, 2002. Proceedings. 18th International Conference on, IEEE, 2002, pp. 673–684.

[10] L. Chen, R. Ng, On the marriage of lp-norms and edit distance, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases-Volume 30, VLDB Endowment, 2004, pp. 792–803.

[11] L. Chen, M. Özsu, V. Oria, Robust and fast similarity search for moving object trajectories, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 491–502.

[12] S. Sankararaman, P.K. Agarwal, T. Mølhave, J. Pan, A.P. Boedihardjo, Model-driven matching and segmentation of trajectories, in: Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2013, pp. 234–243.

[13] E. Frentzos, K. Gratsias, Y. Theodoridis, Index-based most similar trajectory search, in: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, IEEE, 2007, pp. 816–825.

[14] X. Li, K. Zhao, G. Cong, C.S. Jensen, W. Wei, Deep Representation Learning for Trajectory Similarity Computation.

[15] J.-G. Lee, J. Han, X. Li, H. Gonzalez, Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering, Proc. VLDB Endow. 1 (1) (2008) 1081–1094.

[16] M. Nanni, D. Pedreschi, Time-focused clustering of trajectories of moving objects, J. Intell. Inf. Syst. 27 (3) (2006) 267–289.

[17] S. Rinzivillo, D. Pedreschi, M. Nanni, F. Giannotti, N. Andrienko, G. Andrienko, Visually driven analysis of movement data by progressive clustering, Inf. Vis. 7 (3–4) (2008) 225–239.

[18] S. Ranu, P. Deepak, A.D. Telang, P. Deshpande, S. Raghavan, et al., Indexing and matching trajectories under inconsistent sampling rates, in: 2015 IEEE 31st International Conference on Data Engineering (ICDE), IEEE, 2015, pp. 999–1010.

[19] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.

[20] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[21] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (Aug) (2011) 2493–2537.

[23] A. Passos, V. Kumar, A. McCallum, Lexicon infused phrase embeddings for named entity resolution, arXiv preprint arXiv:1404.5367, 2014.

[24] B. Klein, G. Lev, G. Sadeh, L. Wolf, Fisher vectors derived from hybrid gaussian-laplacian mixture models for image annotation, arXiv preprint arXiv:1411.7399, 2014.

[25] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, J. Makhoul, Fast and robust neural network joint models for statistical machine translation, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vol. 1, 2014, pp. 1370–1380.

[26] S. Liu, N. Yang, M. Li, M. Zhou, A recursive recurrent neural network for statistical machine translation, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vol. 1, 2014, pp. 1491–1500.

[27] J. Paparrizos, M.J. Franklin, GRAIL: efficient time-series representation learning, Proc. VLDB Endow. 12 (11) (2019) 1762–1777.

[28] Á. Elekes, A. Englhardt, M. Schäler, K. Böhm, Toward meaningful notions of similarity in NLP embedding models, Int. J. Digit. Libr. (2018) 1–20.

[29] Porto Dataset, http://www.geolink.pt/ecmlpkdd2015-challenge.

[30] J. Yuan, Y. Zheng, X. Xie, G. Sun, Driving with knowledge from the physical world, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 316–324.

[31] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, Y. Huang, T-drive: driving directions based on taxi trajectories, in: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2010, pp. 99–108.

[32] D.J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: KDD Workshop, Vol. 10, Seattle, WA, 1994, pp. 359–370.

[33] H. Wang, H. Su, K. Zheng, S. Sadiq, X. Zhou, An effectiveness study on trajectory similarity measures, in: Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137, Australian Computer Society, Inc., 2013, pp. 13–22.

[34] H. Su, K. Zheng, H. Wang, J. Huang, X. Zhou, CaLibrating trajectory data for similarity-based analysis, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 833–844.

[35] H. Alt, M. Godau, Computing the Fréchet distance between two polygonal curves, Int. J. Comput. Geom. Appl. 5 (01n02) (1995) 75–91.

[36] P. Besse, B. Guillouet, J.-M. Loubes, R. François, Review and perspective for distance based trajectory clustering, arXiv preprint arXiv:1508.04904, 2015.

[37] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781, 2013.

[38] Q. Gao, F. Zhou, K. Zhang, G. Trajcevski, X. Luo, F. Zhang, Identifying human mobility via trajectory embeddings.

[39] J.D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, S. Levine, Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings, arXiv preprint arXiv:1806.02813, 2018.

[40] A. Esuli, L.M. Petry, C. Renso, V. Bogorny, Traj2User: exploiting embeddings for computing similarity of users mobile behavior, arXiv preprint arXiv:1808.00554, 2018.

[41] F.J. Massey Jr, The Kolmogorov-Smirnov test for goodness of fit, J. Amer. Statist. Assoc. 46 (253) (1951) 68–78.

[42] M. Versichele, T. Neutens, M. Delafontaine, N. Van de Weghe, The use of bluetooth for analysing spatiotemporal dynamics of human movement at mass events: A case study of the ghent festivities, Appl. Geogr. 32 (2) (2012) 208–220.

[43] S. Dodge, P. Laube, R. Weibel, Movement similarity assessment using symbolic representation of trajectories, Int. J. Geogr. Inf. Sci. 26 (9) (2012) 1563–1588.

[44] S. Focardi, P. Marcellini, P. Montanaro, Do ungulates exhibit a food density threshold? a field study of optimal foraging and movement patterns, J. Anim. Ecol. (1996) 606–620.

[45] R.A. Enguehard, R. Devillers, O. Hoeber, Geovisualization of Fishing Vessel Movement Patterns Using Hybrid Fractal/velocity Signatures, GeoViz Hamburg, 2011.

[46] Z. Zhang, K. Huang, T. Tan, et al., Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes, in: ICPR (3), Citeseer, 2006, pp. 1135–1138.

[47] B.T. Morris, M.M. Trivedi, Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach, IEEE Trans. Pattern Anal. Mach. Intell. 33 (11) (2011) 2287–2301.

[48] B. Morris, M. Trivedi, Learning trajectory patterns by clustering: Experimental studies and comparative evaluation, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 2009, pp. 312–319.

[49] Z. Zhang, K. Huang, T. Tan, Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes, in: Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, Vol. 3, IEEE, 2006, pp. 1135–1138.

[50] A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Advances in Neural Information Processing Systems, 2002, pp. 849–856.

[51] W. Pedrycz, Knowledge-Based Clustering: From Data to Information Granules, John Wiley & Sons, 2005.

[52] J. Willkomm, J. Bettinger, M. Schäler, K. Böhm, Efficient interval-focused similarity search under dynamic time warping, in: Proceedings of the 16th International Symposium on Spatial and Temporal Databases, ACM, 2019, pp. 130–139.

[53] A. Elekes, A. Englhardt, M. Schäler, K. Böhm, Resources to examine the quality of word embedding models trained on n-gram data, in: Proceedings of the 22nd Conference on Computational Natural Language Learning, 2018, pp. 423–432.