Using deep learning for trajectory classification in imbalanced dataset

Nicksson Freitas, Ticiana Silva, José Macêdo, Leopoldo Junior
{nickssonarrais, ticianalc, jose.macedo,leopoldosmj}@insightlab.ufc.br
Insight Data Science Lab
Fortaleza, Brazil

Abstract

Deep learning has gained much popularity in the past years due to GPU advancements, cloud computing improvements, and its supremacy, considering the accuracy results when trained on massive datasets. As with machine learning, deep learning models may experience low performance when handled with imbalanced datasets. In this paper, we focus on the trajectory classification problem, and we examine deep learning techniques for coping with imbalanced class data. We extend a deep learning model, called DeepeST (Deep Learning for Sub-Trajectory classification), to predict the class or label for sub-trajectories from imbalanced datasets. DeepeST is the first deep learning model for trajectory classification that provides approaches for coping with imbalanced dataset problems from the authors' knowledge. In this paper, we perform the experiments with three real datasets from LBSN (Location-Based Social Network) trajectories to identify who is the user of a sub-trajectory (similar to the Trajectory-User Linking problem). We show that DeepeST outperforms other deep learning approaches from state-of-the-art concerning the accuracy, precision, recall, and F1-score.

Introduction

The recent advances in the development of tracking and surveillance devices and the popularity of Location-Based Social Networks (LBSNs) contribute to the explosive growth of trajectory data. We strongly believe that these data provide a unique opportunity for understanding the patterns and behaviors of several moving objects, such as people, animals, transportation modes, hurricanes, among others.

In the literature, there exist several research problems for trajectory data (Zheng, 2015). One of them is the Trajectory classification that is an efficient way to analyze trajectory. In this paper, we focus on trajectory classification for imbalanced datasets.

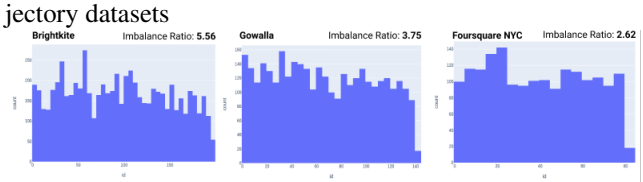
The trajectory classification problem consists of building a prediction model to classify a new trajectory in a single-class or multi-class. The model is trained and learns the patterns (or classes) from a historical labeled trajectory (or sub-trajectory) data. Trajectory classification can be applied among other applications to identify the transportation mode from a moving object trajectory (like car and bus, for instance) or to link the trajectory to its user (TUL problem)(Gao et al., 2017).

In general, the trajectory classification problem is challenging because of: (1) the massive volume of trajectory data is continuously generated by multiple users; (2) the complexity associated with the data representation (how can we represent latitude, longitude, and timestamp features in our models without losing essential information?); (3) the sequence of spatio-temporal points can be sparse in time and space, for instance, LBSNs usually contain samples in days; (4) the nature of multiple dimensions: as technologies advances, trajectories have more and more properties, such as acceleration, velocity, weather condition, POI category, and adverse events; (5) the number of the classes can be much larger than the number of motion patterns (more than fifty or one hundred class). (6) Trajectory datasets may present imbalanced distributions of the target variable, e.g., an Imbalance Ratio (IR) greater than two (Fernández et al., 2008).

More recent Deep Learning models emerged to link trajectories to their generating users. TULER proposed in (Gao et al., 2017), and TULVAE proposed in (Zhou et al., 2018) use a Recurrent Neural Network (RNN) for classification and minimize the sparse problem using an embedding vector, but both models do not handle multidimensional data. On the other hand, MARC proposed in (May Petry et al., 2020) and DeepeST proposed in (blind) handle multidimensional and use embedding to minimize the sparsity problem. However, these models offer no solution for dealing with imbalanced data.

Trajectory databases tend to be imbalanced due to different reasons. For instance, considering the TUL problem and the three LBSNs datasets (Gowalla, Brightkite, Foursquare NYC), some users check in more frequently than others, and the IRs of these datasets are respectively 3.75, 5.56, and 2.62, as shown in Figure 1.

Figure 1: Distribution of users' sub-trajectories for three trajectory datasets



For the trajectory classification problem, DeepeST employs LSTM/BLSTM (Schuster and Paliwal, 1997) and embeds location, time, or any features associated with a trajectory (or sub-trajectory) to jointly learn more meaningful representations of the trajectory data, which boosts DeepeST's performance. In this paper, we provide the following contributions: (1) we explore different approaches for coping with DeepeST to deal with imbalanced training data; (2) we extend the related works with papers that face the imbalanced data problem; (3) we compare the deep learning models for trajectory classification from multidimensional data; (4) we perform an extensive experimental evaluation over three real-world imbalanced datasets, where we assess the validity of our proposal in terms of quality of results.

Related Work

In this paper, we tackle the trajectory classification problem in imbalanced datasets. In this section, we first present the classification trajectory problem and then the imbalanced dataset problem.

Trajectory Classification

We claim that we classify sub-trajectories since our training set is derived from the segmentation of trajectories, as we will explain later. For the sake of brevity, from now on, we will use the term *trajectory classification* in place of *sub-trajectory classification*.

Trajectory classification is one of the widely studied problems on trajectory pattern mining over the years. In the beginning, trajectory classification focused on detecting patterns of mobility from raw trajectories using machine learning methods (Zheng et al., 2008; Patterson et al., 2003; Fang et al., 2016). One of the first methods for trajectory classification was TraClass, proposed by Lee et al. (2008) that supports only the spatial dimension. Patel extended the TraClass to support both the spatial and the time dimensions in (Patel, 2013). Machine learning methods can handle trajectories, but they demand a feature extraction process, and they suffer from the curse of dimensionality.

More recently, studies have investigated Deep Learning models to link trajectories to their generating users. TULER was the first model introduced in (Gao et al., 2017) for identifying and linking a large number of check-in trajectories to their generating-users using RNN based models. TULVAE, also using RNN, was proposed in (Zhou et al., 2018) after the TULER and enhanced in (Zhou et al., 2019), a generative model to mine human mobility patterns, which aims at learning the implicit hierarchical structures of trajectories and alleviating the data sparsity problem with the semi-supervised learning. These models are limited since they only deal with a spatial feature and do not consider other dimensions that are important for classification (e.g., the temporal information).

MARC (May Petry et al., 2020) and DeepeST (blind), both tackle trajectory classification problem for multidimensional data. Both proposals focus on the different spatial, temporal, and semantic attributes that characterize multidimensional trajectories. MARC uses a multi-attribute embedding layer to encode these heterogeneous dimensions. MARC outperforms BITULER, TULVAE, and other models in (May Petry et al., 2020). DeepeST outperforms BITULER, TULVAE, Random Forest, and XGBoost in (blind). However, both MARC and DeepeST do not handle imbalanced datasets and use different spatial representation approaches (MARC uses GeoHash linked to a dense layer, and DeepeST uses Grid Index linked to an embedding layer). Beyond providing a solution for imbalanced data for trajectory classification, this work compares both models for trajectory classification from multidimensional data.

Imbalanced Data in Deep learning Models

An imbalanced classification problem is when the distribution of examples across the known classes is biased or skewed. The distribution can vary from a slight bias to a severe imbalance. In other words, the number of samples in the minority class is smaller than the majority class, considering the proportion (for example, twice large or more (Fernández et al., 2008)).

Most of the classifiers assume that the data is balanced and evenly distributed in each class. Data imbalance can lead to unexpected mistakes and even severe consequences in the data analysis, especially in classification tasks, since the algorithm does not learn the patterns in minority classes. In general, there are two strategies to solve the imbalanced datasets problem: re-sampling and cost-sensitive re-weighting (Haixiang et al., 2017).

Using resampling methods, we modify the dataset itself by adding and removing samples based on three approaches:

1. Over-sampling: we increase the number of samples in the minority class to balance the classes (main techniques are SMOTE and randomly duplicating the minority samples);
2. Under-sampling: we eliminate samples from the majority class (an effective method is random sampling);
3. Hybrid methods: We combined the methods of oversampling and under-sampling.

Using approaches based on cost-sensitive re-weighting, we influence the loss function by assigning relatively higher costs to examples from minor classes and assigning relatively smaller costs to majority classes. In this way, we can control the level of imbalance of the dataset. Considering the deep learning context where we use recurrent neural networks, resampling approaches as over-sampling may either introduce large amounts of duplicated samples, slow down the training, and make the model susceptible to overfitting. Using under-sampling, we may discard valuable examples that are important for the algorithms. Due to these limitations, we focus on methods based on cost-sensitive re-weighting.

In the literature, deep learning models as (Yan et al., 2016; Lin et al., 2017; Cui et al., 2019; Wang et al., 2016) evaluated different loss functions and strategies based on cost-sensitive re-weighting. We can consider two main loss functions to classify tasks using deep learning models: (1) Focal loss applies a modulating term to the cross-entropy loss to focus on complex negative examples. The focal loss proved to be simple and highly effective, considering its accuracy and speed

results compared to state-of-the-art (Lin et al., 2017); (2) Class-Balanced Loss (CBCE) defines a sufficient number of samples as the volume of samples. CBCE provides two main advantages over the focal loss (1): CBCE finds an effective number of samples for each class; (2) CBCE is parameter-free. In the focal loss, the training takes longer, as the user needs to adjust the two parameters. For these two reasons and considering that CBCE outperforms other state-of-the-art functions, as shown in (Cui et al., 2019), we consider using CBCE as a loss function in DeepeST.

Problem Definition

Let a trajectory TR_j be a sequence of points sorted in time $[p_1, .., p_{len_j}]$. Here, p_i $(1 \leq i \leq len_j)$ is a tuple (l_i, t_i, A_i), such that l_i is a location point at time t_i, and $A_i = [a_1, ..., a_m]$ is a sequence of m attributes linked to the trajectory (e.g velocity, acceleration, geographic information, among others).

For sake of brevity, the location point l_i is a spatial coordinate e.g., latitude and longitude collected from a GPS device, or l_i can refer to check-in or stop location (it can be a POI location, for instance). It is worth to mention that if a trajectory is not linked to any semantic information, then $A_i = \emptyset$.

For simplicity, in this work, we represent each location l_i composed of latitude and longitude from a trajectory TR in a spatial grid cell, however it could be in any well-defined geographical space. We also map each timestamp t_i to a time slot in $T = \{t_1, t_2, ..., t_w\}$, such that $T \in \mathbb{R}^w$. A time slot could be a regular time interval, for instance, some minutes, hours, days or weeks. Finally, in order to reduce the computational complexity and capture richer knowledge of sub-trajectory patterns from trajectories, we segment the trajectories into sub-trajectories. There are several methods for trajectory segmentation based on the shape of a trajectory, time interval, and semantic meanings Zheng (2015). Since trajectory segmentation is not at the core part of this work, we adopt the simplest method based on the time interval to trajectories. A trajectory $[p_{q_1}, p_{q_2}, ..., p_{q_n}]$, such that $(1 \leq q_1 < q_2 < ... < q_n \leq len_j$, where $l_k = l_{k-1}$+1) is called a sub-trajectory of S_j. We are now ready to formulate our prediction problem for sub-trajectories.

Problem Statement

Given a set of sub-trajectories $\tilde{S} = \{S_1, S_2, ..., S_z\}$, the task is predict the category by linking each sub-trajectory $S_i \in \tilde{S}$ to a label $y \in Y = \{y_1, ..., y_o\}$. Notice that our problem is generic, Y can be a set of transportation mode (car, bus, bike or walk), a set of users that are owners of the trajectory or any category feature of other domains. In our problem, we consider imbalanced datasets (Imbalance Ratio > 2 Fernández et al. (2008)). Therefore, there exist $\tilde{S}^{y_i}, \tilde{S}^{y_j} \subset \tilde{S}$ two sets of trajectories in $\tilde{S}$ tagged with the labels y_i and y_j, respectively, such that $|\tilde{S}^{y_i}| > |\tilde{S}^{y_j}|$, i.e., the number of trajectories tagged with y_i is at least twice larger than the one tagged with y_j.

DeepeST - Deep Learning for Sub-Trajectory classification

In this paper, we extend a deep learning model, called DeepeST, for the trajectory classification problems in imbalanced domains. DeepeST receives a fit sequence of features like location, time, and any other attributes annotated in a sub-trajectory and outputs the predicted label or class.

Sub-trajectory encoding

The first problem in dealing with sub-trajectories is the data representation. The spatial dimension of trajectories is composed of two attributes, latitude, and longitude. Both are separate tributes that represent a single geographic location.

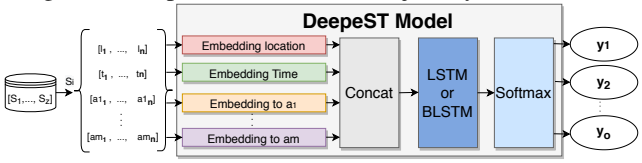
DeepeST creates a spatial grid 2D with a fixed size cell in meters covering all points in the dataset. An integer number represents each cell into the grid, and a user can specify the cell size: high values to cell size (10000 meters, e.g.) become the problem more challenging to capture the movement of an object within a small geographic space (for instance, a parking car). For smaller cell sizes, we can capture the switch between different geographical areas during the movement—however, the number of cells in the grid increases. Finally, we provide the Grid Index of each cell to the embedding layer.

For temporal representation, the timestamp data usually is a feature composed of day, hour, month, year, minute, and second (for instance, 2008/02/02 22:00:00). For each timestamp attribute, DeepeST extracts the hour of the day and the day of the week. Finally, we apply one-hot encoded to all available features and use it to input in its corresponding embedding layer.

DeepeST architecture

DeepeST architecture is composed of embedding layers to each input, a concatenation layer, a recurrent layer (LSTM or BILSTM), and a fully connected layer with softmax as the activation function. The overview of DeepeST is illustrated in Figure 2.

Figure 2: DeepeST model to sub-trajectory classification



DeepeST incorporates embedding layers to receives sequences from the sub-trajectory. An embedding is a relatively low-dimensional space into which he/she can translate high-dimensional vectors. Sub-trajectory embedding alleviates the curse of dimensionality and maintains the input data's proximity with similar patterns in a new dimensional space.

DeepeST uses a recurrent layer that receives input from a feature vector, but it presents embedding layers to each sub-trajectory attribute. A concatenation layer is defined between the embedding layers and the recurrent layer to join

embedding vectors in unique input features that will be used in the recurrent layer, as shown in Figure 2.

We use a Recurrent Neural Network (RNN) for trajectory classification due to its capacity to learn complex patterns from a sequence, unlike feedforward neural networks. DeepeST employs an LSTM (Hochreiter and Schmidhuber, 1997), which has been extensively used to process variable-length input and can allow highly non-trivial long-distance dependencies to be easily learned. We also examined a Bi-directional LSTM (BILSTM) in our DeepeST model (Schuster and Paliwal, 1997), which can take into account an effectively infinite amount of context on both sides of a sub-trajectory and eliminates the problem of limited context that applies to any feedforward model.

Loss Fuctions

DeepeST works with two loss functions: Categorical Cross-Entropy (CCE) and the Class-Balanced Cross-Entropy Loss (CBCE). CCE is a loss function to multi-class classification tasks, where the labels are provided in a one-hot representation. Formally, CCE quantifies the difference between probability distributions to two or more classes. CCE considers a single weighting factor for each class. Therefore, CCE may not be appropriate for imbalanced databases.

On the other hand, CBCE is designed to address training from imbalanced data by introducing a weighting factor that is inversely proportional to the effective number of samples (Cui et al., 2019). An effective number of samples can be calculated by a formula (1 - β^n)/(1-β), where n is the number of samples and $\beta \in [0, 1)$ is a hyperparameter. In other words, a class-balanced re-weighting term is inversely proportional to the effective number of samples is added to the loss function. The class-balanced loss term can be applied to a wide range of deep networks and loss functions (Cui et al., 2019).

Optimization in DeepeST

Overfitting is a major problem in RNN due to a large number of weights and biases. To alleviate overfitting, we determined a dropout layer for regularization. Dropout is a strategy radically different from other approaches, since it changes the network structure itself, instead of the cost function. Suppose we have a training set X and the corresponding desired output y. Normally, we train by direct propagation of X across the network, and then the backpropagation algorithm computes the error to the gradient. When we use a layer dropout, this process is modified. We eliminate by randomly (and temporarily) some of the neurons hidden in the network, but leave the input and output neurons untouched. Heuristically, if we abandon different sets of neurons, we are training with various neural networks. Therefore, dropout can reduce overfitting, whereas other networks adapt in different ways.

Experimental evaluation

In this section, we present the experimental evaluation to evaluate DeepeST in terms of quality prediction. We start by providing details about the datasets, the experimental setup, the baselines, followed by the experimental evaluation.

Datasets

We conduct our experiments on three datasets: (1) a public dataset that contains check-ins of users around the world extracted from Brightkite [1] between April 2008 and October 2010. (2) a public dataset that contains trajectories of check-in extracted from Gowalla [2] between February 2009 and October 2010; (3) a public dataset of Foursquare in New York City (NYC), USA, collected between April 2012 and February 2013. Gowalla, Brightkite, and Foursquare NYC have been widely used in several works on trajectory classification (Gao et al., 2017; Zhou et al., 2018, 2019; May Petry et al., 2020) and their IRs are respectively 3.75, 5.56, and 2.62, as presented in Figure 1.

Performance Comparison

There are four variations for DeepeST concerning the network layers and loss functions: two with BILSTM, one configured with CCE loss (DeepeST-BILSTM-CCE), and another CBCE loss (DeepeST-BILSTM-CBCE). Two with LSTM, one configured with CCE loss (DeepeST-LSTM-CCE), and another with CBCE loss (DeepeST-LSTM-CBCE). We compare DeepeST variation with three state-of-the-art approaches from the field of deep learning classification: BiTULER (Gao et al., 2017), TULVAE (Zhou et al., 2018), and MARC (May Petry et al., 2020).

Experimental setup

The classification task is to predict the corresponding user who generated a given trajectory. We use segmentation based on time and create weekly sub-trajectories from each user check-in, as performed in (May Petry et al., 2020). We selected only sub-trajectories of users who have at least 15 weekly sub-trajectories because we will have at least two samples for each user in the validation set and the test set. For the Grid index approach in DeepeST, we created a virtual grid cell and set a cell size of $30m$, covering all points for each dataset. Therefore, each latitude and longitude is mapped to a $30m$ x $30m$ region. We use 30 meters because we believe it is sufficient to separate even small points of interest, like bars, homes, and restaurants. For the GeoHash representation used in MARC, we use Base32 in GeoHash for building instance, as performed in (May Petry et al., 2020).

To validate the models, we split performing a stratified holdout evaluation in each dataset, considering 70% to training, 15% to validation, and 15% to test. We shuffle sub-trajectory data, run the baselines algorithms ten times for each dataset, and compared the models using Accuracy, Macro Precision, Macro Recall, and Macro F1-Score.

To find the optimal set of hyperparameters, we apply the grid-search technique to combine several hyperparameters for each model. Although it is extremely computationally expensive and may take your machine quite a long time to run, grid-search ensures that we find the best set of hyperparameters considering each model and dataset. We keep 64 as the batch size and 0.001 as the learning rate for all the

[1] https://snap.stanford.edu/data/loc-Brightkite.html

[2] https://snap.stanford.edu/data/loc-gowalla.html

models and 0.5 as dropout (**dp**). We also vary the units (**un**) of the recurrent layers, the embedding size to each attribute (**es**), and the units of the dense layer used after the GeoHash attribute (**ds**). We determine a first stopping callback, which is a stop training when, in our case, the accuracy has stopped improving. We set the early stopping as 20 for the patience argument to minimize overfitting, i.e., the number of epochs that produced the model's accuracy with no improvement after which training should be stopped. For further details, we refer to Keras library [3]. Exclusively for TULVAE, besides vary the units (**un**) and the embedding size (**es**) of the POI identifier, we also vary the latent variable (**z**). For more details about parameters or reproducibility purposes, we made the source code available on GitHub [4].

Classification results

From the results reported in Tables 1, 2 and 3, we summarizes the performance comparison between the variants of DeepeST, MARC, BITULER, and TULVAE using all available features for the three datasets. We highlighted the two best values in **bold**, and the third one in *underlined*. For what concerns DeepeST variations and MARC, a sub-trajectory S is a sequence with each of the following attributes $(ig_i, hr_i, poi_i, wk_i)$, where ig is the grid index cell, and poi is the POI identifier, wk is the weekday, and hr is the hour); BITULER and TULVAE only deal with one feature, so the input is a sequence of POI identifier as presented in (Gao et al., 2017; Zhou et al., 2019).

Table 1: Classification results on stratified holdout evaluation in the Brighkite dataset

Model	Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	0,9729	0,0017	**0,9716**	0,0034	0,9691	0,0020	**0,9669**	0,0027
DeepeST-BILSTM-CCE	**0,9757**	0,0021	**0,9738**	0,0037	**0,9695**	0,0026	**0,9687**	0,0033
DeepeST-LSTM-CBCE	0,9735	0,0017	0,9715	0,0035	**0,9693**	0,0026	0,9668	0,0032
DeepeST-LSTM-CCE	**0,9748**	0,0019	0,9706	0,0043	0,9678	0,0031	0,9660	0,0036
MARC	0,9718	0,0033	0,9685	0,0061	0,9642	0,0043	0,9628	0,0052
BITULER	0,9520	0,0053	0,9470	0,0068	0,9350	0,0057	0,9364	0,0060
TULVAE	0,9581	0,0012	0,9459	0,0028	0,9426	0,0021	0,9403	0,0022

Table 2: Classification results on stratified holdout evaluation in the Gowalla dataset

Model	Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9688**	0,0021	**0,9608**	0,0024	**0,9645**	0,0016	**0,9574**	0,0022
DeepeST-BILSTM-CCE	0,9665	0,0044	0,9577	0,0075	0,9619	0,0039	0,9541	0,0058
DeepeST-LSTM-CBCE	**0,9682**	0,0016	**0,9609**	0,0053	**0,9638**	0,0031	**0,9568**	0,0043
DeepeST-LSTM-CCE	0,9669	0,0024	0,9587	0,0062	0,9619	0,0037	0,9541	0,0051
MARC	0,9456	0,0060	0,9358	0,0122	0,9457	0,0063	0,9340	0,0091
BITULER	0,9008	0,0111	0,8964	0,0126	0,8927	0,0117	0,8782	0,0140
TULVAE	0,9199	0,0092	0,9240	0,0143	0,9210	0,0137	0,9110	0,0124

We can see that DeepeST with (LSTM/BILSTM) using all features combined in sub-trajectory classification yields improvements over the baselines on the three datasets, considering accuracy, precision, recall, and F1-score. DeepeST takes advantage of the CBCE/CCE, data representation using Grid Index and embedding, and LSTM/BILSTM that operates at embedding levels to learn the underlying user

[3]https://keras.io/

[4]https://github.com/nickssonarrais/DeepeST-FLAIRS

Table 3: Classification results on stratified holdout evaluation in the Foursquare NYC dataset

Model	Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9957**	0,0022	**0,9961**	0,0022	**0,9948**	0,0027	**0,9947**	0,0026
DeepeST-BILSTM-CCE	**0,9933**	0,0016	0,9930	0,0015	**0,9919**	0,0016	**0,9918**	0,0016
DeepeST-LSTM-CBCE	0,9927	0,0038	**0,9931**	0,0030	0,9915	0,0034	0,9915	0,0036
DeepeST-LSTM-CCE	0,9927	0,0021	0,9930	0,0020	0,9914	0,0020	0,9914	0,0021
MARC	0,9893	0,0052	0,9913	0,0044	0,9883	0,0052	0,9878	0,0058
BITULER	0,9890	0,0039	0,9917	0,0042	0,9877	0,0041	0,9880	0,0049
TULVAE	0,9820	0,0050	0,9854	0,0043	0,9809	0,0052	0,9798	0,0058

categories from check-ins sub-trajectory data. RNN models (LSTM/BILSTM) are proper models to learn from temporal sequences as sub-trajectories. We can notice that MARC outperforms BITULER and TULVAE across all metrics in most datasets. Both MARC and DeepeST built a more representative model using a set of variables instead of only using a POI identifier. As we can see, only POIs identification is not sufficient to distinguish different users in Gowalla and Brightkite. In Foursquare NYC, all models reached values above 98% considering accuracy, precision, recall, and above 97% considering F1-macro. This occurs when only the spatial feature is the relevant one to detect user mobility patterns. It is worth noting that results of MARC and DeepeST for all datasets could be higher if there exist more relevant features to separate the classes from different users in the dataset (maybe features based on external events and features that characterize different people). The expert's view of the application domain can be essential to increase the performance of the model.

Comparing DeepeST variations and MARC, note hat DeepeST yields improvements over MARC on the three datasets, considering accuracy, precision, recall, and F1-score. MARC uses GeoHash linked to a dense layer to represent the spatial feature. While DeepeST uses the Grid Index linked to an embedding layer. Using the Grid Index linked to an embedding provided better results because an embedding layer approximates the input data to recurrent neural. The embedding output is denser than the dense layer output. Grid index generates an integer vector that can be directly provided for an embedding layer, considering that we use a padding sequence to ensure the fixed-size sequence. Using the GeoHash approach presented in (May Petry et al., 2020), there is a transformation to a binary matrix that cannot provide input for an embedding layer. In this case, the GeoHash uses a dense layer that makes the sparse data to the RNN. When we use a dense layer, we lose the ability to make the data denser, as in the embedding layer.

Regarding DeepeST and its variations, in general, we note that DeepeST-BILSTM provided improvements over DeepeST-LSTM. DeepeST-BILSTM runs its inputs in two ways, one from past to future and one from future to past. In our experiments, DeepeST-BILSTM achieved slightly more significant results than DeepeST-LSTM for the three datasets. DeepeST-BILSTM takes into account an effectively infinite amount of context on both sides of a sub-trajectory position and eliminates the problem of limited context that applies to any feed-forward model. It is important to highlight that the results between DeepeST-BILSTM and DeepeST-LSTM are very close. However, DeepeST-

BILSTM proved better and faster to understand the context using past and future information.

If we look at the solutions for coping with imbalanced datasets, we can notice that CBCE loss provides improvements in the recall's model. Note that DeepeST-BILSTM-CBCE achieved the highest recall values in all the datasets. CBCE quantifies the weighting factor inversely proportional to the effective number of samples per class. In other words, considering the dataset distribution to each user, the CBCE provides a lower weight to the majority classes and a higher weight to the minority classes. In this way, the algorithm learns the patterns of the minority classes. When we use CCE, the weight factor is the same for all classes, so the algorithm does not learn the patterns in minority classes.

Conclusion

In this paper, we investigate the trajectory classification problem from imbalanced datasets for classifying a category from a set of labels. We evolved a deep learning model, called DeepeST (Deep Learning for Sub-Trajectory classification), to identify any category from many sub-trajectories extracted from imbalanced datasets. We believe that DeepeST is the first model for trajectory classification that provides resources to deal with imbalanced datasets. We conducted extensive experiments on three real datasets extracted from LBSNs to evaluate DeepeST performance with state-of-the-art approaches from the field of Deep Learning (MARC, BITULER, and TULVAE). DeepeST achieves more expressive values of accuracy, precision, recall, and f1-score in all experiments. Although we evaluated the classification considering the Trajectory-User Linking problem, DeepeST is generic enough to deal with different trajectory classification problems in imbalanced domains, such as transportation mode inference, the prediction to the next-stop points.

As future directions, first, we aim at analyzing other loss functions in different scenarios. We can integrate more functionalities into DeepeST to become a framework for trajectory classification. Finally, although we achieved very high results across all datasets, we aim to study how to improve accuracy using other deep learning techniques, like attention mechanisms.

References

blind. Blind. In *blind*.

Cui, Y.; Jia, M.; Lin, T.-Y.; Song, Y.; and Belongie, S. 2019. Class-Balanced Loss Based on Effective Number of Samples.

Fang, S.-H.; Liao, H.-H.; Fei, Y.-X.; Chen, K.-H.; Huang, J.-W.; Lu, Y.-D.; and Tsao, Y. 2016. Transportation modes classification using sensors on smartphones. *Sensors* 16(8):1324.

Fernández, A.; García, S.; del Jesus, M. J.; and Herrera, F. 2008. A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets. *Fuzzy Sets and Systems* 159(18):2378–2398.

Gao, Q.; Zhou, F.; Zhang, K.; Trajcevski, G.; Luo, X.; and Zhang, F. 2017. Identifying human mobility via trajectory embeddings. In *IJCAI*, volume 17, 1689–1695.

Haixiang, G.; Yijing, L.; Shang, J.; Mingyun, G.; Yuanyue, H.; and Bing, G. 2017. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications* 73:220–239.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Lee, J.-G.; Han, J.; Li, X.; and Gonzalez, H. 2008. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment* 1(1):1081–1094.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal Loss for Dense Object Detection.

May Petry, L.; Leite Da Silva, C.; Esuli, A.; Renso, C.; and Bogorny, V. 2020. MARC: a robust method for multiple-aspect trajectory classification via space, time, and semantic embeddings. *International Journal of Geographical Information Science*.

Patel, D. 2013. Incorporating duration and region association information in trajectory classification. *Journal of Location Based Services* 7(4):246–271.

Patterson, D. J.; Liao, L.; Fox, D.; and Kautz, H. 2003. Inferring high-level behavior from low-level sensors. In *International Conference on Ubiquitous Computing*, 73–89. Springer.

Schuster, M., and Paliwal, K. K. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681.

Wang, S.; Liu, W.; Wu, J.; Cao, L.; Meng, Q.; and Kennedy, P. J. 2016. Training deep neural networks on imbalanced data sets. In *2016 International Joint Conference on Neural Networks (IJCNN)*, 4368–4374. IEEE.

Yan, Y.; Chen, M.; Shyu, M. L.; and Chen, S. C. 2016. Deep Learning for Imbalanced Multimedia Data Classification. In *Proceedings - 2015 IEEE International Symposium on Multimedia, ISM 2015*, 483–488. Institute of Electrical and Electronics Engineers Inc.

Zheng, Y.; Liu, L.; Wang, L.; and Xie, X. 2008. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, 247–256. ACM.

Zheng, Y. 2015. Trajectory Data Mining. *ACM Transactions on Intelligent Systems and Technology* 6(3):1–41.

Zhou, F.; Gao, Q.; Trajcevski, G.; Zhang, K.; Zhong, T.; and Zhang, F. 2018. Trajectory-user linking via variational autoencoder. In *IJCAI*, 3212–3218.

Zhou, F.; Yin, R.; Trajcevski, G.; Zhang, K.; Wu, J.; and Khokhar, A. 2019. Improving human mobility identification with trajectory augmentation. *GeoInformatica* 1–31.