TemporalNode2vec: Task-agnostic and Task-specific Temporal Node Embedding

Mounir HADDAD (mounir.haddad@imt-atlantique.fr)
IMT Atlantique Bretagne-Pays de Loire
Cécile BOTHOREL
IMT Atlantique Bretagne-Pays de Loire
Philippe LENCA
IMT Atlantique Bretagne-Pays de Loire
Dominique BEDART
DSI Global Services

Research

Keywords: Dynamic network embeddings, Graph representation learning, Latent representations

Posted Date: March 4th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-16018/v1

TemporalNode2vec: Task-agnostic and Task-specific Temporal Node Embedding

Mounir Haddad[1,2], Cécile Bothorel[1], Philippe Lenca[1], Dominique Bedart[2]

[1] IMT Atlantique, LUSSI Department, UMR Lab-sticc, 29238 Brest Cedex 3, France,
[2] DSI Global Services, 41 avenue du Général Leclerc, 92350 Plessis-Robinson, France,
{cecile.bothorel, philippe.lenca, mounir.haddad}@imt-atlantique.fr
{dominique.bedart, mounir.haddad}@dsi-globalservices.fr

Abstract. The goal of graph embedding is to learn a representation of graphs vertices in a latent low-dimensional space in order to encode the structural information that lies in graphs. While real-world networks evolve over time, the majority of research focuses on static networks, ignoring local and global evolution patterns. A simplistic approach consists of learning nodes embeddings independently for each time step. This can cause unstable and inefficient representations over time.
In this paper, we present TemporalNode2vec, a novel dynamic graph embedding approach that learns continuous time-aware node representations. Overall, we demonstrate that our method improves node classification tasks comparing to previous static and dynamic approaches as it achieves up to 14% gain regarding the F1 score metric. We also prove that our model is more data-efficient than several baseline methods, as it affords to achieve good performances with a limited number of node representation features. Moreover, we develop and evaluate a task-specific variant of our method called TsTemporalNode2vec, aiming to improve the performances and the data-efficiency of node classification tasks.

Keywords: Dynamic network embeddings, Graph representation learning, Latent representations

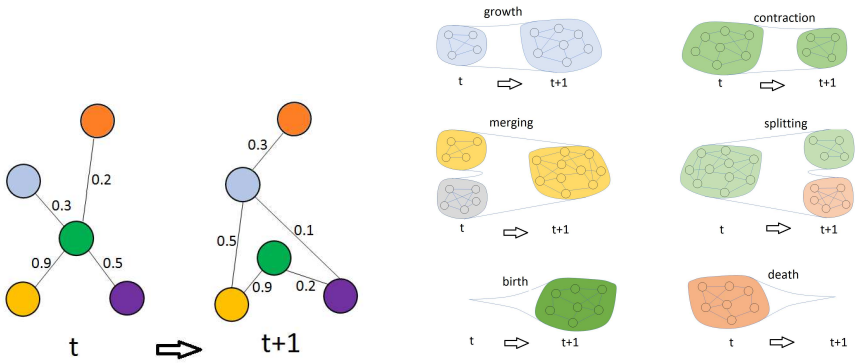
1 Introduction

The last decade has seen social networks flourish, as well as the data they generate. These data, usually represented using graphs, contain important information on interaction phenomena at multiple scales: it goes from the local interaction patterns between elementary entities to the global dynamics of communities.

Since several years, a multitude of research has focused on extracting relevant structural information in networks, such as communities characteristics or information diffusion patterns. However, in order to make graph data exploitation easier for machine learning inference models, one has to build a relevant representation of nodes/edges. Traditional machine learning approaches rely on user defined heuristics [4,26,17]. Nevertheless, those methods are time-consuming in terms of feature engineering. They also are without guarantee of performance for prediction tasks different than the one they were conceived for.

Since few years, new research approaches known as data embeddings aim to learn data representation in low dimensional spaces [3]. They are to be considered as a preprocessing step and they cover a large spectrum of data science fields ranging from image processing to time series. As these methods are data-driven and not relying on hand-engineered statistics, they encode data into a generic representation, independent from downstream machine learning tasks. The early applications of data embeddings focused on text mining [20]. Regarding the advantages of such approaches, they have been adapted to graphs among other data structures. Therefore, last years have seen a surge in graph representation learning. Some approaches are based on matrix factorization (Laplacian eigenmaps [2], GraRep [6], HOPE [22]) and other techniques are based on random walks ([24,12,8]). Another type of methods uses recent advances in deep learning: auto encoders for DNGR [7] and SDNE [27], or convolutional neural networks like [14]. Also, it is worthwile to cite [5] which addresses the uncertainty for building vertices embeddings.

Although graph embedding methods were disruptive and have imposed themselves as baselines, they avoid considering the temporal dimension, which remains a fundamental aspect in the understanding of interaction phenomena. They focus on static networks where the structure of vertices and edges stays fixed among time. However, recent work incorporating time is beginning to emerge [29,19,21,31]. The aim of temporal embedding is to obtain a representation of a network at each time step and at different scales. The local evolution of nodes and edges over time is captured as well as the changes of communities at a more global scale (Figure 1).



(a) Local scale (b) Global scale

Fig. 1: Network evolution

In many use cases, time is crucial for inference purposes. For example, churn detection would be very inefficient on static networks as its signatures happen across time. Literature deals with time incorporation in several ways.

– Embeddings can be learned independently for each time step, using static methods [24,12].

- Some approaches learn time steps embeddings separately and search for an optimal linear transformation of the output matrices which minimizes the distance between consecutive embeddings [15].
- Another approach is to learn current time step representation by initializing it with the previous embedding vectors [19].

Those different methods can be sensitive to data sparsity: for example a missing node in a time step would mislead its representation and place it far from its previous and next embeddings.

In this paper, we propose TemporalNode2vec [13], a novel dynamic graph embedding approach, based on the static node embedding algorithm Node2vec and a smoothing mechanism for dynamic words embeddings [28]. Our main contribution lies in the way we incorporate the temporal information: the different time steps embeddings are learned jointly. We show in our experiments that this method improves node classification and node class prediction tasks comparing to relevant baseline algorithms. We also go further the simple comparison between models to interpret the hyper parameters, especially the influence of the embedding dimension. Moreover, we adapt TemporalNode2vec to task-specific embedding and present its usefulness.

The remainder of the paper is organized as follows. In Section 2, we provide the problem definition. In Section 3, we describe how we set up our temporal graph embedding model. In section 4, we expose the performed experiments and interpret the obtained results. Lastly, we introduce and evaluate the task-specific variant of TemporalNode2vec in section 5.

2 Problem definition

We consider a network composed of $|V|$ vertices $V = \{v_1, \dots, v_{|V|}\}$ connected by timestamped interactions $I = \{i_{j,k,t}\}$ where $i_{j,k,t}$ represents an interaction between v_j and v_k at t. The goal of temporal embedding is to have a representative vector of each vertex at each time step. However, regarding the number of the different timestamps (sometimes equal to the number of interactions), it is obvious that there should be some aggregation over time: having an embedding for each network change (i.e. at each single interaction) would not be relevant as consecutive embeddings would barely change. We consider T time slices (referred by time steps in the rest of the paper) rather than instant moments. Within each time step, we aggregate the interactions between pairs of vertices, to form weighted edges (where weights represent the number of grouped interactions). Formally, the whole input dataset is assumed to be a set of T graphs $\{G_1, \dots, G_T\}$ where G_t is the undirected weighted graph representing the interactions of the t-th time step. We define temporal node embedding as follows: Given a sequence of graphs $\{G_1, \dots, G_T\}$, we aim to learn for each time step t ($t \in [\![1, T]\!]$) mapping functions $f_t : V \to \mathbb{R}^d$. Here, d is the embedding dimension ($d << |V|$). The output expected is a set of T matrices of size $d \times |V|$ corresponding to the learned mappings: the t-th matrix (let U_t be this matrix in the rest of the paper) contains the embedding vectors of all vertices at time step t.

3 TemporalNode2vec

3.1 Random walks

The main idea behind the very first graph embedding approaches is adapting word embedding techniques that offer remarkable improvement in terms of inference comparing to previous work. For that purpose, one has to transform graph data to *sentences* of vertices. DeepWalk comes with a method to build those sequences of vertices called *walks*, inspired by early applications of random walks [23]. The idea is to simulate a probabilistic *walk* all over a graph vertices through its (un)directed (un)weighted edges. Node2vec comes with an extension of random walks by offering more flexibility to the way nodes neighborhoods are explored, i.e. whether to privilege local exploration (breadth-first sampling or BFS) or a macro-view structures discovery (depth-first sampling or DFS). This flexibility, due to adding two tunable parameters, improves the embedding quality [12]. In our method, we build our random walks in the same fashion. This means that, given a sequence of temporal graphs $\{G_1, \dots, G_T\}$, we compute T sets of random walks. We then obtain $T \times N \times |V|$ sequences of vertices of length l, where N is the number of walks starting from each vertex.

3.2 PPMI matrices

In the literature of words embeddings, we observe that similar words have similar neighboring words [9]. This property, called homophily, is also present in social graphs [10]. One way to capture nodes neighboring structures uses statistics on the frequencies by which vertices co-appear in the set of random walks considered [7]. We define the Positive Pointwise Mutual Information matrix (PPMI) as follows:

$$PPMI(v_1,\, v_2)_t = max\left(0,\, log\left(\theta\, \frac{|v_1,\, v_2|_t^w \cdot |V|}{|v_1|_t \cdot |v_2|_t}\right)\right) \qquad (1)$$
$$\forall (v_1,\, v_2 \neq v_1,\, t) \in V^2 \times [\![1,\, T]\!]$$

where $|v_1,\, v_2|_t^w$ is the number of times v_1 and v_2 co-appear in the set of walks of G_t within a window of size w, $|v_i|_t$ is the number of occurrences of v_i in the set of walks of G_t, and θ is a tunable hyper parameter. The $PPMI$ matrices can be seen as temporal similarity matrices. It is worthwhile to mention that we use *positive* PMI matrices in order to reduce instability, as $log\left(\theta\, \frac{|v_1,\, v_2|_t^w \times |V|}{|v_1|_t \times |v_2|_t}\right)$ can produce large negative values, which illustrate rare co-appearance, assimilable to zero co-appearance. The trade-off between stability and rare co-appearance is controlled by θ. Thresholding PMI matrices in this way is also used in [16].

3.3 Objective function components

The key idea behind several static graph embedding techniques [1,6,22,24,12] is to learn U_t matrices satisfying:

$$u_1(t)^T\, u_2(t) \approx PPMI(v_1,\, v_2)_t \qquad (2)$$

where $u_i(t)$ is the embedded vector of v_i at time step t. Node2vec uses negative sampling to implicitly satisfy (2). In the case of words embeddings, [16] observes that it is equivalent to low-rank factorization of PMI matrices, shifted by a constant (θ in equation (1)). Therefore, the static term L_{St} of our objective function can be represented as:

$$L_{St} = \sum_{t=1}^{T} \left\| PPMI_t - U_t\, U_t^T \right\|_F^2 \tag{3}$$

An important issue of learning temporal embeddings of a dynamic network is called alignment: the learned temporal vectors, encoding nodes/edges, may not be placed in the same latent space over time. To conceive a dynamic embedding method, one has to deal with alignment, since L_{St} is invariant to U_t matrices different rotations:

$$L_{St} = L_{St}(U_1, \ \dots \ , U_T) = L_{St}(U_1\, R_1, \ \dots \ , U_T\, R_T) \tag{4}$$

where $R_t \in SO(d)$ and $SO(d)$ is the special orthogonal group of dimension d, i.e. the group of rotations. Thus, it is necessary to add a smoothing term L_{Sm} to our objective function:

$$L_{Sm} = \sum_{t=2}^{T} \|U_t - U_{t-1}\|_F^2 \tag{5}$$

After adding a final term L_{LR} standing for low-rank data-fidelity enforcement as adopted in [28], the overall objective function to minimize is as follows:

$$\begin{aligned} L = \ & L_{St} \ + \tau\, L_{Sm} \ + \lambda\, L_{LR} \\ = & \sum_{t=1}^{T} \left\| PPMI_t - U_t\, U_t^T \right\|_F^2 + \tau \ \sum_{t=2}^{T} \|U_t \ - \ U_{t-1}\|_F^2 + \lambda \ \sum_{t=1}^{T} \|U_t\|_F^2 \end{aligned} \tag{6}$$

3.4 Objective function optimization

Common temporal networks contain tens of thousands of vertices. So, the $PPMI$ matrices may not fit in memory, even though they are sparse. Minimizing the objective function may then require additional simplifications. In the same fashion as [28], we introduce W_t matrices and γ hyper parameter to break the symmetry of $PPMI$ factorization. Thus, we can write our modified objective function as follows:

$$\begin{aligned} & \sum_{t=1}^{T} \left\| PPMI_t - U_t\, W_t^T \right\|_F^2 \ + \lambda \ \sum_{t=1}^{T} \|U_t\|_F^2 + \lambda \ \sum_{t=1}^{T} \|W_t\|_F^2 \\ & + \tau \ \sum_{t=2}^{T} \|U_t \ - \ U_{t-1}\|_F^2 + \tau \ \sum_{t=2}^{T} \|W_t \ - \ W_{t-1}\|_F^2 \ + \ \gamma \ \sum_{t=1}^{T} \|U_t \ - \ W_t\|_F^2 \end{aligned} \tag{7}$$

Minimizing (7) for U_t (equivalently for W_t) is solvable by simple 0 gradient, i.e. $U_t\, M = N$, where

$$M = W_t^T\, W_t + (\lambda + 2\tau + \gamma)\, I$$
$$N = PPMI_t\ W_t + \gamma\, W_t + \tau\, (U_{t-1} + U_{t+1})$$

for $t \in [\![2,\, T-1]\!]$ and constants adjusted for $t \in \{0,\, T\}$. We use block coordinate descent in order to solve $U_t\, M = N$ to afford scaling large $PPMI$ matrices, as only blocks of U_t (equivalently W_t) are updated at each time, therefore only blocks of $PPMI_t$ are loaded in memory at each time.

3.5 Temporal embeddings initialization

As $PPMI$ are a sort of temporal similarity matrices, they can be used during the initialization step to speed up the convergence of our method. For $t \in [\![1,\, T]\!]$, we initialize U_t and W_t by choosing carefully d rows of $PPMI_t$. We choose the top d l1-normalized rows maximizing the variance: a high variance means a better discrimination between nodes, and the l1-norm stands for normalizing the nodes regarding their number of occurrences within the random walks.

Algorithm 1: TemporalNode2vec

Data: $(\{G_1,\, \dots\, ,\, G_T\},\, V,\, d,\, l,\, w,\, p,\, q,\, \theta,\, \lambda,\, \gamma,\, \tau,\, iter)$
for $t \leftarrow 1$ ***to*** T **do**
  $walks = generateWalks(G_t,\, l,\, p,\, q)$;
  $PPMI_t = computePPMI(walks,\, w,\, \theta)$;
end
$\{U_1,\, \dots\, ,\, U_T,\, W_1,\, \dots\, ,\, W_T\} = initialize(PPMI,\, d)$;
for $i \leftarrow 1$ ***to*** $iter$ **do**
  for $t \leftarrow [\![1,\, T]\!]$ **do**
    for $b \leftarrow batch(V)$ **do**
      $U_t[b] = update(W_t[b],\, \lambda,\, \gamma,\, \tau,\, PPMI_t)$;
      $W_t[b] = update(U_t[b],\, \lambda,\, \gamma,\, \tau,\, PPMI_t)$;
    end
  end
end
return $\{U_1,\, \dots\, ,\, U_T\}$

4 Experiments

4.1 Baseline methods

As seen above, our algorithm has multiple hyper parameters: $\{l, w, p, q, \theta, \lambda, \gamma, \tau\}$. Testing 3 values for each over a grid search would lead to more than 6k sets of parameters. To workaround this issue, we choose to look for optimal parameters one by one, by fixing the others. Table 1 summarizes the tested values. To evaluate TemporalNode2vec, we consider 3 state-of-the-art baseline methods.

- **DeepWalk** [24]: This is one of the first methods to learn representations of a static graph nodes, using random walks over edges. DeepWalk has two hyper parameters: the walk length l and the window size w. For our experiments, we test a grid search over $(l, w) \in \{20, 40, 60, 80\} \times \{4, 8, 12, 16\}$.
- **Node2vec** [12]: Node2vec extends DeepWalk's way of discovering nodes neighborhood by adding two hyper parameters: the return parameter p and the in-out parameter q. For our experiments, we keep the values of l and w giving the best performances in DeepWalk tests, and perform a grid search over $(p, q) \in \{0.5, 1, 1.5, 2, 5\}^2$.
- **DynamicTriad** [29]: DynamicTriad is a dynamic representation learning approach focusing on how triads of vertices close, i.e how a pair of nodes sharing a common neighbor vertex can connect over an edge. This method has two hyper parameters: β_0 which is the weight of triad closure process and β_1 the temporal smoothing parameter. For our benchmarking experiments, we refer to DynamicTriad paper tests and consider a grid search over $(\beta_0, \beta_1) \in \{0.01, 0.1, 1, 10\}^2$.

Also, there are few other dynamic approaches we do not consider in our comparison due to the unavailability of the code (Dynnode2vec [19]) or their proven inefficiency[3] (Temporal network embedding [30]).

Concerning the embedding dimension, we have chosen $d = 48$ for the sake of memory space. Smaller values of d are tested in section 4.8.

4.2 Datasets

To compare the different algorithms embeddings performances, we gathered 3 real-world datasets. These datasets must meet some criteria. They should consist of temporal networks, i.e. sets of timestamped graphs. Also, the datasets must include metadata showing the existing evolving ground-truth communities (called labels in the rest of the paper). Part of the evaluation determines whether embedding approaches preserve community membership.

- **AMiner**: This dataset[4] consists of 51k researchers and 624k coauthor relationships. It is divided into 17 timestamped weighted graphs, where weights represent the number of common co-authorships between pairs of researchers within a time step. As articles are published in conferences addressing different research fields, it is possible to assign labels to authors: at each time step, we assume that a researcher belongs to a community (i.e. research field) if the majority of his papers are published in related conferences. Also, for the experiments made for embedding dimension analysis, we consider 3 samples of different sizes of this dataset (Table 2).
- **Yelp**: This dataset is an extract of Yelp[5] challenge dataset. It traces internet users comments on businesses (like restaurants and malls). We consider users

[3] [29] proved its inefficiency comparing to later graph embedding methods
[4] We use DynamicTriad [29] derived version of ArnetMiner [25].
[5] https://www.yelp.com/dataset/challenge

and businesses as being nodes, and comments are regarded as interactions. Businesses are assigned with categories. We keep only the top eight categories for our experiments. As businesses categories do not change over time, we assign labels for users: within each time step, a user is assumed to belong to a category if the majority of his comments are made on this category related businesses. Finally, our whole temporal graph is divided into 17 timestamped graphs, with a total of 744k edges between 38k nodes.

– **Tmall**: This dataset is an extract of the sales at Tmall.com[6] 6 months before the "Double 11 Day" event in 2014. It stores buyers interactions with products (i.e. click on product page, add to cart, purchase or add to favourites). Products are assigned with categories. For our experiments, we consider only the top five categories. We assign labels to users as described for Yelp dataset. We obtain a temporal graph divided into 10 timestamped graphs, with a total of 2.9M edges between 27k nodes.

Parameter	Tested Values
l	{20, 40, 60, 80}
w	{4, 7, 10, 13, 16}
p	{0.2, 0.5, 1, 2, 5}
q	{0.2, 0.5, 1, 2, 5}
θ	{0.05, 0.25, 4, 20}
λ	{0, 0.1, 0.3, 1, 3, 10}
γ	{0.1, 0.3, 1, 3, 10}
τ	{0, 0.1, 0.3, 1, 3, 10}

Table 1: Model tested values

Dataset	nodes	edges	time steps
AMiner	51 060	624 381	17
Smp_A	16 546	12 826	
Smp_B	9 244	5 884	
Smp_C	2 300	1 230	
Yelp	38 475	744 195	17
Tmall	27 039	2 976 392	10

Table 2: Datasets

4.3 Application scenarios

We challenge our approach and baseline methods by executing different inference tasks on the basis of their respective embeddings, regarding the F1 score metric. As those tasks are identical for the compared methods, better scores should reflect more efficient embeddings. Following inference tasks have been experimented:

– Node classification: A classifier is trained on nodes embeddings to find their labels. Logistic regression is used as the classification method.
– Node class prediction: This task is similar to node classification. We predict nodes labels at a time step using their embeddings at the previous time step.
– Edge reconstruction: In this task, the goal is to determine whether there is an edge between a pair of nodes, based on the distance between their embeddings.
– Edge prediction: This task is similar to edge reconstruction with the difference that we aim to predict the presence of edges between pairs of nodes regarding their previous embeddings.

[6] https://tianchi.aliyun.com/competition/entrance/231576/information

4.4 Comparison results

Table 3 shows the results of the experiments made using the different considered methods. It demonstrates that TemporalNode2vec outperforms other baseline methods in the tasks related to node classification (up to 14.2% in terms of F1 score). This means that learning the representations jointly across all time steps improves the overall performances, as we force embeddings continuity over time.

On the other hand, our approach is less efficient in edge related inference tasks[7]. We interpret those results as following: TemporalNode2vec captures better the global temporal structures than the local ones, as edge reconstruction/prediction tasks focus on pairs of nodes independently from the rest of the whole network, while node classification tasks consider the overall placement of the embeddings. Thus, our model seems to be more suitable for detecting communities and capturing their evolution. Concerning the efficiency of DynamicTriad at the edge prediction task, we suppose that it is due to the algorithm key idea, as DynamicTriad focuses on edges dynamics, i.e. how triads close/open. Finally, further investigation is needed to explain Node2vec performances on the edge reconstruction task.

Dataset	Algorithm	Node classification	Node class prediction	Edge reconstruction	Edge prediction
AMiner	DeepWalk	0.69884	0.67333	0.92628	0.75845
	Node2vec	0.77080	0.75846	**0.98362**	0.80619
	DynamicTriad	0.74432	0.73967	0.92363	**0.87294**
	TemporalNode2vec	**0.87999**	**0.85189**	0.86086	0.74929
Yelp	DeepWalk	0.29759	0.25357	0.94453	0.75710
	Node2vec	0.29747	0.26062	**0.98831**	0.79555
	DynamicTriad	0.25554	0.24698	0.93704	**0.91593**
	TemporalNode2vec	**0.31512**	**0.28118**	0.93214	0.81067
Tmall	DeepWalk	0.94436	0.68835	0.84765	0.68754
	Node2vec	0.96451	0.70008	**0.91305**	0.72787
	DynamicTriad	0.56336	0.53252	0.77821	**0.73260**
	TemporalNode2vec	**0.96632**	**0.77306**	0.77289	0.68043

Table 3: Models scores

4.5 Embeddings visualization

Figure 2 shows the projection of AMiner obtained embeddings (at the 13-th time step) on a 2D plane using the well known dimensionality reduction algorithm t-SNE [18]. We consider only three research fields, i.e. computer architecture, data mining and computing theory, as the other communities are much less represented. We can see that our embeddings succeed in discriminating vertices from different communities. Also, we notice a conglomerate of nodes from different communities: this can be explained by t-SNE side effects and the loss of information that dimensionality reduction implies.

[7] More tuning of the hyper parameters (especially p and q) may improve edge reconstruction/prediction tasks results.

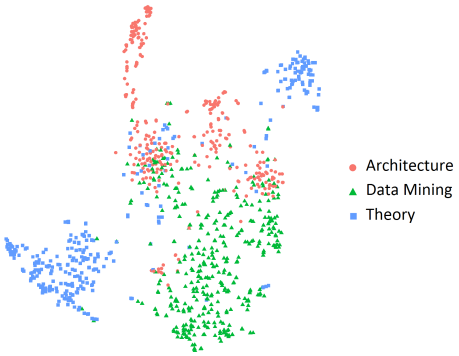


Fig. 2: 2D t-SNE projections

4.6 Hyper parameter analysis

We also analyze the impact of each hyper parameter on the overall method performance. From figure 3 we can see that the most sensitive parameters are q (walk return parameter) and θ (rare co-appearance/stability trade-off parameter).

Furthermore, there are two interesting observations we can notice. First, larger widow sizes give better results. This may confirm the idea that our model is more suitable for finding global temporal structures of networks, and consequently giving better results on node classification/class prediction tasks; as a matter of fact, while considering large values of w, vertices neighborhoods are not confined to their immediate neighbors. The second observation concerns the walk length l. We notice that smaller walks give better results. This may be due to the fact that small l values output more balanced walks over nodes. For example, if we imagine dense regions (in terms of edges and weights) in our temporal graph, a long random walk may be unable to step out of it.

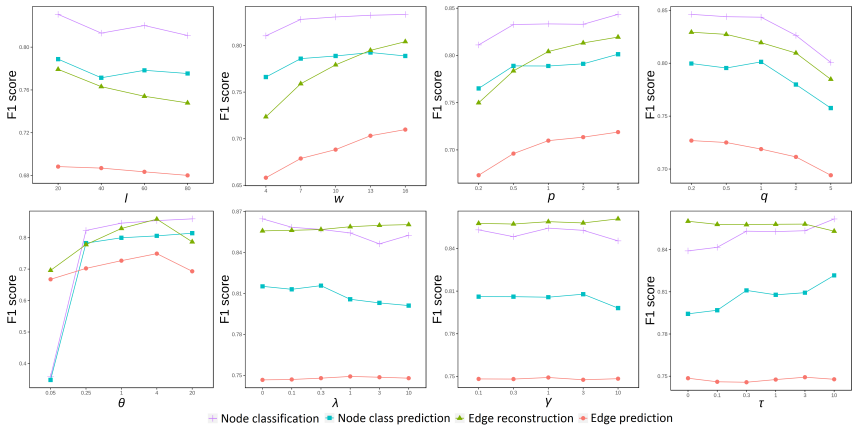


Fig. 3: Hyper parameter analysis

4.7 Alignment

Alignment is an important aspect in dynamic networks embedding, e.g. for visualization purposes where the position of nodes has to be stable in the coordinate referential. In order to challenge the different baseline methods alignment, embeddings are submitted to a test consisting on predicting nodes class change, based on the difference vector of consecutive embeddings: the ground-truth communities of the considered datasets have a semantic meaning that we reasonably assume to be constant over time (research fields, categories of products and businesses). Consequently, if alignment is respected, communities locations in the embedding latent space should be stable as well, or continuous at least. Thus, for good aligned methods, a node's trajectory in the embedding space gives information about its future class. Then, given the difference vector of a node's consecutive embeddings (input data), it should be possible to predict the class change of a node (target label). Table 4 shows the results of this experiment. We observe that the dynamic embedding methods outperform the static ones, proving alignment importance. However, the static methods present better results than expected, suggesting that they might include some inherent *normalization* mechanism of the output matrix, forcing alignment. Further investigation is needed to explain this point.

Algorithm	**AMiner**	**Yelp**	**Tmall**
DeepWalk	0.48921	0.40369	0.40710
Node2vec	0.48264	0.41124	0.43139
DynamicTriad	0.52823	**0.48088**	0.49889
TemporalNode2vec	**0.54446**	0.46089	**0.56838**

Table 4: Node class change prediction F1 score

4.8 Embedding dimension

The target latent space dimension is an important hyper parameter: the strength of embedding algorithms lies in inference performance, but also on how efficient their data encoding process is. For that purpose, we compare TemporalNode2vec to the considered baseline methods regarding the influence of the embedding dimension for the node classification task. Figure 4 shows the results of our experiment: while TemporalNode2vec affords to have good performances with a very limited number of features (from $d = 10$), the other approaches need a consequent embedding dimension and are unstable when dimension grows.

Furthermore, we notice a sort of saturation regarding the embedding dimension: it seems that 20 features are sufficient to describe nodes neighborhoods. This value may be the *intrinsic latent dimensionality* of our temporal graph, i.e. the minimal dimension of a latent space able to capture all structural information present in a graph. It is related to the nature of the input data and do not depend on its size [11]. To confirm this idea, we perform more experiments. Figure 5 shows the impact of the embedding dimension on the inference score

(node classification) on the three samples of different sizes previously described in table 2. We observe that the shapes of the curves are similar. In other words, the ratio between the information encoded at a dimension d and the *maximum* information we can capture do not depend on the sample size.

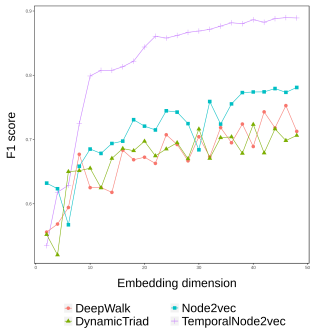


Fig. 4: Models data efficiency

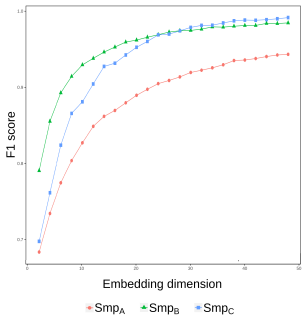


Fig. 5: Data efficiency on samples

5 Task-specific TemporalNode2vec

As seen above, it is possible to extract a small number of features describing accurately the input temporal graphs. This means that the dimensions of the latent embedding space found contain sufficient information to perform all the considered inference tasks, as embedding algorithms are agnostic regarding downstream machine learning tasks. However, at this point, the distribution of each task's useful information over the latent dimensions remains unknown. For example, in a simplistic scenario, some of the retained features may be relevant to edge reconstruction tasks and irrelevant to node classification ones. In such configuration, in order to perform node classification, one can settle for a smaller number of features. Upon this idea, we propose a variant of TemporalNode2vec we call TsTemporalNode2vec (task-specific TemporalNode2vec) : It is a semi-supervised embedding algorithm tied up to node classification and node class prediction tasks, made possible by a slight modification of TemporalNode2vec objective function.

5.1 TsTemporalNode2vec model

In addition to a list of timestamped interactions, TsTemporalNode2vec takes also in input a list of timestamped vertices labels $S = \{s_{i,c,t}\}$ where $s_{i,c,t}$ signifies that the vertex v_i belongs to the community c at t.

In order to conceive TsTemporalNode2vec, we modify the objective function of TemporalNode2vec by forcing proximity of embeddings for the nodes that

belong to the same community. This may be achieved by adding a term to the objective function:

$$L_{ts} = \sum_{(i,j,t)\in S_d} \|u_i(t) - u_j(t)\|^2 \tag{8}$$

where $S_{pos} = \{(i,j,t)\}$ is derived from S and signifies that the vertices v_i and v_j belong to the same community at t. However, introducing the term L_{ts} is not practical as it involves vectors rather than matrices, unlike the other terms. Also, the objective function optimizations described in section 3.4 would become invalid.

Instead, we choose to incorporate the supervision additional data into the $PPMI$ matrices: as they represent nodes temporal similarity matrices, one can increase the values corresponding to pairs of nodes belonging to the same community. We introduce S_{sc} and $SPMI$ matrices :

$$S_{sc}(v_i,\, v_j)_t = \begin{cases} 1 \text{ if } (i,j,t) \in S_{pos} \\ 0 \text{ otherwise} \end{cases} \tag{9}$$

$$SPMI(v_i,\, v_j)_t = PPMI(v_i,\, v_j)_t + \alpha \cdot m_t \cdot S_{sc}(v_i,\, v_j)_t \tag{10}$$

where m_t is the median of the non zero values of $PPMI_t$ and α is a hyper parameter controlling $PPMI$ values increase. We use m_t as a *normalization* so that α is of order unity.

On another note, it is relevant to notice that the timestamped labels S encompass also information about pairs of nodes belonging to different communities: in the same fashion as S_{pos}, we derive S_{neg} from S. It is then possible to incorporate S_{neg} in S_{sc} and $SPMI$ matrices :

$$S_{sc}(v_i,\, v_j)_t = \begin{cases} 1 \quad \text{if } (i,j,t) \in S_{pos} \\ -1 \text{ if } (i,j,t) \in S_{neg} \\ 0 \quad \text{otherwise} \end{cases} \tag{11}$$

$$SPMI(v_i,\, v_j)_t = max\Big(0,\, PPMI(v_i,\, v_j)_t + \alpha \cdot m_t \cdot S_{sc}(v_i,\, v_j)_t \Big) \tag{12}$$

That being set, we substitute $PPMI$ matrices by $SPMI$ ones in the objective function and its optimization expressed respectively in equations (6) and (7). The following steps of TemporalNode2vec remain unchanged.

5.2 Experiments

In order to challenge TsTemporalNode2vec performances, we submit its output embeddings to node classification and node class predictions tasks for AMiner dataset.

Labeling ratio
As TsTemporalNode2vec is a semi-supervised embedding model, one has to define and set the labeling ratio. For the datasets we consider, it is important to note that the nodes labels are not equal regarding the *amount of information* they provide: some nodes interact within all the time steps while others are active during very few ones. Therefore, it seems worthwhile to rate nodes contributions in terms of provided information. For each node, we define the quantity Q_i:

$$Q_i = \frac{q_i}{\sum\limits_{v_j \in V} q_j} \tag{13}$$

where q_i is the number of time steps where the vertex v_i is active. Then, a labeling ratio r can be obtained by a random sample of nodes RS_r satisfying :

$$\sum_{v_j \in RS_r} Q_j \approx r \tag{14}$$

Model tested values
Besides TemporalNode2vec hyper parameters, TsTemporalNode2vec has an additional one α, as described in section 5.1. For our experiments, we consider the values of $(l, w, p, q, \theta, \lambda, \gamma, \tau)$ giving the best performances on node classification and node class prediction tasks on AMiner dataset and perform a grid search over $(d, r, \alpha) \in \{2, 4, 7, 10, 15, 20, 25\} \times \{0.1, 0.25, 0.5\} \times \{0.1, 0.33, 0.67, 1\}$.

Results analysis and interpretation
Table 5 shows the results of the experiments. Overall, TsTemporalNode2vec improves the performances on node classification tasks. This is particularly the case for small embedding dimensions (28% gain in terms of F1 score metric for $d = 2$). This finding confirms our prior intuition stating that task-specific embedding captures as much relevant information as possible within the embedding latent dimensions. Also, as embedding dimension grows, the gap between task-specific and task-agnostic approaches scores tends to narrow: for large values of d, relevant information is captured by both methods.

Concerning the labeling ratio, the experiments show that setting r to 0.1 is sufficient to enhance the F1 score. This means that labelling 10% of the nodes helps to significantly improve the inference performances. On the other hand, it seems that larger values of r (0.25 and 0.5) do not bring more improvement to the performances (respectively, an average F1 score gain of 0.8% and 1.4% comparing to $r = 0.1$).

5.3 TsTemporalNode2vec application context

Task-specific temporal graph embedding can be useful in many cases. One can take advantage of total or partial metadata giving information about nodes communities. Also, when such data is not available, it is possible to perform a *soft*

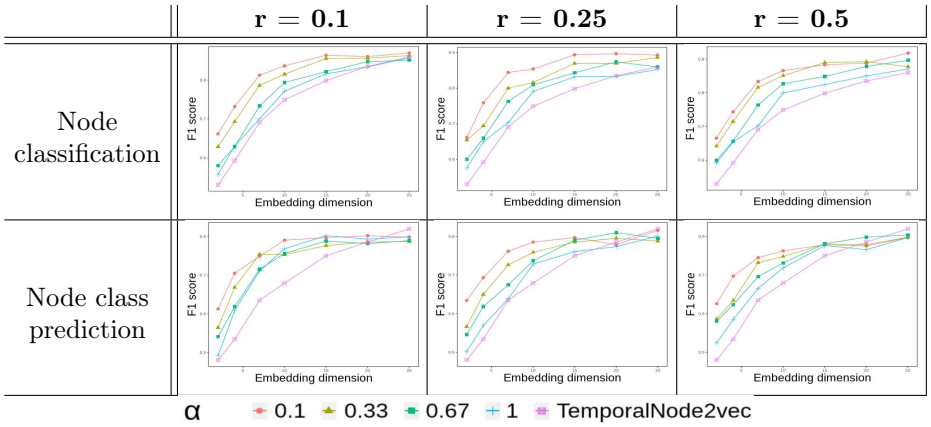


Table 5: TsTemporalNode2vec results analysis

labelling step, consisting of marking pairs of nodes probably belonging to the same community or probably belonging to different communities, based on some user-defined heuristic. In this case, task-specific embedding offers more flexibility, as it is possible to specify and highlight the desired communities or types of communities. One must nevertheless note that the hand-engineered co-belonging rule is to be considered as a method to reproduce the ground-truth communities partially but reliably. It should then privilege precision rather than recall.

6 Conclusion

In this paper, we presented TemporalNode2vec, a novel approach of dynamic networks embedding. We compared our algorithm to several state-of-the-art techniques and showed its efficiency in node classification tasks. Furthermore, we demonstrated the data encoding efficiency of our approach, as we showed that, in a real-world dataset consisting of more than 51k nodes, a representation of 20 features is sufficient for gathering relevant temporal and structural characteristics of the network. Moreover, we introduced and evaluated TsTemporalNode2vec, a task-specific variant. We also proved its advantages in terms of data-efficiency. Further work tied up to embeddings exploitation for different purposes (dynamics analysis and prediction, visualization) is under way.

References

1. Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., Smola, A.J.: Distributed large-scale natural graph factorization. In: Proceedings of the 22nd international conference on World Wide Web. pp. 37–48. ACM (2013)
2. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Advances in neural information processing systems. pp. 585–591 (2002)
3. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35(8), 1798–1828 (2013)
4. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. In: Social network data analytics, pp. 115–148. Springer (2011)
5. Bojchevski, A., Günnemann, S.: Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. arXiv preprint arXiv:1707.03815 (2017)
6. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM international on conference on information and knowledge management. pp. 891–900. ACM (2015)
7. Cao, S., Lu, W., Xu, Q.: Deep neural networks for learning graph representations. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
8. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: Harp: Hierarchical representation learning for networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
9. Firth, J.R.: A synopsis of linguistic theory, 1930-1955. Studies in linguistic analysis (1957)
10. Fortunato, S.: Community detection in graphs. Physics reports 486(3-5), 75–174 (2010)
11. Granata, D., Carnevale, V.: Accurate estimation of the intrinsic dimension using graph distances: Unraveling the geometric complexity of datasets. Scientific reports 6, 31377 (2016)
12. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864. ACM (2016)
13. Haddad, M., Bothorel, C., Lenca, P., Bedart, D.: Temporalnode2vec: Temporal node embedding in temporal networks. In: International Conference on Complex Networks and Their Applications. pp. 891–902. Springer (2019)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
15. Kulkarni, V., Al-Rfou, R., Perozzi, B., Skiena, S.: Statistically significant detection of linguistic change. In: Proceedings of the 24th International Conference on World Wide Web. pp. 625–635. International World Wide Web Conferences Steering Committee (2015)
16. Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. In: Advances in neural information processing systems. pp. 2177–2185 (2014)
17. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. Journal of the American society for information science and technology 58(7), 1019–1031 (2007)
18. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research 9(Nov), 2579–2605 (2008)
19. Mahdavi, S., Khoshraftar, S., An, A.: dynnode2vec: Scalable dynamic network embedding. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 3762–3765. IEEE (2018)

20. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
21. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: Companion of the The Web Conference 2018 on The Web Conference 2018. pp. 969–976. International World Wide Web Conferences Steering Committee (2018)
22. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1105–1114. ACM (2016)
23. Pearson, K.: The problem of the random walk. Nature 72(1867), 342 (1905)
24. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
25. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 990–998. ACM (2008)
26. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research 11(Apr), 1201–1242 (2010)
27. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234. ACM (2016)
28. Yao, Z., Sun, Y., Ding, W., Rao, N., Xiong, H.: Dynamic word embeddings for evolving semantic discovery. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. pp. 673–681. ACM (2018)
29. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
30. Zhu, L., Guo, D., Yin, J., Ver Steeg, G., Galstyan, A.: Scalable temporal latent space inference for link prediction in dynamic social networks. IEEE Transactions on Knowledge and Data Engineering 28(10), 2765–2777 (2016)
31. Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., Wu, J.: Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2857–2866. ACM (2018)

7 Declarations

7.1 Availability of data and material

All used datasets are publicly available. Their repositories are mentioned in this manuscript.

7.2 Competing interests

None of the authors has any competing interest.

7.3 Funding

This article presents work produced as part of a PhD intern thesis, financed by DSI Global Services (41 avenue du Général Leclerc, 92350 Plessis-Robinson, France) and ANRT (Association Nationale de la Recherche et de la Technologie, 33 Rue Rennequin, 75017 Paris, France)

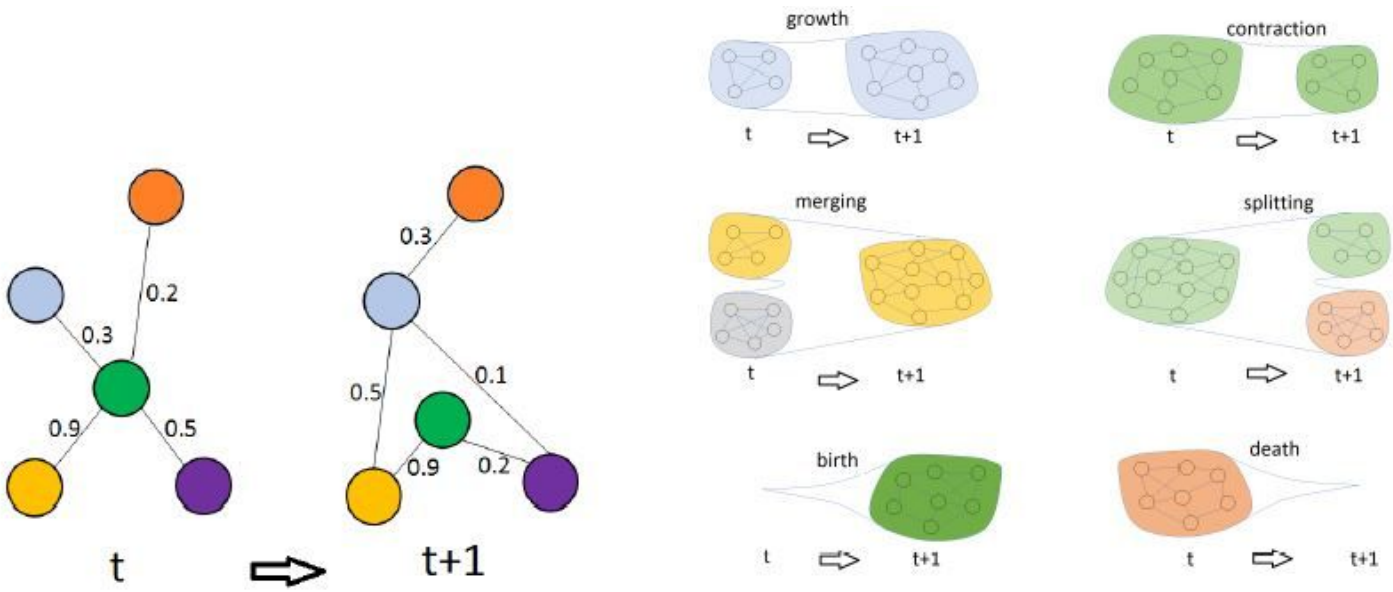
7.4 Authors' contributions

- **Mounir HADDAD**: Principal author.
- **Cécile Bothorel**: Principal supervisor.
- **Philippe Lenca**: team research supervisor.
- **Dominique Bedart**: Company supervisor.

7.5 Acknowledgments

Authors of this paper are the only contributors in the presented work. There is no other person or institution to acknowledge.

Figures



(a) Local scale

(b) Global scale

Figure 1

Network evolution

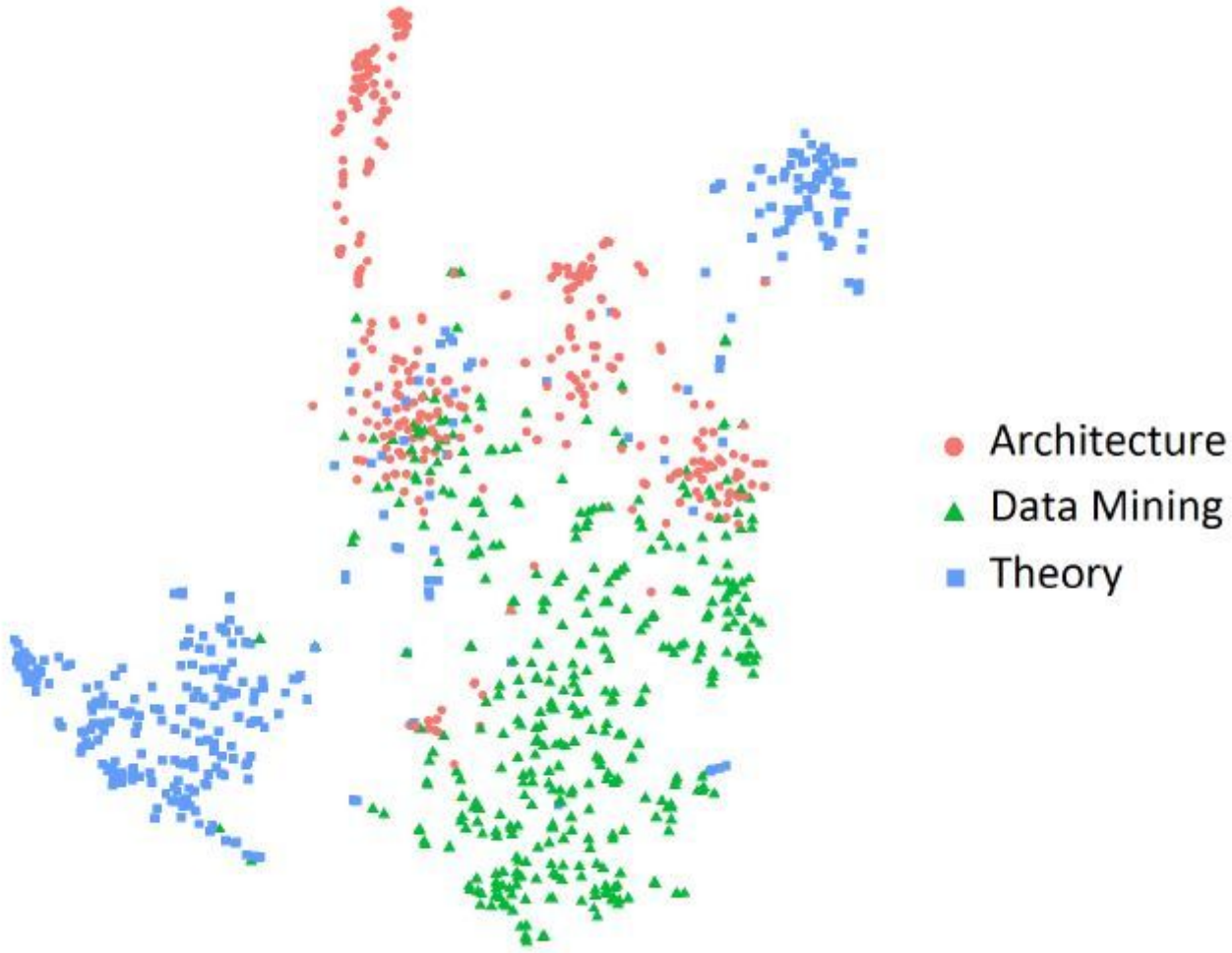


Figure 2

2D t-SNE projections

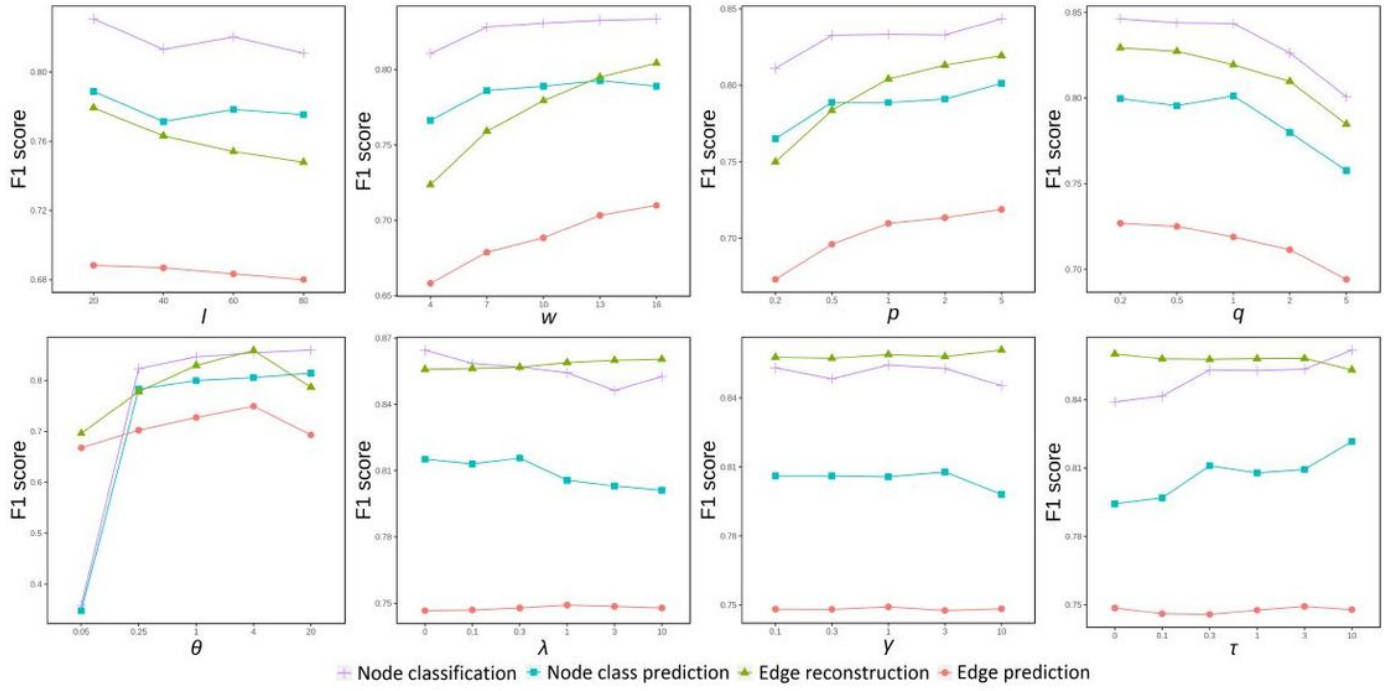


Figure 3

Hyper parameter analysis

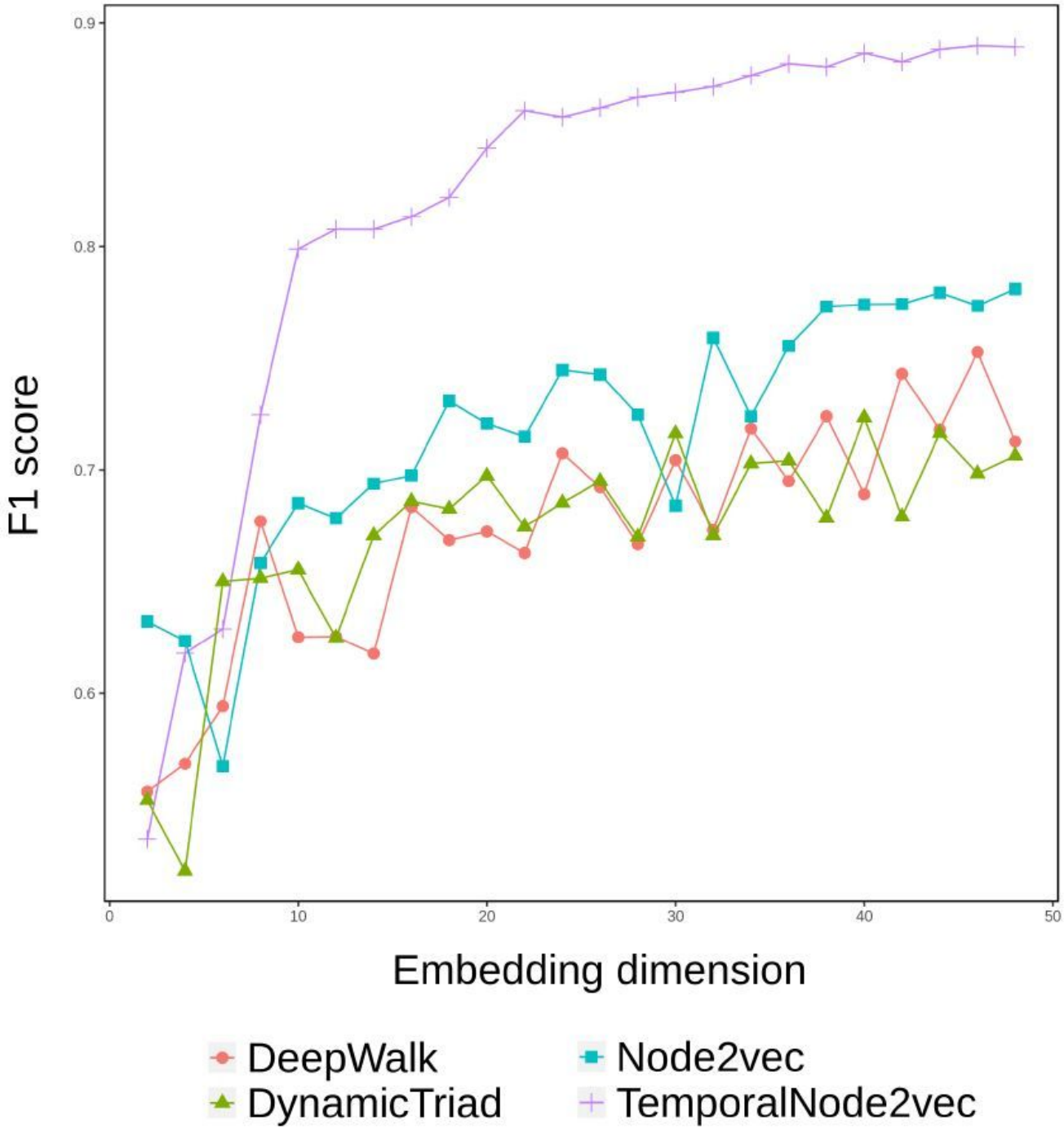


Figure 4

Models data efficiency

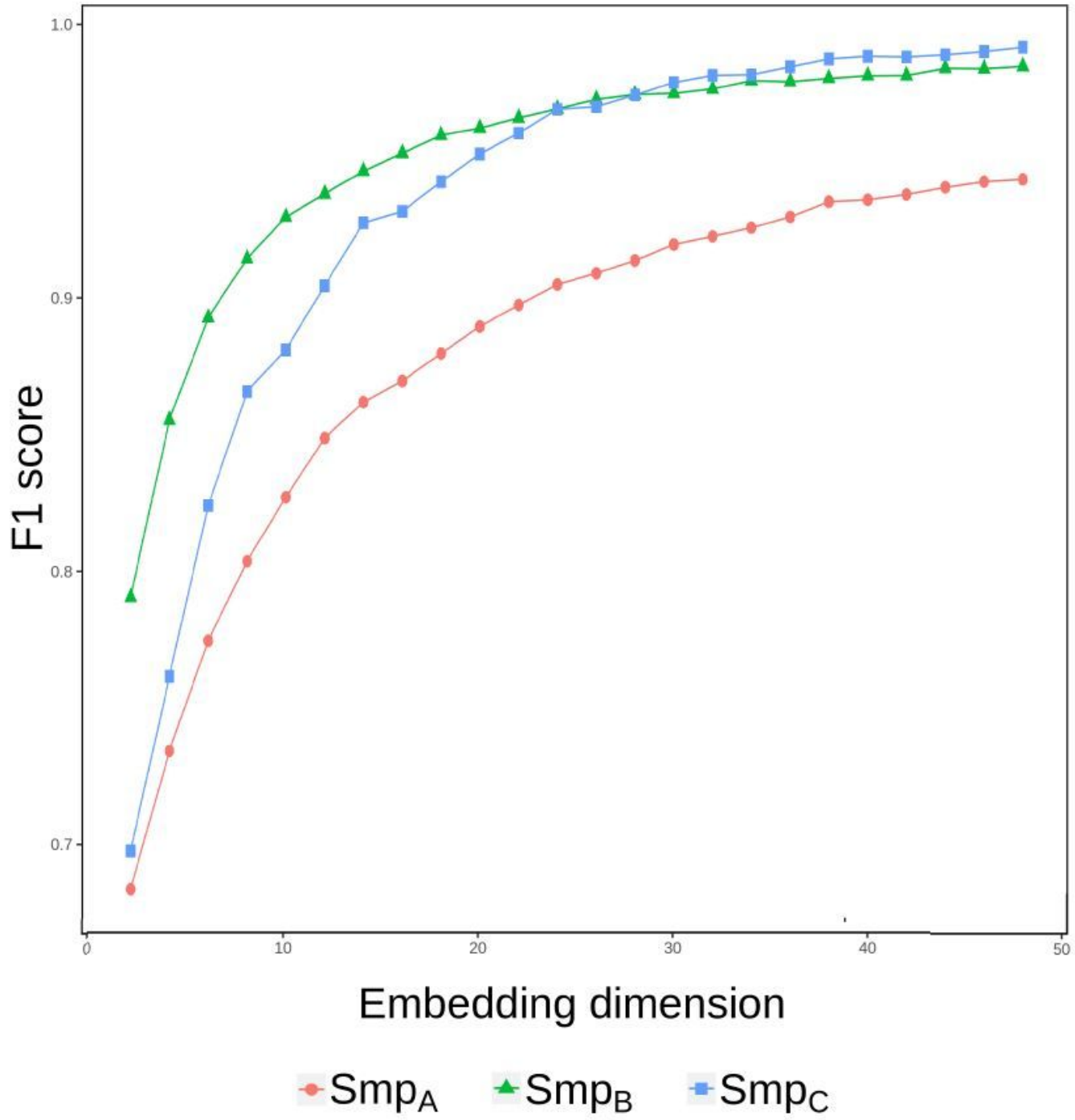


Figure 5

Data efficiency on samples