Trajectory Embedding

WANG XIANG CHUN*

We have 10,000 geographic trajectories without time information and thier LCSS distance matrix. We need to find a embedding vector which dimension is much smaller than the average length of the trajectory for each trajectory. Moreover, we need to make the embedding vector retain the information of the original trajectory as much as possible, which were be evaluated by hit rate. We used pca[8], deepwalk[6],and vgae[3] to complete this task respectively, but the result of vgae is not good. We tried to analyze the reasons: characteristics of the trajectory data mostly depend on the time series trajectory points, so the node characteristics are not highly correlated.

Additional Key Words and Phrases: embedding, graph neural networks, trajectory, LCSS

ACM Reference Format:
Wang Xiang Chun. 2021. Trajectory Embedding. 1, 1 (September 2021), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We have 10,000 geographic trajectories without time information and thier four kinds of distance matrices (all are symmetric matrices), they are Fréchet Distance, Hausdorff Distance, Dynamic Time Wrapping (DTW), and Longest Common SubSequence (LCSS). With this information, we need to find a embedding vector which dimension is much smaller than the average length of the trajectory for each trajectory. Moreover, we need to make the embedding vector retain the information of the original trajectory as much as possible, which were be evaluated by hit rate:

$$hit\ rate = \frac{hits}{trajs} \tag{1}$$

According to the selected distance matrix, for each trajectory, we take the highest similarity 10 trajectory as the test set.At the other hand, we calculate the distance of the embedding vector corresponding to trajectory one by one. Then perform a nearest neighbor search for each trajectory according to it, and establish a top-50 list for each trajectory. The hits in formula[1] indicates the number of trajectories whose all similar trajectories in the test set appear in the top-50 list.

In this project, we will use pca[8], deepwalk[6],and vgae[3] to complete this task respectively base on LCSS distance matric and analysis their performances at the end.

*sjtu-summer-project

Author's address: Wang Xiang Chun, stephenwang0205@outlook.com.

XXXX-XXXX/2021/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Classical techniques for dimensionality reduction include principal component analysis (PCA)[8] and multidimensional scaling (MDS)[4]. Both methods are capable of capturing linear structural information, but fails to discover the non-linearity within the input data.

Another dimensionality reduction technique is graph embedding. First, we convert the data set into a graph. The central idea of graph embedding is to find a mapping function which converts each node in the network to a low-dimensional latent representation. These representations can then be used as features for common tasks on graphs such as classification, clustering, link prediction, and visualization. In 2014, inspired by word2vec [5], Perozzi proposed DeepWalk [6], which combine graph embedding and deep learning. These methods generally use a specific strategy to sample the nodes in the graph to learn the similarity of the nodes in the graph.

In 2013, on the basis of Graph Signal Processing[7] , Bruna proposed for the first time a convolutional neural network based on Spectral-domain and spatial domain.[1], which calls graph convolutional neural network (gcn). Afterwards, there are many new methods bases on that.

3 PROBLEM DEFINITION

The purpose of this project is to find a embedding vector which dimension is much smaller than the average length of the trajectory for each trajectory. The result wiil be evaluated by hit-rate [formula]

4 ALGORITHM1 PRINCIPAL COMPONENTS ANALYSIS (PCA)

The Goal of principal component analysis computes the most meaningful basis to re-express a noisy, garbled data set. Therefore, the main question is: We are going to find another basis, which is a linear combination of the original basis (it's an assumption), that best reexpresses our data set?

4.1 Principle

4.1.1 Change of Basis. Assuming that the original data set has n samples and m features, so it can be recorded as a matrix X of order $m \times n$, and each column vector is a sample In the above example, let Y be another $m \times n$ order matrix obtained by X through linear transformation P.

$$Y = PX \tag{2}$$

- p_i represents the row vectors of P
- x_i represents the row vectors of X
- y_i represents the row vectors of Y
- the row vectors of P $\{p_1, p_2, ..., p_m\}$ is a new basis

4.1.2 VARIANCE AND THE GOAL. Now, we are going to answer one of the most critical questions: How to choose the best base ? An intuitive view is: I hope that the projected values after projection are as scattered as possible, because if they overlap, some samples will disappear. Of course, this can also be understood from the perspective of entropy. The greater the entropy, the more information it contains. On the other hand, in order to make the two variables represent as much original information as possible, we hope that there is no linear correlation between them; otherwise, there must be duplicated information.

We define a new matrix $Y_{m\times n}$ whose each column represents a sample and whose each row $\{y_1, y_2..., y_m\}$ represent a feature. Then we can get the covariance matrix of Y

$$Y = \begin{bmatrix} y_1 \\ . \\ . \\ y_m, \end{bmatrix} D_y = \frac{1}{n} YY^T \tag{3}$$

Specifically, the ij^{th} element of D_y is the dot product between the vector y_i and the vectoe y_j, which means the covariance of the two variables (when $i = j$ means the variance of the variable). The absolute value of the diagonal elements should be as large as possible, so we will arrange them in descending order of the corresponding variance. On the other hand, the absolute value of non-diagonal elements should be as small as possible, so we want D_y to be a diagonal matrix.

$$D = \frac{1}{m} YY^T = \frac{1}{m}(PX)(PX)^T = \frac{P}{X} X^T P^T = P(\frac{1}{m} XX^T)P^T = PCP^T \tag{4}$$

We can see that: The first k rows of P are the base to find. Multiplying the matrix composed of the first k rows of P by X will reduce X from the N-dimensional to the K-dimensional and satisfy the above optimization conditions. And we know that C is a matrix (which is a $m \times m$ symmetric matrix), so it has m unit orthogonal eigenvector $\{e_1, e_2, ..., e_m\}$

$$E^T CE = \Lambda = \begin{bmatrix} \lambda_1 & & & & & \\ & \lambda_2 & & & & \\ & & . & & & \\ & & & . & & \\ & & & & . & \\ & & & & & \lambda_m \end{bmatrix} \tag{5}$$

4.1.3 *Summary of Assumptions.*

- Linear
  Linearity reduces the problem to a basis transformation.
- Large variance corresponds to the main structure
  This assumption also means that the SNR of the data is very high. Therefore, the principal components with larger variance represent the main structure, and the principal components with smaller variance represent noise. This is a strong assumption, but this assumption is sometimes inappropriate.
- The principal components are orthogonal to each other
  This hypothesis indicates that PCA can be solved by some decomposition methods in linear algebra

4.2 Steps

- Process the data so that the average value is zero.
- Calculate the covariance matrix.
- Calculate the eigenvalues and eigenvectors of the covariance matrix.
- Sort eigenvalues.
- Keep the eigenvectors corresponding to the first N largest eigenvalues.
- Convert the original features to the new space constructed by the N feature vectors obtained above.

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$
Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

Fig. 1. algorithm1

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

Fig. 2. algorithm2

5 ALGORITHM2 DEEPWALK

5.1 Random Walk

Graph is not a data form like image or natural language, which are structured data. In this case, how to use deep learning to complete the task of graph embedding? One of the most simple ideas is to convert graph data (topological structure data) into structured data. The method of transformation is random walk.

The steps of Random walk are: pick a starting point in the graph, and then sample m (which is called walklength) nodes in the graph. The data chain connected by the nodes can be obtained to form a structured data, which can be applied to common neural networks.

5.2 Deepwalk

In the paper[6], two algorithm pseudo-codes are introduced. The entire DeepWalk algorithm consists of two parts, one is the generation of random walk, and the other is the parameter update, as shown in Figure 1 and Figure 2.

6 ALGORITHM3 VARIATIONAL GRAPH AUTO-ENCODERS (VGAE)

6.1 VE and VAE

In 1986, Rumelhart proposed the concept of autoencoder (AE) and used it for high-dimensional data processing (dimensionality reduction), which promoted the development of neural networks.

Autoencoder is an unsupervised learning algorithm that uses back propagation algorithm in order to make the target value equal to the input value.

The autoencoder framework contains two parameterized functions. The first function f_θ is called encoder, which extracts the feature vector $h = f_\theta(x)$ from the original input x. The second function g_θ is called the decoder, which is a mapping from the feature space to the input space, which produce a reconstruction $r = g_\theta(h)$. By making the reconstruction produced by the decoder as similar as possible to the original data as input, ; that is, trying to obtain the minimum reconstruction loss $L(x, r)$ on the training set, . There, the parameter set θ of the encoder and the decoder is obtained through training at the same time. Good generalization means that a lower reconstruction loss can be obtained in the test set, and the feature vector extracted by the encoder is the representation learned by the autoencoder framework. In conclusion, the training purpose of the autoencoder framework is to find the value of a set of parameters θ to minimize the reconstruction loss. This parameter θ is the featue vector(dimensionality reduction vector) we want.

An important development of AE is VAE (Variational Auto-Encoder) [2]. It can solve a shortcoming of AE : The second function of AE g_θ can only guarantee to restore z generated by x to x If we generate a z randomly, as the input of g_θ, we often won't get a valid output.

The VAE can make z generated by f_θ conforms to a specified distribution as much as possible, such as a multidimensional normal distribution with a standard deviation of 1. Then then we only need to sample z from this distribution, and we can get a valid output through g_θ. Specifically, this is achieved through a reparameterization trick.[2]

6.2 GAE

6.2.1 *Related Variables.*

- The graph is represented by $G = (V, E)$, where V represents the set of nodes, and E represents the set of edges
- A : Adjacency matrix
- D : Degree matrix
- d : Feature dimension of node
- $X \in \mathbb{R}^{N \times d}$: Feature matrix of nodes
- f : Embedding dimension
- $Z \in \mathbb{R}^{N \times f}$: Node embedding

6.2.2 *Encoder.*

GAE uses GCN as an encoder to get the latent representations (node embedding) of nodes. This process can be expressed in a short line of formula:

$$Z = GCN(X, A) = GCN(X, A) = AReLU(\hat{A}XW_0)W_1 \tag{6}$$

Consider GCN as a function, and then take X and A as input. Input into the GCN function, and output is $Z \in \mathbb{R}^{N \times f}$. Z represents the embedding of all nodes. The entire encoder has only two layers, and each layer uses the first-order approximation of the Chebyshev polynomial as the convolution kernel to process the data. It can be seen that except for the initial input X, which is the feature matrix representing the node, the remaining parameters are all objects that need to be learned. In short, here GCN is equivalent to a function that takes node features and adjacency matrix as input and node embedding as output, and the purpose is only to get embedding.

6.2.3 *Decoder.*

$$\hat{A} = \sigma(ZZ^T) \tag{7}$$

GAE uses inner-product as a decoder to reconstruct the original graph. The output of decoder is the reconstructed adjacency matrix, and the loss function can be constructed based on the reconstructed adjacency matrix and the adjacency matrix of original graph.

6.2.4 Loss Function.
The adjacency matrix represents the structure of the graph, so the reconstructed adjacency matrix should be as similar as possible to the original adjacency matrix. Therefore, GAE uses cross entropy as the loss function during training.

$$\mathcal{L} = -\frac{1}{N}\sum y \log \hat{y} \log(1-\hat{y}) \tag{8}$$

In the above formula, y represents the value (0 or 1) of an element in the adjacency matrix A , and $\hat{y}$ represents the value (between 0 and 1) of an element in the reconstructed adjacency matrix $\hat{A}$. It can be seen from the loss function that the adjacency matrix is expected to be as similar to the original adjacency matrix as possible.

6.3 VGAE

In GAE, once W_0 and W_1 in GCN are determined function, which means given X and A, the output Z is defined.

However, in VGAE, Z is no longer obtained by a certain function, but sampled from a multi-dimensional Gaussian distribution. To be more specific, we first determine a multi-dimensional Gaussian distribution through GCN, and then sample from this distribution to get Z. Below is a brief description of this process.

6.3.1 Determine the Mean and Variance.
The Gaussian distribution can be uniquely determined by the mean and the variance. Therefore, to determine a Gaussian distribution, you only need to know the mean and variance. VGAE uses GCN to calculate the mean and the variance respectively. W_0 in GCN_μ and GCN_σ is shared but W_1 is different, so the subscript is used to distinguish.

$$\mu = GCN_\mu(X, A), \quad \log \sigma = GCN_\sigma(X, A) \tag{9}$$

6.3.2 Sampling. Now that the mean and variance have been obtained (to be precise, it should be the mean vector and covariance matrix), Z can be obtained by sampling. However, the sampling operation cannot provide gradient information; that is to say, backpropagation cannot be used in the sampling operation to calculate the gradient, so it is impossible to update W_0 and W_1. The solution is to reparameterization.

Reparameterization
If ϵ obeys $N(0, 1)$, then $z = \mu + \epsilon\sigma$ obeys $N(\mu, \sigma^2)$. Therefore, you can first sample an ϵ from the standard Gaussian, and then calculate z through $\mu + \epsilon\sigma$, so that the expression of z is clearly visible, and the gradient information is also available.

6.3.3 Loss Function.
The decoder of VGAE is also a simple inner-product, which is no different from the decoder of GAE.

7 RESULTS

7.1 pca

Pca is the most convenient model to use. We could directly use the distance matrix as input, and finally got the 64-dimensional embedding vector. The hit equaled to 0.1059, as the baseline of vgae.

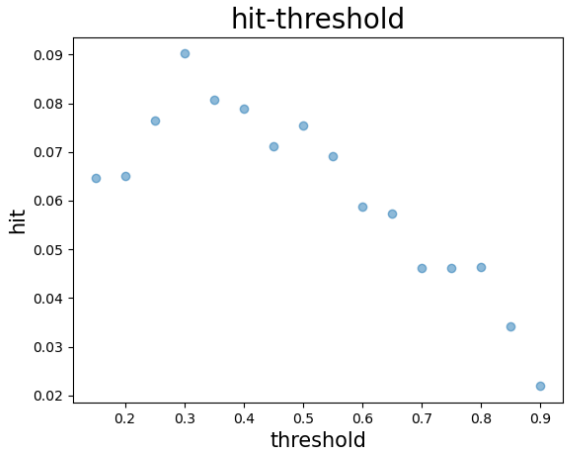

Fig. 3. hit-threshold

7.2 deepwalk

The initial effect of deepwalk is not good, and the model ran too long, so there was no more testing.

7.3 vgae

The first step is to turn the LCSS distance matrix into a similarity matrix. The range of LCSS is 0-1. We used 10,000 trajectories as the nodes of the graph. If the LCSS between the two trajectories is less than the threshold, there is an edge between the two corresponding nodes. Then we constructed a graph according to this rule.

At the beginning, we tried different thresholds every 0.5 to find some threshold with a better effect(like in fighure 3), and then tried to use [] to find a better threshold. However we found the best result whose hit only equaled to 0.0902, when the threshold equaled to 0.3.

In view of the poor effect of a single threshold, we tried to use embedding vectors with different thresholds. In the experiment, embedding vectors with thresholds of 0.25, 0.30, 0.35, and 0.40 were used comprehensively, but the effect did not improve significantly, which only slightly improved when hit equaled to 0.0951 .

8 CONCLUSION

Analysis for the poor results: Graph network learning is applicable to data with a high degree of correlation between graph network structure and node characteristics, such as typically cora data, traffic flow data, and recommendation system data. The characteristics of the trajectory data mostly depend on the time series trajectory points, so the node characteristics are not highly correlated.

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.
[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
[3] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.
[4] Joseph B Kruskal. *Multidimensional scaling*. Number 11. Sage, 1978.
[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[7] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[8] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.