ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

Implementation of Graph Embedding Algorithms in Node Classification for Wiki

Tong Wu*, Ying Li

New Media College, Communication University of Zhejiang, Hangzhou, China
**Corresponding author email: 516868946@qq.com*

Abstract: *As wiki data integration has become one of the most popular and freely available data sources on the Internet, more and more domestic and foreign researchers are studying and analyzing wiki data in-depth, However, when classifying nodes in wiki data sets, we encountered poor relevance of node classification, which led to the problem of reduced data quality, accuracy, and reliability. In response to existing problems, we propose to use graph embedding related algorithms to classify wiki data sets. To improve the accuracy of the classification task, we have implemented Deepwalk (Deep Walk Random Algorithm), Node2vec, and SDNE (Structural Deep Network Embedding) The algorithm is tested on the wiki data set. The final experimental results show that the Graph Embedding algorithm has a good effect on the node classification of graph structure data, and the SDNE algorithm is the best inaccurate.*

Keywords: *Wiki dataset, Graph Embedding, Node Classification*

1. Introduction

Wikidata is a collaboratively editable knowledge base. It is the first project of the Wikimedia Foundation Inc after the Wikiacademy in 2006. It was officially launched on October 30, 2012. It is available for Wikipedia in more than 280 languages. With data support, Wikipedia is also a powerful tool for user information acquisition.

Nowadays, with the rapid development of Internet technology, the growth rate of data information has increased exponentially, and the amount of information in Wikidata has also increased rapidly. However, as the amount of data has increased, the quality of data has gradually become uneven. Wikidata is an auxiliary database, the collection of data items is usually automatically captured by machines, and the correlation of node classification is poor, which leads to low accuracy and reliability in data quality. For machine statistics errors, 830 unique sources are included in the 260 personal items. There are only 70 available sources, accounting for only 8% [1]. As for the sources of data, most types of data sources come from web pages.

To improve the accuracy of data quality, we tested the relevant algorithms of graph embedding to select the optimal algorithm. These algorithms are: Deepwalk (deep walk random algorithm) [3], node2vec [4], and SDNE [5] algorithm

2. Related work

Wikipedia is based on wiki technology and emphasizes freedom, free, and open content. Anyone can edit any entry in the encyclopedia. The auxiliary database used by Wikipedia is Wikidata. Wikidata has the characteristics of open, collaborative editing, multilingual, structured, etc. It is released under the Creative Commons "CC0" license agreement. It allows data reuse while allowing users to copy, modify, distribute, and perform data node classification. Wikidata uses structured forms to collect data and supports Wikimedia projects and third parties for these data. And enables the computer to easily understand and process these data, by using OWL, RDF, SKOS, etc. to describe entities, entries, attributes, and their relationships [6], secondly, it is also a wiki in more than 280 languages Encyclopedia provides support for data sources. At the same time, it integrates unstructured data into structured data through machine algorithms, which is convenient for queries and uses in various fields. In addition, in order to better implement downstream tasks such as recommendation and prediction, Wikipedia needs to classify graph data nodes. Node classification refers to the category corresponding to certain nodes in a given graph, so as to predict which category a node born without a label belongs to. This task is also called

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

semi-supervised node classification.

In node classification, traditional machine learning algorithms include K-means clustering algorithm, KNN nearest neighbor classification algorithm, SVM, random forest algorithm and so on. The K-means clustering algorithm is one of the commonly used partition clustering algorithms [7]. The samples are divided by selecting a similarity measure to make the data in the same category as similar as possible [8]. The KNN algorithm is mainly based on user-based recommendations when applied to the recommendation system. It first uses the principle of "closely related users have the same hobbies". Later, some people apply the KNN algorithm to item-based collaborative filtering. Recommended in the system [9].

However, traditional machine learning algorithms are not good at classifying the data types of graph structures, and Wikidata uses graph structures, which results in traditional machine learning algorithms not being able to perform high-efficiency and high-efficiency on Wikidata. Accurate node classification. To this end, we introduce graph embedding to classify data nodes. Graph Embedding is a process of mapping graph structure data into low-dimensional dense vectors so as to capture the topological structure of the graph, the relationship between vertices and vertices, and other information.

Compared with traditional machine learning, Graph Embedding maintains richer calculation methods for graph structure data with multiple entries and attributes like Wikidata. At the same time, graph embedding can compress the data and reduce the dimensionality of the data. Therefore, Graph Embedding's processing of graph structure data such as Wikidata is more flexible and faster than traditional machine learning algorithms.

3. Methodology

3.1 DeepWalk

DeepWalk is the first algorithm to apply the ideas in NLP to Graph Embedding. The algorithm consists of two main components; first a random walk generator, and second, an update procedure. [10]It focuses on solving the related problems from graph sampling to how the sequence is performed. The main idea is to perform a "random walk (Random Walk)" to generate a large number of item sequences, and then use these item sequences as training samples as the input of the Skip-gram model. Finally we can get the Embedding of the items. The following figure shows the process of DeepWalk in a graphical way:

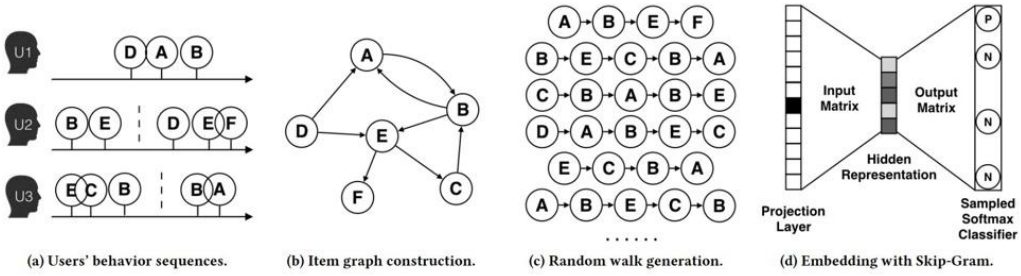

Figure 1. the process of DeepWalk

a) *User's behavior sequences:* According to the data set, we can establish the user's original behavior sequence.

b) *Item graph construction:* We establish the item correlation diagram on a basis of the user's original behavior sequence. For example, user 1 (U1) has purchased items D and then he has purchased A and B successively, so a three-point directed edge of D, A, and B is generated. If there are multiple identical directed edges subsequently, then the weight of these directed edges will increase. When the user's behavior sequence has been converted into a directed graph completely, we can construct the item correlation graph successfully.

c) *Random walk generation:* Random walk in the directed graph b. We can randomly select the starting point and direction to generate a new item sequence.

d) *Embedding with Skip-Gram:* Inputting the new item sequence into the Skip-gram model which can get the Embedding vector.

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

In the DeepWalk algorithm, the most important step is the third step. In this step, what needs to be formally defined is the jump probability of random walk. After reaching the node v_i, the next step is traversing the probability of its neighbor node v_j. The probability of jumping from v_i to v_j is as follows:

$$P(v_j \mid v_i) = \begin{cases} \frac{M_{ij}}{\sum_{j \in N_+(v_i)} M_{ij}}, & v_j \in N_+(v_i), \\ 0, & e_{ij} \notin \mathcal{E}, \end{cases} \tag{1}$$

In this formula(1), $N_+(vi)$ represents the set of nodes connected by all the outgoing edges of node v_i, and M_{ij} represents the weight of the edge connected from v_i to v_j. It can be seen that the jump probability of the original DeepWalk algorithm is the ratio of the weight of the jump edge to the sum of the weights of all related outgoing edges.

3.2 Node2vec

Researchers at Stanford University had supposed the Graph Embedding model named Node2vec in 2016. This model adjusts the weight of random walk to make the result of the embedding vector of the node more inclined to reflect the homogeneity and structure of the network.

Homogeneity: It means that the embedding vectors of nodes that are close together should be similar. As shown in figure2, the embedding vectors of nodes s_1, s_2, s_3, and s_4 which were connected to a node should be similar. So that the embedding vector can reach the homogeneity of the network, it is necessary to make random walks more inclined to DFS. Because DFS is more likely to reach distant nodes through multiple jumps, so that the walk sequence is concentrated in a larger internal set, and the nodes within a set have higher similarity, thus expressing the homogeneity of the graph.

Structural: The embedding vectors of nodes with similar structures should be similar. As shown in the figure 2, the embedding vectors of s_6 with similar structures to node u should be similar. In order to express the structure, random walk needs to be more inclined to BFS. As a result of fact, BFS will walk more in the neighborhood of the current node which is equivalent to scanning the network structure of the current node, so that the embedding vector can describe the neighborhood of the node's structure information. For instance, in Figure 2, we observe nodes u and s1 belonging to the same tightly knit community of nodes, while the nodes u and s6 in the two distinct communities share the same structural role of a hub node.[11]

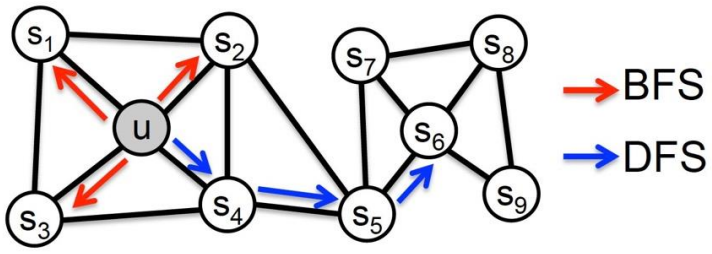

Figure 2 BFS and DFS search strategies from node u (k = 3).

In the node2vec model, the BFS and DFS tendencies are also dominated by controlling the jump probability among nodes. We define a 2nd order random walk with two parameters p and q which guide the walk: Consider a random walk that just traversed edge (t, v) and now resides at node v (Figure 3). The walk now needs to decide on the next step so it evaluates the transition probabilities π_{vx} on edges (v, x) leading from v. We set the unnormalized transition probability to $\pi_{vx} = \alpha_{pq}(t,x) \cdot \omega_{vx}$.[11]

$$\alpha_{pq}(t,x) \begin{cases} \frac{1}{p} & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ \frac{1}{q} & if\ d_{tx} = 2 \end{cases} \tag{2}$$

In the formula(2), dtx denotes the shortest path distance between nodes t and x. Note that dtx must be one of {0, 1, 2}, and hence, the two parameters are necessary and sufficient to guide the walk. Intuitively, parameters p and q control how fast the walk explores and leaves the neighborhood of starting node u.

Return Parameter p: Parameter p controls the probability of revisiting a node in the walk. Setting it

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

to a high value (> max(q, 1)) ensures that we are less likely to sample an already- visited node in the following two steps (unless the next node in the walk had no other neighbor). What's more, if p is low (< min(q, 1)), it would lead the walk to backtrack a step (Figure 3) and this would keep the walk "local" close to the starting node u. Actually, the algorithm pays more attention to expressing the structure of the network.

In-out Parameter q: Parameter q controls the probability of random walk to a distant node. if q > 1, the random walk is biased towards nodes close to node t. Such walks obtain a local view of the underlying graph with respect to the start node in the walk and approximate BFS behavior in the sense that our samples comprise of nodes within a small locality.

On the contrary, if q < 1, the walk is more inclined to visit nodes which are further away from the node t. Actually, the algorithm pays more attention to expressing the homogeneity of the network.

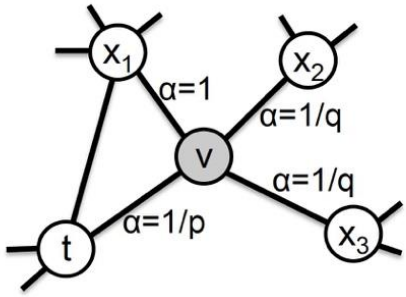

Figure 3 Illustration of the random walk procedure in node2vec. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

3.3 SDNE

The SDNE (Structural Deep Network Embedding) algorithm was published in the 2016 KDD conference. It uses an auto-encoder structure to optimize the first-order and second-order similarities at the same time (LINE is optimized separately). By jointly optimizing them in the proposed semi-supervised deep model, SDNE can preserve the highly-nonlinear local-global network structure well and is robust to sparse networks. [12]The SDNE model structure is as follows:

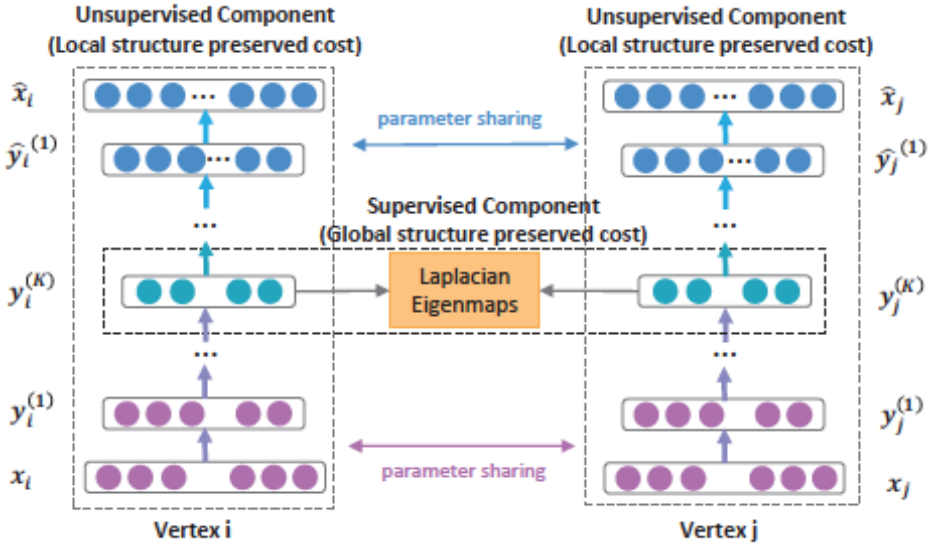

Figure 4 The framework of the semi-supervised deep model of SDNE

The left side of the figure is the structure of an auto-encoder. The input and output are the adjacency matrix and the reconstructed adjacency matrix. By optimizing the reconstruction loss, the global structural characteristics of the vertices can be preserved.

The $y_i^{(K)}$ in the dashed box in the middle of the figure is the Embedding vector what we need. The model uses the first-order loss function to make the embedding vectors corresponding to adjacent vertices

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

close, thereby preserving the local structural characteristics of the vertices.

a) 2nd order similarity optimization goal

After getting $y_i^{(K)}$, we can get the output $\hat{x}_i$ by reversing the calculation process of encoder. The goal of the autoencoder is to decrease the reconstruction error of the output and the input. The loss function for this goal is shown as:

$$L_{2nd} = \sum_{i=1}^{n} \| \hat{x}_i - x_i \|_2^2 \tag{3}$$

we use the adjacency matrix S as the input to the autoencoder. For the i^{th} vertex, $x_i=s_i$, and each s_i contains the neighbor structure information of vertex i. Thus this reconstruction process can make vertices with similar structures have similar embedding vectors. However, the non-zero elements in the adjacency matrix S are far less than zero elements because of the sparseness of the graph. So for the neural network, only that all output is 0 can obtain a good result. But we don't want this effect like this. We suppose to use a weighted loss function which has a higher penalty coefficient for non-zero elements. The revised objective function is shown as follows:

$$L_{2nd} = \sum_{i=1}^{n} \| (\hat{x}_i - x_i) \odot b_i \|_2^2 \tag{4}$$

$$= \| (\hat{x}_i - x_i) \odot B \|_F^2 \tag{5}$$

The element-wise product is $b_i = \{b_{i,j}\}_{j=1}^{n}$, If si,j=0, bi,j=1, else $b_{i,j} = \beta > 1$. Now by using the revised deep autoencoder with the adjacency matrix S as input, the vertexes which have similar neighborhood structure will be mapped near in the representations space. In the other words, the unsupervised component of our model can preserve the global network structure by reconstructing the second- order proximity between vertexes.

b) 1st order similarity optimization goal

In order to constrain the similarity of the latent representations of a pair of vertexes. we design the supervised component to exploit the first-order proximity. The loss function for this goal is defined as follows:

$$L_{1st} = \sum_{i,j=1}^{n} S_{i,j} \| y_i^{(K)} - y_j^{(K)} \|_2^2 \tag{6}$$

$$= \sum_{i,j=1}^{n} S_{i,j} \| y_i - y_j \|_2^2 \tag{7}$$

The loss function can make the embedding vectors corresponding to two adjacent vertices in the graph close in the hidden space. What's more, L_{1st} can also be defined as:

$$L_{1st} = \sum_{i,j=1}^{n} \| y_i - y_j \|_2^2 = 2tr(Y^T L Y) \tag{8}$$

c) Overall optimization goal

To preserve the first-order and second-order proximity simultaneously, we use a semi-supervised model, which combines Eq. 5 and Eq. 6 and joint minimizes the following objective function:

$$L_{mix} = L_{2nd} + \alpha L_{1st} + v L_{reg} \tag{9}$$

L_{reg} is a regularization term, a parameter that controls the first-order loss, and a parameter that controls the regularization term.

4. Experiment and Discussion

4.1 Dataset

We use wikidata to train and verify the role of deepwalk, node2vec, and SDNE in node classification. Wikidata extracts the common cognition entries in Wikipedia in different languages, including the Chinese version, and extracts the structured data of the page to build a large knowledge database that can be read, written and edited by both machines and humans, and other data Compared with the source, Wikidata is open, collaborative, multilingual, and structured. [13] To ensure the accuracy of the experiment, we use category, edgelist, and labels under wikidata.

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

4.2 Implementation Datail

The three algorithm codes used in the experiment are as follows:

```
def evaluate_embeddings(embeddings):
    X, Y = read_node_label('../data/wiki/wiki_labels.txt')
    tr_frac = 0.8
    print("Training classifier using {:.2f}% nodes...".format(
        tr_frac * 100))
    clf = Classifier(embeddings=embeddings, clf=LogisticRegression())
    clf.split_train_evaluate(X, Y, tr_frac)

def plot_embeddings(embeddings,):
    X, Y = read_node_label('../data/wiki/wiki_labels.txt')

    emb_list = []
    for k in X:
        emb_list.append(embeddings[k])
    emb_list = np.array(emb_list)

    model = TSNE(n_components=2)
    node_pos = model.fit_transform(emb_list)

    color_idx = {}
    for i in range(len(X)):
        color_idx.setdefault(Y[i][0], [])
        color_idx[Y[i][0]].append(i)

    for c, idx in color_idx.items():
        plt.scatter(node_pos[idx, 0], node_pos[idx, 1], label=c)
    plt.legend()
    plt.show()

if __name__ == "__main__":
    G = nx.read_edgelist('../data/wiki/Wiki_edgelist.txt',
                         create_using=nx.DiGraph(), nodetype=None, data=[('weight', int)])

    model = DeepWalk(G, walk_length=10, num_walks=80, workers=1)
    model.train(window_size=5, iter=3)
    embeddings = model.get_embeddings()

    evaluate_embeddings(embeddings)
    plot_embeddings(embeddings)
```

Figure 5 The Deepwalk Algorithm

```
 def evaluate_embeddings(embeddings):
     X, Y = read_node_label('../data/wiki/wiki_labels.txt')
     tr_frac = 0.8
     print("Training classifier using {:.2f}% nodes...".format(
         tr_frac * 100))
     clf = Classifier(embeddings=embeddings, clf=LogisticRegression())
     clf.split_train_evaluate(X, Y, tr_frac)

 def plot_embeddings(embeddings,):
     X, Y = read_node_label('../data/wiki/wiki_labels.txt')

     emb_list = []
     for k in X:
         emb_list.append(embeddings[k])
     emb_list = np.array(emb_list)

     model = TSNE(n_components=2)
     node_pos = model.fit_transform(emb_list)

     color_idx = {}
     for i in range(len(X)):
         color_idx.setdefault(Y[i][0], [])
         color_idx[Y[i][0]].append(i)

     for c, idx in color_idx.items():
         plt.scatter(node_pos[idx, 0], node_pos[idx, 1], label=c)
     plt.legend()
     plt.show()

 if __name__ == "__main__":
     G=nx.read_edgelist('../data/wiki/Wiki_edgelist.txt',
                        create_using = nx.DiGraph(), nodetype = None, data = [('weight', int)])
     model = Node2Vec(G, walk_length=10, num_walks=80,
                      p=0.25, q=4, workers=1, use_rejection_sampling=0)
     model.train(window_size = 5, iter = 3)
     embeddings=model.get_embeddings()

     evaluate_embeddings(embeddings)
     plot_embeddings(embeddings)
```

Figure 6 The Node2vec Algorithm

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

```
def evaluate_embeddings(embeddings):
    X, Y = read_node_label('../data/wiki/wiki_labels.txt')
    tr_frac = 0.8
    print("Training classifier using {:.2f}% nodes...".format(
        tr_frac * 100))
    clf = Classifier(embeddings=embeddings, clf=LogisticRegression())
    clf.split_train_evaluate(X, Y, tr_frac)

def plot_embeddings(embeddings,):
    X, Y = read_node_label('../data/wiki/wiki_labels.txt')

    emb_list = []
    for k in X:
        emb_list.append(embeddings[k])
    emb_list = np.array(emb_list)

    model = TSNE(n_components=2)
    node_pos = model.fit_transform(emb_list)

    color_idx = {}
    for i in range(len(X)):
        color_idx.setdefault(Y[i][0], [])
        color_idx[Y[i][0]].append(i)

    for c, idx in color_idx.items():
        plt.scatter(node_pos[idx, 0], node_pos[idx, 1],
                    label=c)  # c=node_colors)
    plt.legend()
    plt.show()

if __name__ == "__main__":
    G = nx.read_edgelist('../data/wiki/Wiki_edgelist.txt',
                         create_using=nx.DiGraph(), nodetype=None, data=[('weight', int)])

    model = SDNE(G, hidden_size=[256, 128],)
    model.train(batch_size=3000, epochs=40, verbose=2)
    embeddings = model.get_embeddings()

    evaluate_embeddings(embeddings)
    plot_embeddings(embeddings)
```

Figure 7 The SDNE Algorithm

By running the specific codes of the three algorithms to train the wiki dataset, we got the following results:

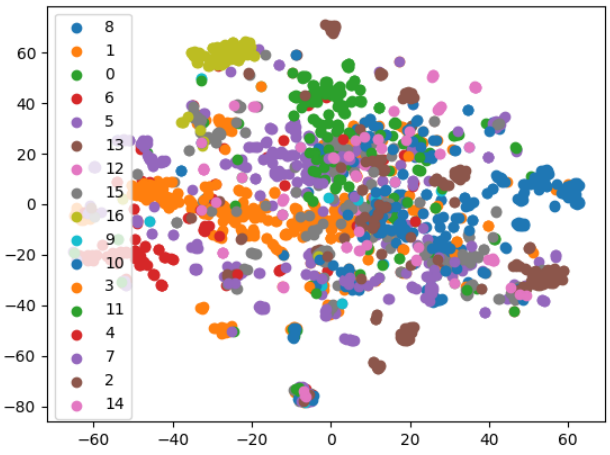

Figure 8 Deepwalk on Node Classification

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

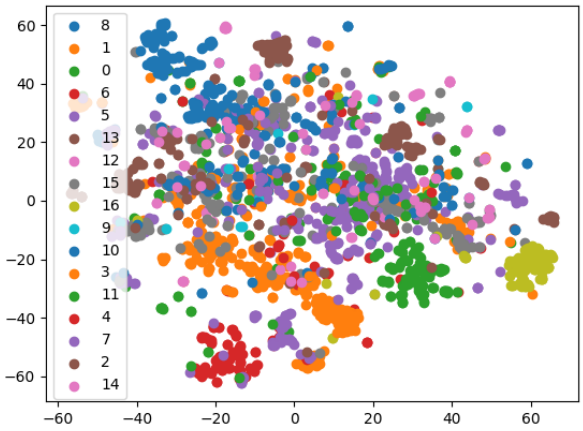

Figure 9 Node2vec on Node classification

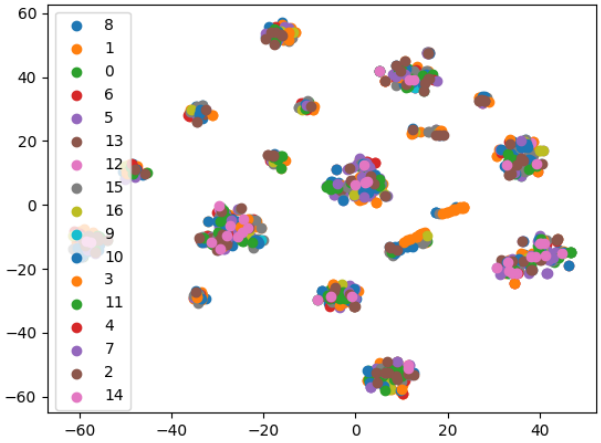

Figure 10 SDNE on Node classification

4.3 Discussion

Through the comparison of the above three figures, it can be seen that the node classification result of the SDNE algorithm is the best.

The principle of the DeepWalk algorithm is simple. It can get better results when the network has a few vertices, and has good scalability and can adapt to changes in the network. However, because the walking strategy adopted by DeepWalk is too simple (BFS), it cannot effectively characterize the structural information of the nodes of the graph.

Node2vec was born to overcome the relatively simple problem of the random walk strategy in deepwalk. Compared with DeepWalk, Node2vec can balance the structural similarity and homogeneity of nodes in the graph by designing a biased-random walk strategy, making the model more flexible. The homogeneity and structure of the network embodied by Node2Vec are both very important feature expressions in the recommendation system. But like DeepWalk, Node2vec cannot specify a walking path, and is only suitable for solving a homogeneous network containing only one type of node, and cannot effectively represent a complex network containing multiple types of nodes and edge types.

SDNE (Structural Deep Network Embedding) is a parallel work with Node2vec, but compared to Node2vec and deepwalk, SDNE is different in that it is not based on the idea of random walk and is relatively stable in practice. SDNE wants to preserve the first-order and second-order similarity, and also wants to optimize at the same time to capture the local pairwise similarity and the similarity of the node neighborhood structure.

ISSN 2616-5767 Vol.4, Issue 2: 89-97, DOI: 10.25236/AJETS.2021.040215

5. Conclusion

In this article, we mainly discuss the role of three graph embedding algorithms: deepwalk, node2vec and SDNE when classifying WIKI data. Due to the structured characteristics of Wiki data, in the analysis and research, we compared the accuracy and efficiency of three graph embedding algorithms in node classification. In deepwalk, Node2vec and SDNE, because the SDNE algorithm will first-order and second -order is jointly applied to the deep learning process. The former is used to capture the local structure, and the latter is used to capture the global structure. Therefore, the SDNE algorithm has a greater role in node classification of wiki data. At present, there are few types of research on the application of node classification and visualization of wiki data sets at home and abroad, and node classification research of wiki data sets can play a huge role in the fields of recommendation, search, etc. Therefore, in-depth research on wiki data sets It is important and meaningful work.

Acknowledgment

Supported by "College Students Innovation and Entrepreneurship Training Program" (Grant No-20211647017)

References

[1] Kou Leilei. Analysis of Data Sources in Wikidata. Library Theory and Practice. 2020:73-77
[2] Zhai Zhengli, Feng Shu, Li Penghui. Improved collaborative filtering recommendation model of graph convolutional network [J/OL]. Computer Engineering and Application: 1-10[2021-05-06]. http://kns.cnki.net/kcms/detail/11.2127.TP.20210419.1334.029.html.
[3] Liu Feng; Wang Baoliang; Zou Rongyu; Zhao Haochun. Research on the recommendation algorithm of network representation learning based on random walk. Computer Engineering: 1-9
[4] Grover Aditya;Leskovec Jure.node2vec: Scalable Feature Learning for Networks.[J].KDD: proceedings. International Conference on Knowledge Discovery & Data Mining, 2016, 2016:855-864.
[5] Ma Yang; Cheng Guangquan; Liang Xingxing; Li Yan; Yang Yuling; Liu Zhong. Improved SDNE algorithm in directed weighted network. Computer Science. 2020:239-243
[6] Xue Qiuhong, Jia Junzhi, Liu Huizhou. Implementation of semantic association between Chinese name specification data and Wikidata[J]. Information Theory and Practice, 2019, 42(10) :146-150.
[7] Zeng Ruming; Li Yunfei. Research on an improved method of K-means clustering algorithm. Journal of Shaoyang University (Natural Science Edition). 2021:14-20
[8] Xie Juanying, Qu Yanan. K-medoids clustering algorithm for density peak optimization initial center [J]. Computer Science and Exploration, 2016, 10(2) :230-247
[9] Han Linyi; Wu Sheng. Collaborative filtering algorithm based on K nearest neighbors optimized by scoring features. Information technology. 2018:83-87
[10] Bryan Perozzi,Rami Al-Rfou,Steven Skiena. DeepWalk[P]. Knowledge discovery and data mining, 2014.
[11] Grover Aditya,Leskovec Jure. node2vec: Scalable Feature Learning for Networks. [J]. KDD: proceedings. International Conference on Knowledge Discovery & Data Mining, 2016.
[12] Daixin Wang,Peng Cui,Wenwu Zhu. Structural Deep Network Embedding [P]. Knowledge Discovery and Data Mining, 2016.
[13] Wikidata main page[EB/OL].[2016-02-07].https://www.wikidata.org.