Graph Embedding for link prediction

一、簡介

這次作業我們使用embedding methods去對Facebook undirected network做link prediction，使用的方法有DeepWalk(DeepWalk: Online Learning of Social Representations)、Node2Vec(node2vec: Scalable Feature Learning for Networks)、LINE(LINE: Large-scale Information Network Embedding)，除了用embedding方法訓練外，最後也會跟Baseline Indexes(jaccard coefficient、preferential attachment、common neighbors)方法的預測結果做比較。

二.程式架構及演算法流程

<u>Data preprocessing</u>

1.先隨機remove掉50%的positive edges，再隨機取相同數量的negative edges，positive label為1，negative label為0

2.取得每個edge的features(不同feature值有各自演算法)

3.用train_test_split函式分割train data與test data(testsize=0.5)

<u>Algorithm與程式處理</u>

1.jaccard coefficient、preferential attachment、common neighbors使用networkx內建涵式(演算法即是上課影片的公式)

2.Node2Vec

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
    π = PreprocessModifiedWeights(G, p, q)
    $G' = (V, E, \pi)$
    Initialize $walks$ to Empty
    for $iter = 1$ **to** r **do**
        for all nodes $u \in V$ **do**
            $walk$ = node2vecWalk(G', u, l)
            Append $walk$ to $walks$
    f = StochasticGradientDescent($k, d, walks$)
    return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
    Inititalize $walk$ to $[u]$
    for $walk\_iter = 1$ **to** l **do**
        $curr = walk[-1]$
        V_{curr} = GetNeighbors($curr, G'$)
        s = AliasSample(V_{curr}, π)
        Append s to $walk$
    return $walk$

總共進行r次random walk，對於walk的每一步都根據轉移概率πvx進行採樣，πvx可預先計算，所以效率為O(1)。node2vec主要有3階段:

(1)先計算PI。dtx代表t與x間的最短路徑，dtx只能是0,1,2。p控制返回上個node的可能性，q控制往裡走或往外走。如果p,q都是1，游走方式等同於DeepWalk中的隨機游走。

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

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

(2)開始r次的random walk，每次walk都要以每個node為源node進行node2vecWalk，存入array中，後面SGD會用到

(3)對於每個node2vecWalk，從源節點u找1個節點，初始時walk={u}，curr代表前一個node，Vcurr代表當前node，AliasSample取得下一個node s，將s加進walk array中，最後返回walk加進walks array中進行SGD優化。

拓展到link prediction:

給定node u,v，定義二元算子使g(u,v)=f(u)。f(v)

g:V x V → Rd’(此即為特徵向量，trainng時用得到) d’是維度

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\Vert \cdot \Vert_{\bar{1}}$	$\Vert f(u) \cdot f(v)\Vert_{\bar{1}i} = \vert f_i(u) - f_i(v)\vert$
Weighted-L2	$\Vert \cdot \Vert_{\bar{2}}$	$\Vert f(u) \cdot f(v)\Vert_{\bar{2}i} = \vert f_i(u) - f_i(v)\vert^2$

Table 1: Choice of binary operators $\circ$ for learning edge features. The definitions correspond to the ith component of $g(u, v)$.

我們這裡選Hadamard算子

程式:(使用python node2vec函式庫)

```
from node2vec import Node2Vec
from node2vec.edges import HadamardEmbedde
```

參數d = 128, r = 10, l = 80 p,q=0.25，選用Hadamard算子算出特徵向量回傳edges_embs為dict type　　ex: edges_embs[('1','2')]可以查edge(1,2)的特徵向量

```
def n2v_embedding(train_G):
    node2vec  = Node2Vec(train_G, dimensions=128, walk_length=80, num_walks=10, workers=4,p=0.25,q=0.25)
    model = node2vec .fit(window=10, min_count=1, batch_words=4)
    edges_embs = HadamardEmbedder(keyed_vectors=model.wv)
    return edges_embs
```

```
edges_embs = n2v_embedding(train_G)
```

```
Computing transition probabilities: 100%|██████████
0:00, 1441.17it/s]
```

將每個edge特徵向量併入表格，一維度為一column

```
def n2v_combine_embedding(data):
    i=0
    X = []
    for node1,node2 in data:
        X.append(np.concatenate((data[i],embeddings[(str(int(node1)), str(int(node2)))])))
  # print(embeddings[str(int(data[0]))])
        i+=1
    return X
```

3.DeepWalk

使用github函式庫(https://github.com/shenweichen/GraphEmbedding)

```
from ge import DeepWalk
```

```
from ge import LINE
```

```
model = DeepWalk(train_G, walk_length=10, num_walks=80, workers=4)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   2 out of   4 | elapsed:   19.9s remaining:   19.9s
[Parallel(n_jobs=4)]: Done   4 out of   4 | elapsed:   22.4s finished
```

```
model.train(window_size=5, iter=3)
```

```
Learning embedding vectors...
Learning embedding vectors done!
```

```
DeepWalk_embeddings = model.get_embeddings()
```

將每個edge特徵向量併入表格(LINE也可使用此函式)

```python
def combine_embedding(data,embeddings):
    i=0
    X = []
    for node1,node2 in data:
        X.append(np.concatenate((data[i],embeddings[int(node1)],embeddings[int(node2)])))
   # print(embeddings[str(int(data[0]))])
        i+=1
    return X
```

4.LINE

This approach learns d-dimensional feature representations in two separate phases. In the first phase, it learns d/2 dimensions by BFS-style simulations over immediate neighbors of nodes. In the second phase, it learns the next d/2 dimensions by sampling nodes strictly at a 2-hop distance from the source nodes

參數order='all'代表前128維度是1st-order，後128維度是2st-order

```python
model = LINE(train_G, embedding_size=128, order='all')
```

```python
model.train(batch_size=10240, epochs=50, verbose=2)
```

```
Epoch 1/50
51/51 - 5s - loss: 1.3863 - first_order_loss: 0.6931 - second_order_loss: 0.6931
Epoch 2/50
51/51 - 5s - loss: 1.3851 - first_order_loss: 0.6927 - second_order_loss: 0.6924
Epoch 3/50
51/51 - 5s - loss: 1.3814 - first_order_loss: 0.6920 - second_order_loss: 0.6894
Epoch 4/50
51/51 - 5s - loss: 1.3610 - first_order_loss: 0.6914 - second_order_loss: 0.6696
Epoch 5/50
51/51 - 6s - loss: 1.2899 - first_order_loss: 0.6899 - second_order_loss: 0.6000
Epoch 6/50
51/51 - 6s - loss: 1.1730 - first_order_loss: 0.6886 - second_order_loss: 0.4844
Epoch 7/50
51/51 - 6s - loss: 1.1010 - first_order_loss: 0.6852 - second_order_loss: 0.4158
Epoch 8/50
51/51 - 6s - loss: 1.0391 - first_order_loss: 0.6826 - second_order_loss: 0.3565
```

```python
LINE_embeddings = model.get_embeddings()
```

Training and Prediction

使用RandomForest隨機森林分類器訓練model，再進行預測

```python
clf1 = RandomForestClassifier(n_estimators=300)
clf1.fit(X_train, y_train)
```

```python
predict_Y = clf1.predict(X_test)
print(get_accuracy(predict_Y, y_test))
```

X_test是一開始remove 50%的edges與同數量negative edges合併而成

X_train是另 50%的edges與同數量negative edges合併而成

三、實驗結果

下面表格顯示出明顯Node2Vec在選用Hadamard為算子的情況下預測效果最好。

dataset:https://snap.stanford.edu/data/facebook-large-page-page-network.html

Algorithm	auc
Jaccard Coefficient	0.844821
Pref. Attachment	0.804172
Common Neighbors	0.844370
Node2vec	0.911551
LINE	0.841265
DeepWalk	0.906504
jc,pa,cn	0.856645

參考資料:
https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf
https://github.com/shenweichen/GraphEmbedding
http://pythonsparkhadoop.blogspot.com/2016/12/spark-randomforest_83.html
https://github.com/eliorc/node2vec
https://github.com/aditya-grover/node2vec
https://github.com/eliorc/node2vec
https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/557816/