Homeworks_Questions

December 20, 2021

1 Homeworks

1. Find the answer to the question raised in the lab1

   Some helpful resources:

- DeepWalk: https://arxiv.org/pdf/1403.6652.pdf
- Word2vec: https://arxiv.org/pdf/1301.3781.pdf
- Repository Github of Word2vec at this link

2. Implement a simple word2vec algorithm for the DeepWalk (Attributes for each node should be created).
3. Use some libraries to solve a real problem

2 Answer

2.1 Implement Word2vec

2.1.1 Download data and install packages

[ ]:

```
!pip install karateclub==1.2.0
!pip install umap-learn
```

[ ]:

```
!pip uninstall numpy -y
!pip install numpy
```

[ ]:

```
!gdown --id "1RmrHId0d-uY7kJCSgCtNYbwYfp40um3c&export=download"
!unrar x -Y "/content/lab3.rar" -d "/content/"
```

2.1.2 Packages

[ ]:

```
# Task 1
import networkx as nx
from joblib import Parallel, delayed
import random
import itertools
import numpy as np
import pandas as pd

# Task 2
```

```
import json
import umap
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, confusion_matrix
from karateclub.utils.walker import RandomWalker
from gensim.models.word2vec import Word2Vec
import seaborn as sns
```

2.1.3 Utils

[ ]:

```
def partition_num(num, workers):
    if num % workers == 0:
        return [num//workers]*workers
    else:
        return [num//workers]*workers + [num % workers]

def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()



def get_attributes_of_node(node_paths):
  node_paths_attributes = []
  # Get attribute (word) for each node
  df_attr = pd.read_csv("lab3_attributes.csv").astype(str)
  dict_attr = {}
  for i in range(len(df_attr)):
    dict_attr[df_attr.iloc[i, 0]] = df_attr.iloc[i, 1]
  for path in node_paths:
    for index, node in enumerate(path):
      path[index] = dict_attr[node]
    node_paths_attributes.append(path)
  return node_paths_attributes

def preprocessing(sentences):
    training_data = []
    for sentence in sentences:
        x = [word for word in sentence]
        training_data.append(x)
    return training_data



def prepare_data_for_training(sentences,w2v):
    data = {}
    for sentence in sentences:
        for word in sentence:
            if word not in data:
                data[word] = 1
            else:
                data[word] += 1
    V = len(data)
    data = sorted(list(data.keys()))
    vocab = {}
    for i in range(len(data)):
        vocab[data[i]] = i

    for sentence in sentences:
        for i in range(len(sentence)):
            center_word = [0 for x in range(V)]
            center_word[vocab[sentence[i]]] = 1
            context = [0 for x in range(V)]

            for j in range(i-w2v.window_size,i+w2v.window_size):
                if i!=j and j>=0 and j<len(sentence):
                    context[vocab[sentence[j]]] += 1
            w2v.X_train.append(center_word)
            w2v.y_train.append(context)
    w2v.initialize(V,data)

    return w2v.X_train,w2v.y_train
```

2.1.4 TO DO

[ ]:

```
class word2vec():
    pass
    # TO DO
```

2.1.5 DeepWalk

[ ]:

```
class RandomWalker:
  def __init__(self, G, num_walks, walk_length):
      """
      :param G: Graph
      :param num_walks: a number of walks
      :param walk_length: Length of a walk. Each walk is considered as a␣
 ↪sentence
      """
      self.G = G
      self.num_walks = num_walks
      self.walk_length = walk_length
  def deepwalk_walk(self, start_node):
      """
      :param start_node: Starting node of a walk
      """
      walk = [start_node]
      while len(walk) < self.walk_length:
          cur = walk[-1]
          # Check if having any neighbors at the current node
          cur_nbrs = list(self.G.neighbors(cur))
          if len(cur_nbrs) > 0:
              # Random walk with the probability of 1/d(v^t). d(v^t) is the␣
↪node degree
              walk.append(random.choice(cur_nbrs))
          else:
              break
      return walk


  def simulate_walks(self, workers=1, verbose=0):
      """
      :param workers: a number of workers running in parallel processing
      :param verbose: progress bar
      """
      G = self.G
      nodes = list(G.nodes())
      results = Parallel(n_jobs=workers, verbose=verbose)(
          delayed(self._simulate_walks)(nodes) for num in
          partition_num(self.num_walks, workers))
      walks = list(itertools.chain(*results))
      return walks


  # INFORMATION EXTRACTOR
  def _simulate_walks(self, nodes):
      walks = []
      # Iterate all walks per vertex
      for _ in range(self.num_walks):
          random.shuffle(nodes)
          # Iterate all nodes in a walk
          for v in nodes:
            walks.append(self.deepwalk_walk(start_node=v))
      return walks
```

[ ]:

```
class DeepWalk:
    def __init__(self, graph, walk_length, num_walks, workers=1):
        self.graph = graph
        self.w2v_model = None
        self._embeddings = {}

        self.walker = RandomWalker(graph, num_walks=num_walks,␣
↪walk_length=walk_length)
        self.walks = self.walker.simulate_walks(workers=workers, verbose=1)
        self.sentences = get_attributes_of_node(self.walks)


    def train(self, window_size=5, epochs=100):
        print("Learning embedding vectors...")
        training_data = preprocessing(self.sentences)
        w2v = word2vec(window_size, epochs)
        prepare_data_for_training(training_data, w2v)
        w2v.train()
        print("Learning embedding vectors done!")
        self.w2v_model = w2v


    def test(self, word):
        print(self.w2v_model.predict(word,3))
```

2.1.6 Run graph embedding

[ ]:

```
G = nx.read_edgelist('lab3_edgelist.txt',create_using=nx.
↪DiGraph(),nodetype=None,data=[('weight',int)])# Read graph
model = DeepWalk(G, walk_length=3, num_walks=10, workers=1)#init model
model.train(window_size=5)# train model
```

[ ]:

```
print(model.sentences)
model.test("to")
model.test("this")
```

2.2 TO DO: Solve a real problem using some libraries

Goal: When we have a large graph dataset like the Facebook dataset below, we want to classify which company (node) will likely belong to a type of page. If we categorize well, we could apply marketing strategies in a domain on a company that we are surveying. Therefore, our task is to learn a model which can classify a company using related features.

1. Analyze and visualize the dataset Facebook downloaded in this website.
2. Use DeepWalk to embed the graph
3. Train a classifier to do the node classification task using the embedding graph from step 2.

You can do many things with the data. I recommend that you could try many tasks with this data, not only the classification task.

2.2.1 Read data

[ ]:

```
edges_path = 'facebook_edges.csv'
targets_path = 'facebook_target.csv'
features_path = 'facebook_features.json'
```

2.2.2 Visualize datasets

Create a graph. If you want to use smaller graph, please try to create one. It will be lighter when running the code.

2.2.3 Embedding graph using DeepWalk

Embedding graph using DeepWalk

2.2.4 Train a classifier

Train a classifer from the embedding graph to the target. Here we use the Random Forest classifier.

3 THANK YOU

Please dive more into the codes and papers if you are interested.

Thank you for joining all the labs.