EECS 4415 - Part 6

Exploring Big Data with Embedding Techniques

Setup

First of all, we install the graph2vec library which offers a fast implementation of the node2vec method.

To learn how to implement fast random walks on graphs, read the blog post which explains some of the design choices behind this library.

```
!python -m pip install 'csrgraph @ git+https://github.com/VHRanger/CSRGraph@f052c1cf128ab21d21a4710337dc7c1cd5658df7'
!python -m pip install 'nodevectors @ git+https://github.com/aman0456/nodevectors@67e9af0506236be9aae6d460e96d4dec6329bcd2'
```

```
Collecting csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128ab21d21a4710337dc7c1cd5658df7
  Using cached csrgraph-0.1.29-py3-none-any.whl
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf1
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128a
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128a
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128a
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from csrgraph@ git+https://github.com/VHRanger/CSRGraph@f052c1cf128ab
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->csrgraph@ git+https://github.com/VH
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->csrgraph@ git+https://github.com/VHRan
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->csrgraph@ git+https://github.com/VHRanger/CSRGra
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->csrgraph@ git+https://github.com/VHRanger/CSRG
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->csrgraph@ git+https://github.com/VHRanger
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->csrgraph@ git+https://github.com/V
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->csrgraph@ git+https://github
Collecting nodevectors@ git+https://github.com/aman0456/nodevectors@67e9af0506236be9aae6d460e96d4dec6329bcd2
  Using cached nodevectors-0.1.23-py3-none-any.whl
Requirement already satisfied: csrgraph in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e9a
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e9af
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e9af
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors@67e9af
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from nodevectors@ git+https://github.com/aman0456/nodevectors
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->nodevectors@ git+https://github.c
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->nodevectors@ git+https://github.com/aman045
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->nodevectors@ git+https://github.com/aman0
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from csrgraph->nodevectors@ git+https://github.com/aman0456/nodevecto
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim->nodevectors@ git+https://github.com/aman045
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->nodevectors@ git+https://github.com
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->nodevectors@ git+https://github.com/aman0
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->nodevectors@ git+https://github.co
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0->nodevectors@ git+https:
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from smart-open>=1.8.1->gensim->nodevectors@ git+https://github.com/
```

We now import the library, and create a small wrapper class which will expose only the few hyperparameters we will need to tune in this Colab

```
from nodevectors import Node2Vec
import networkx as nx

class Node2VecNew(Node2Vec):
  """
  Parameters
  ----------
  p : float
      p parameter of node2vec
  q : float
      q parameter of node2vec
  d : int
      dimensionality of the embedding vectors
  """
  def __init__(self, p=1, q=1, d=32):
    super().__init__(
                 n_components=d,
                     walklen=10,
```

```
                       epochs=50,
                       return_weight=1.0/p,
                       neighbor_weight=1.0/q,
                       threads=0,
                       w2vparams={'window': 4,
                                  'negative': 5,
                                  'iter': 10,
                                  'ns_exponent': 0.5,
                                  'batch_words': 128})
```

Lastly, let's import some of the common libraries needed for our task.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline
```

Function to print graph information

```
def print_graph_info(G, directed=True):
  print("Number of nodes:", len(G.nodes))
  print("Number of edges:", len(G.edges))
  if directed:
    print("Average in-degree:", sum(dict(G.in_degree).values()) / len(G.nodes))
    print("Average out-degree:", sum(dict(G.out_degree).values()) / len(G.nodes))
  else:
    print("Average degree:", sum(dict(G.degree).values()) / len(G.nodes))
```

Example

In the example below, we will try to reproduce the plot in the lecture notes.

```
# Load the Zachary's Karate Club as a NetworkX Graph object
KCG = nx.karate_club_graph()

# Fit embedding model to the Karate Club graph
n2v = Node2VecNew(1, 1, 2)
n2v.fit(KCG)
```

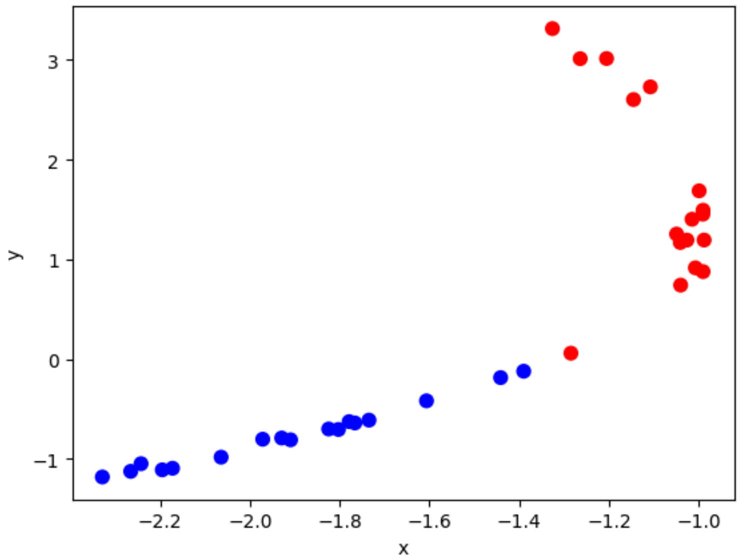
```
Making walks... Done, T=12.48
Mapping Walk Names... Done, T=0.04
Training W2V... WARNING: gensim word2vec version is unoptimizedTry version 3.6 if on windows, versions 3.7 and 3.8 have had issues
Done, T=0.37
```

```
embeddings = []
for node in KCG.nodes:
  embedding = list(n2v.predict(node))
  club = KCG.nodes[node]['club']
  embeddings.append(embedding + [club])

# Construct a pandas dataframe with the 2D embeddings from node2vec,
# plus the club name that each node belongs to after the split
df = pd.DataFrame(embeddings, columns=['x', 'y', 'club'])
```

```
# Nodes who stayed with the Mr. Hi will be plotted in red, while nodes
# who moved with the Officer will be plotted in blue
colors = ['red' if x == 'Mr. Hi' else 'blue' for x in df.club]
df.plot.scatter(x='x', y='y', s=50, c=colors)
```

```
<Axes: xlabel='x', ylabel='y'>
```



If our example trained correctly, you should notice a clear separation between the blue and red nodes. Solely from the graph structure, node2vec could predict how the Zachary's Karate Club split!

Tune the hyperparameters `p` and `q`, and notice how they affect the resulting embeddings.

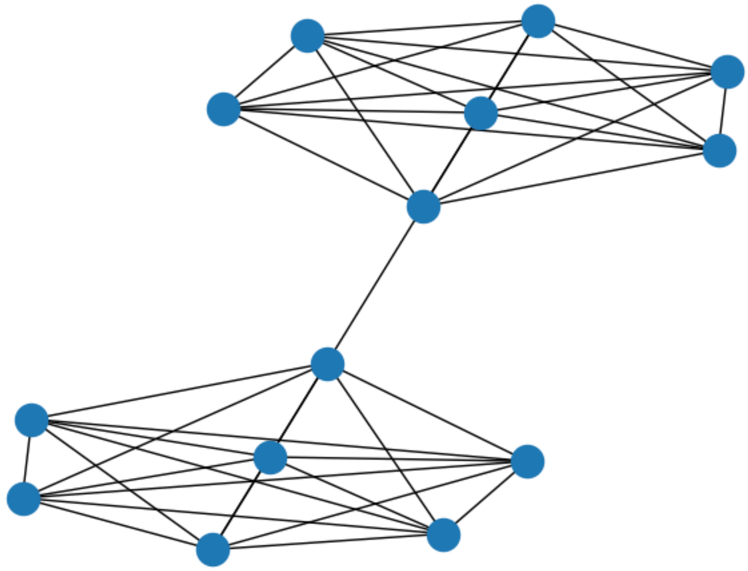
1. p > 1: as p grows larger, embeddings show tendancy of clear separation in most of the time. However, repeated execution of fitting embedding model sometimes yields one or two embeddings of each color not so clearly separated.
2. q > 1: as q grows larger, seperation of blue and red nodes becomes severely compromised by around 50% chance. When this happens, the graph shows blue dots are enclosed by red ones as if we can have 3 seperation instead of 2.
3. p < 1: as p approaches to 0, blue and red nodes distributed almost randomly and become inseperable in some cases.
4. q < 1: repeated execution of fitting embedding mode shows all the patterns mentioned above sometimes.

Your Task

Now we will study the behavior of node2vec on [barbell graphs].

Below you can see a toy example of a barbell graph generated with NetworkX.

```
toy_barbell = nx.barbell_graph(7, 0)
nx.draw_kamada_kawai(toy_barbell)
```



Generate a larger barbell graph, where each complete graph has exactly 1000 nodes. Print graph information.

Then, learn node2vec embeddings on this graph, setting `p = 1, q = 1` and `d = 10`.

```
''' 5-7 lines of code in total expected but can differ based on your style.'''

# YOUR CODE HERE
barbell = nx.barbell_graph(1000, 0)
print_graph_info(barbell, False)
barbell_n2v = Node2VecNew(1, 1, 10)
barbell_n2v.fit(barbell)
barbell_embeddings = []
for node in barbell.nodes:
  barbell_embeddings.append(list(barbell_n2v.predict(node)))
```

```
Number of nodes: 2000
Number of edges: 999001
Average degree: 999.001
Making walks... Done, T=1.62
Mapping Walk Names... Done, T=0.50
Training W2V... WARNING: gensim word2vec version is unoptimizedTry version 3.6 if on windows, versions 3.7 and 3.8 have had issues
Done, T=24.76
```

Write a function that takes as input a node id `n` in the graph and returns a list containining the cosine similarity between the node2vec vector of the input node `n` and all the nodes in the given barbell graph (including the similarity with `n` itself).

```
''' 12-20 lines of code in total expected but can differ based on your style.'''

# YOUR CODE HERE
from sklearn.metrics.pairwise import cosine_similarity

def cos_sim(n, v):
  df = pd.DataFrame(barbell_embeddings, columns=range(len(barbell_embeddings[0])), index=range(len(barbell_embeddings)))
  cosine = cosine_similarity(df.loc[n].values.reshape(1, -1), df.values)
  return pd.DataFrame(cosine, columns=df.index, index=[n])
```

Print the result of the function for n=5.

```
''' 1 line of code in total expected.'''

#YOUR CODE HERE
cos_sim(5, barbell_embeddings)
```

	0	1	2	3	4	5	6	7	8	9	...	1990	1991	1992	1993	19
5	0.922979	0.936762	0.955705	0.954775	0.939406	1.0	0.912121	0.938216	0.927516	0.976644	...	0.145497	0.102861	0.13255	0.150351	0.12

1 rows × 2000 columns

Write the function to compute the number of nodes in the graph whose embeddings have exactly 1000 neighbors with a cosine similarity greater than 0.8. Print the result.

```
''' 12-15 lines of code in total expected but can differ based on your style.'''

#YOUR CODE HERE
def node_filter(embeddings, neighbors, cos_threshold):
  df = pd.DataFrame(embeddings, columns=range(len(embeddings[0])), index=range(len(embeddings)))
  cosine = cosine_similarity(df, df)
  cosine_filtered = []
  for row in cosine:
    count = 0
    for value in row:
      if eval("%s%s" % (value, cos_threshold)):
        count += 1
    #excluding the similarity from one node to itself by adding 1 to count
    if eval("%s%s" % (count+1, neighbors)):
      cosine_filtered.append(row)
  return len(cosine_filtered)

print("Number of nodes in the graph satisfies the given condition: ",node_filter(barbell_embeddings, '==1000', '>0.8'))
```

```
Number of nodes in the graph satisfies the given condition:  246
```

Generate another barbell graph, this time adding a path of length 51 between

---

the two complete graphs and print graph information. To find out how, refer to the NetworkX documentation: [https://networkx.github.io/documentation/stable/reference/

---

generated/networkx.generators.classic.barbell_graph.html#networkx.generators.classic.barbell_graph] (https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.classic.barbell_graph.html#networkx.generators.classic.barbell_graph)

Learn the node2vec embeddings for the nodes of this new graph, using the same hyperparameters as before.

```
''' 4-6 lines of code in total expected but can differ based on your style.'''

# YOUR CODE HERE
another_barbell = nx.barbell_graph(1000, 51)
print_graph_info(another_barbell, False)
barbell_n2v = Node2VecNew(1, 1, 10)
barbell_n2v.fit(another_barbell)
another_embeddings = []
for node in another_barbell.nodes:
  another_embeddings.append(list(barbell_n2v.predict(node)))
```

```
Number of nodes: 2051
Number of edges: 999052
Average degree: 974.2096538274012
Making walks... Done, T=1.00
Mapping Walk Names... Done, T=0.51
Training W2V... WARNING: gensim word2vec version is unoptimizedTry version 3.6 if on windows, versions 3.7 and 3.8 have had issues
Done, T=24.67
```

Print the number of nodes in the graph whose embeddings have exactly 1000 neighbors with a cosine similarity greater than 0.8.

```
''' 1 line of code in total expected.'''

# YOUR CODE HERE
print("Number of nodes satisfies the given condition: " ,node_filter(another_embeddings, '==1000', '>0.8'))
```

```
Number of nodes satisfies the given condition:  1253
```

Write a function to compute the number of nodes in the graph whose embeddings have less than 100 neighbors with a cosine similarity greater than 0.8. Print the result.

```
''' 6-10 lines of code in total expected but can differ based on your style.'''

# YOUR CODE HERE
print("Number of nodes satisfies the given condition: ", node_filter(another_embeddings, '<100', '>0.8'))
```

```
Number of nodes satisfies the given condition:  53
```