Technical University of Denmark

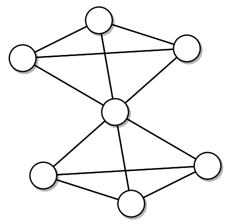
Advanced machine learning. Module 3.

Exercise 1. Graphs and node embeddings.

Contents

Exercise A Node level statistics

Consider the following graph



The following code prints the adjacency matrix and computes its eigendecomposition:

```
>>> print(A)
[[0. 0. 1. 1. 0. 1. 0.]
[0. 0. 0. 0. 1. 1. 1.]
[1. 0. 0. 1. 0. 1. 0.]
[1. 0. 1. 0. 0. 1. 0.]
[0. 1. 0. 0. 0. 1. 1.]
[1. 1. 1. 1. 1. 0. 1.]
[0. 1. 0. 0. 1. 1. 0.]]

>>> lambda, E = np.linalg.eig(A)

>>> print(np.round(lambda, 3))
[ 3.646  2.    -1.646 -1.    -1.    -1.    -1.   ]

>>> print(np.round(E, 3))
[[-0.339 -0.408 -0.228 -0.816  0.004 -0.084  0.126]
[-0.339  0.408 -0.228 -0.    -0.374  0.69   -0.255]
[-0.339 -0.408 -0.228  0.408  0.511  0.114 -0.493]
[-0.339 -0.408 -0.228  0.408 -0.515 -0.03   0.367]
[-0.339  0.408 -0.228  0.    -0.176 -0.709 -0.377]
[-0.558 -0.     0.83  -0.     0.    -0.    -0.   ]
[-0.339  0.408 -0.228  0.     0.55   0.019  0.632]]
```

Question A.1: Determine the eigenvector centrality for each node in the graph.

Question A.2: Determine the clustering coefficient for each node in the graph.

Exercise B Random walks

Question B.1: Given a graph with adjacency matrix $\boldsymbol{A}$ and a starting node chosen randomly according to a discrete distribution $\boldsymbol{p}$, what is the final node's probability distribution after taking a single step from the starting node along an edge chosen uniformly at random?

Question B.2: Given a graph with adjacency matrix $\boldsymbol{A}$, how many distinct paths of length t can we find starting from a specific node (say node 1)?

Hint: We can represent the initial state as a vector that is one for the start node and zero elsewhere. Where can we end up after taking a single step, and how can this be computed using a matrix-vector product? How can this approach be generalized to t steps?

B

Exercise C Shallow embeddings

In this exercise we will use the decoder $\mathrm{DEC}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)$, where $\sigma(x) = \frac{1}{1-e^{-x}}$ denotes the sigmoid function.

Question C.1: As a warm-up, show that $1 - \sigma(x) = \sigma(-x)$.

Based on this, can you spot a typo in eq. 3.12 on page 27 in the book?

Let $S_{u,v} \in \{0, 1\}$ denote a binary feature corresponding to an non-edge/edge between nodes u and v. Let $P_{u,v} = P(S_{u,v} = 1 | \boldsymbol{z}_u, \boldsymbol{z}_v, b) = \sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)$ denote the predicted probability that the edge is present, given the latent node embeddings $\boldsymbol{z}_u$, $\boldsymbol{z}_v$ and bias b.

Question C.2: Write the cross entropy loss for the single observation $S_{u,v}$ (in terms of $P_{u,v}$ and $S_{u,v}$).

Question C.3: Let us assume that the embeddings $\boldsymbol{z}_u$ and $\boldsymbol{z}_v$ are orthogonal, such that their dot product is zero, and let us further assume that the bias is zero, $b = 0$. What is the probability of an edge between node u and v?

C

Exercise D Programming exercise

In this exercise you will work with a shallow node embedding implemented in the script `shallow_embedding.py`. The code loads a graph from a file: This graph is simulated from a shallow embedding model, so that we know the ground truth probability of each possible link. In this exercise we will fit a shallow embedding model to the data and see how well we can estimated the ground truth.

Question D.1: Examine and run the code for loading the graph data.

- Understand how the graph is represented as a matrix as well as in the form of a set of index pairs and target values.
- It can perhaps help to visualize the adjacency matrix.

Question D.2: Examine and run the implementation of the class `Shallow`.

- Understand how the node embeddings are implemented using `torch.nn.Embedding`. Look up the documentation if needed.
- Understand what the forward function computes. What exactly is the role of the variables `rx` and `tx`?

Question D.3: Examine and run the code to fit the model. In this version, the loss is computed on the entire graph (no train/validation split and no mini batching).

- Experiment with different number of `max_step`.
- Experiment with different embedding dimensions. How does the embedding dimension influence the training loss?

Question D.4: Modify the code to use a train/validation split.

- Make a random split of the data (each node pair) into a training set (e.g. 80%) and a validation set (e.g. 20%).
- Modify the code to train on only the training data.
- Write code to compute the loss of the trained model on the validation set.
- Experiment with different embedding dimensions. What is the optimal embedding dimension when computing the loss on the validation set?

Question D.5: Hand in your predictions:

- Using the train/validation procedure you have implemented (or any other updates, hacks and modifications) to optimize the model. Compute what you believe is the best possible predicted link probability.
- Using the provided code, save your predictions in a file, `link_probabilities.pt`, and hand it in on DTU Learn.

I will compute the ground truth loss on your predictions and lowest generalization loss will be honored as the class winner.

D