Machine Learning in Network Science
CentraleSupélec

Lab 2: Link Prediction

Instructor: Fragkiskos Malliaros
TA: Rajaa El Hamdani

21 February 2022

Description

This lab depicts a link prediction task - and is thus supposed to get you ready for the Kaggle challenge.

Two datasets are available. The first network: `karate.gml`, captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club. The second, `socfb-Caltech36.mtx` is a facebook network where nodes represent persons and edges friendship relations between them. It is bigger than the first network and can be found at `http://networkrepository.com/socfb-Caltech36.php`.

Link prediction is the problem of predicting the existence of a link between two entities in a network. It is an important task used in a variety of fields, with a wide range of applications such as friendship recommendation (determine whether two unlinked users are similar enough in the sense that they could become friends).

Part I: Unsupervised link prediction

In this section, you will implement various similarity metrics evoked in the lecture (Jaccard, Adamic-Adar, etc.). You will then use these metrics to predict future edges for the graph. Note that this is a simple unsupervised link prediction task performed with neighbourhood based methods - no representation learning or graph pre-processing is requested.

Exercise 1

1. Load the network data into an undirected graph G, using the `read_edgelist()` function, similarly to Lab1.
2. Complete the `preferential_attachment()` function, which computes a similarity metric between any two non-edges[1] of the graph. You might find the networkx function `non_edges()` useful. As a reminder, the preferential attachment similarity formula for two nodes u and v is:

$$C(u,v) := |\Gamma(u)| \cdot |\Gamma(v)| \tag{1}$$

3. Complete the `Jaccard` function, which computes the Jaccard similarity metric between every

[1] edges (or node pairs) not belonging to the original graph

pair of non-edges. The related formula is:

$$C(u, v) := \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{2}$$

4. Complete the `Adamic-Adar` function, which computes the Adamic-Adar similarity metric between any pair of non-edges. The related formula is:

$$C(u, v) := \sum_{z \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(z|} \tag{3}$$

5. Use one of the metrics implemented above to predict 10 future edges for the graph. In practice, you would like to output the node-pairs of the graph presenting the highest similarity score. *Note*: it might be desirable to randomly shuffle the similarity scores (contained in the dictionary).
6. (Optional): Evaluate and compare the efficiency of the approaches implemented above. To do so, we can remove at random some edges of the graph while leaving the graph connected, and apply one of the above method on non-edges of the graph. We observe the top-k node-pairs with highest similarity score and calculate how many are part of the removed edges. This yields an accuracy metric.

Part II: Supervised Link Prediction

In this section, we first pre-process the input graph so as to formulate the link prediction problem as a supervised machine learning task. We then create a feature vector for each edge and use it in a downstream traditional machine learning task to predict if an edge should included in the graph or not - binary classification problem. More precisely we adopt the following experimental set-up:

- We remove a certain fraction of edges chosen randomly from the network in order to construct positive samples (real existing edges - label 1) for the testing set. *Note*: the network remains connected during the process.
- The remaining edges in the residual graph constitute positive samples for the training set.
- We create negative samples by randomly sampling non-existing edges in the original network (non-edges, label 0). We sample as many negative samples as there are of positive samples, for both the training and test sets. This yields a balanced dataset, otherwise we would have much more negative than positive samples.
- As the supervised learning task is formulated, we can now learn node representations using the residual network. Node representations can be created/learned in various ways: handcrafted manually using carefully targeted graph properties and node features' information (if there is), via random walks or using deep learning approaches. In this lab, we use the first option.
- We define manually the feature vector of each edge (v, u) using several graph properties (node similarity measures, node centrality measures, etc.)
- In the experiments, we adopt logistic regression with $L2$ regularization as downstream machine learning model, built on the edge feature vectors obtained previously. Ultimately, we obtain a binary classification score for training/test samples, stating if an edge was/should be added to the network or not.

Connecting this with the Kaggle competition, edge features should (when possible) be computed using summaries of node features. Indeed, nodes are often endowed with useful information that is essential for link prediction tasks. In the Kaggle dataset, nodes have textual information, which is interesting to process and exploit inside edge feature vectors. This will be further developed in the next practical session.

Exercise 2

1. In the `generate_samples` function, generate positive samples to constitute the test set. More concretely, remove randomly $test\_set\_size$ edges from the graph while preserving connectedness. *Tip*: use the `remove_edge()` attribute of the graph. Further down in this function, generate negative samples for testing and training sets (see above description).
2. In the `feature_vector()` function, fill in the blanks to compute several features for an edge, which ultimately form its handcrafted feature vector. You can add various features to improve this feature vector, and thus probably the classification performance.
3. In the `prediction` function, build and train a Logistic Regression model on edge feature vectors, before computing the associated roc auc score. The associated packages are already imported.
4. Perform the link prediction experiment for a training ratio of 80% as described above and plot the ROC curve. How does AUC score behave depending on the features included in the edge feature vector ? *Go futher:* Could you spot important features?

.