Machine Learning In Network Science

Predicting missing links in an actor co-occurrence network

Raphaël FERRONI raphael.romandferroni@student-cs.fr
CentraleSupélec

Alexandre SELVESTREL alexandre.selvestrel@student-cs.fr
CentraleSupélec

Abstract

In this project, we are looking to predict the existence or non-existence of links between the nodes of an actor graph. The aim is to tell whether the two actors are on the same Wikipedia page (in which case the link exists) or whether this is not the case (no link). We have tried two methods. The first is based on classic machine learning, the second is a graph auto-encoder. However, only the first method gives satisfactory results, particularly when several different models are trained (XGBoost, Random Forest, logistical regression, etc.). In this report, we present our models and results.

1 Data Exploration

The dataset comprises node information, including textual features extracted from Wikipedia pages, and edge information representing co-occurrence on Wikipedia pages. We performed an initial exploration to understand the dataset characteristics.

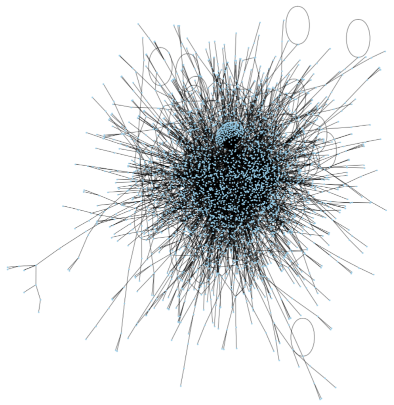

Figure 1: Graph Visualisation

With 3597 nodes and 5248 edges, the network exhibits a substantial but not overwhelming size, indicating a considerable amount of connectivity among its constituents. The network's density of 0.000811 underscores its sparse nature, indicating that only a small fraction of possible connections among nodes are realized.

The analysis of node features in the context of the network reveals an interesting challenge posed by the distribution of shared keyword positions in linked and unlinked pairs. The distribution of shared keyword positions is nearly equivalent between linked and unlinked pairs with a combined average of approximately 0.83 shared keywords, highlighting a potential issue in distinguishing between these two types of pairs based solely on text-based features. The Figure 2 is quite telling on the distribution of these keywords. Few of them are overwhelmingly surrepresented among the other.

To overcome this challenge, we thought to apply dimensionality reduction techniques such as PCA (Principal Component Analysis) or merely removing the top k most represented keywords from the node feature information but these two techniques failed to show their ability to improve the capability of distinguish between these two types of pairs.

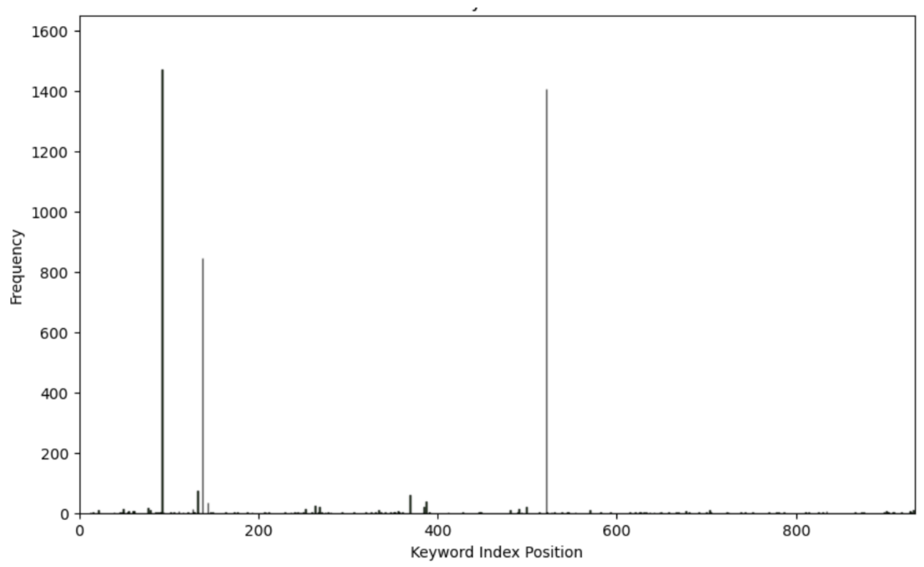

Figure 2: Distribution of shared keyword positions in linked/unlinked pairs

2 Machine Learning Approach

In this section, we present the classical machine learning approach used for link prediction in the actor graph dataset.

2.1 Feature Extraction

With the help of Zhang (2022), we extracted features for each edge sample to capture both network topology and textual information. Most of them are heuristic features. Latent-based features using DeepWalk and Node2Vec failed also to show their capability to improve the model. The complete description of the **28 features** can be seen in the Appendix (see 1).

In Figure 3 we can visualise the correlation matrix between these features. Features such as current flow closeness centrality, harmonic centrality, degree centrality, PageRank, and load centrality in both source and target nodes demonstrate high positive correlations, indicating that nodes with higher centrality and importance metrics are more likely to be associated with the target outcome.

Interestingly, the co-occurrence of keywords (calculated with the dot product between the two source and target vectors) shows a weak negative correlation with the label, suggesting that certain characteristics related to keyword associations may be associated with a lower likelihood of the target outcome. This is mainly in relation with the specific distribution of shared keyword positions depicted earlier.

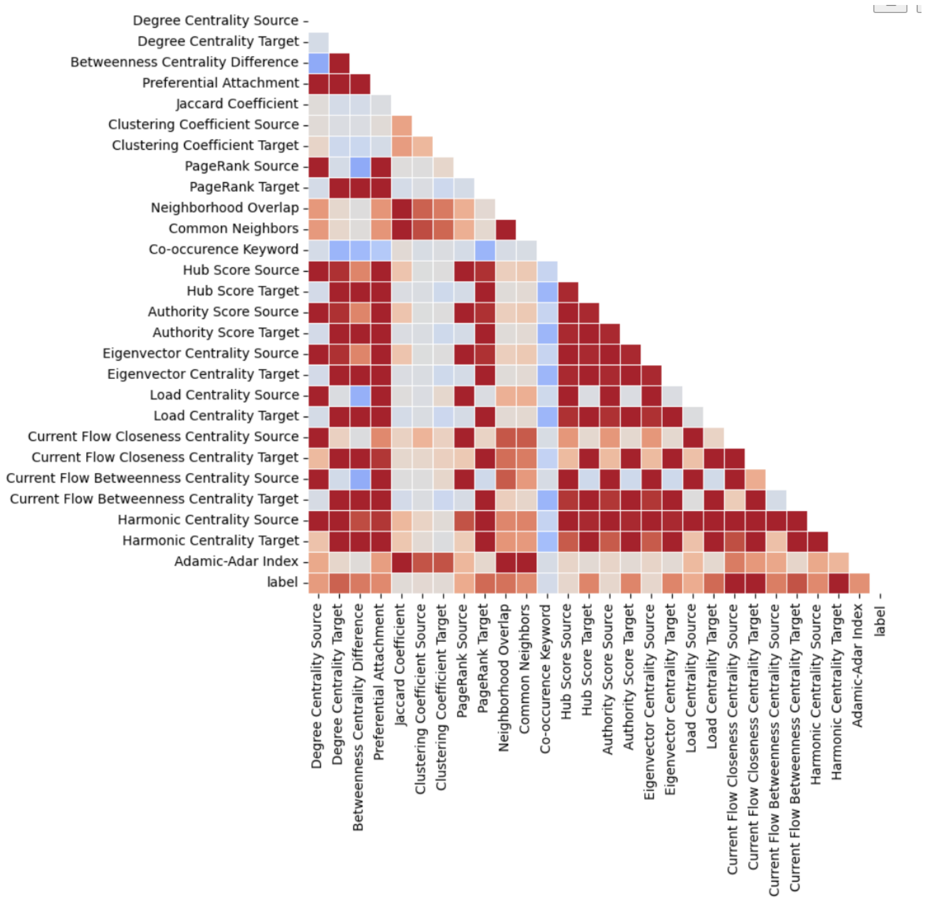

Figure 3: Correlation matrix of the features

2.2 Model Training and Validation

To identify optimal model parameters, a comprehensive approach involving cross-validation and hyperparameter tuning was employed. Initially, each model, including Logistic Regression, Support Vector Classifier (SVC), Random Forest, and XGBoost, was instantiated with a wide range of parameter values. The final models chosen for the Voting Classifier are listed in 2.

The dataset was divided into training and testing sets to evaluate our models effectively. We then prepared the features by standardizing them and reducing their dimensionality using PCA. Using the elbow method on the scree plot (see Figure 4), we have chosen 4 as the optimal number of components.

Eventually, we created a soft voting classifier, which combines the strengths of our best models. This ensemble was trained on the transformed features and evaluated on the test set. By pooling the insights of multiple models, the voting classifier demonstrated its ability to predict accurately on new data, enhancing our overall predictive performance and make our best score of 76.45

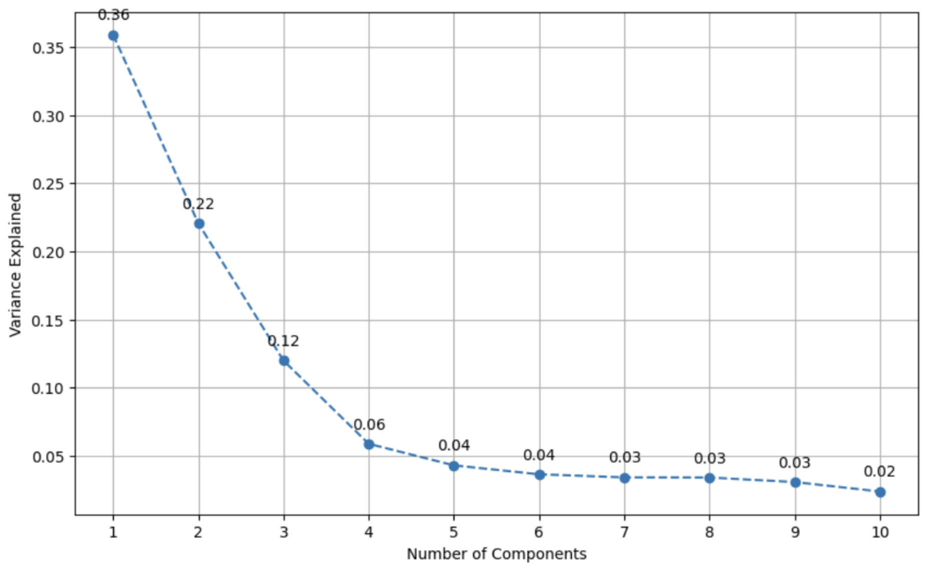

Figure 4: Scree Plot

2.3 Final Model and Predictions

The final model was trained on the entire dataset without validation (setting the parameter submission to True). Test features were extracted, scaled, and used to make predictions using the trained model. We carefully extract the self-loop edges from test data for later inclusion with label equal to 1. Eventually, a submission file was prepared for the Kaggle competition.

3 Graph Auto Encoder (GAE)

3.1 Architecture

We attempted to leverage graph networks to effectively utilize the 932 features of each node. To achieve this, we opted for Graph Autoencoders (GAE) due to their simple architecture, which efficiently computes node embeddings, thus replacing the need for feature engineering. Despite its simplicity, research by Kipf and Welling (2016) suggests that this model yields results nearly as impressive as Variational Autoencoders (VAE) on datasets like Cora or Citeseer, boasting an average precision of over 90%. As outlined by Kipf and Welling (2016) and Zhang (2022), the architecture is as follows:

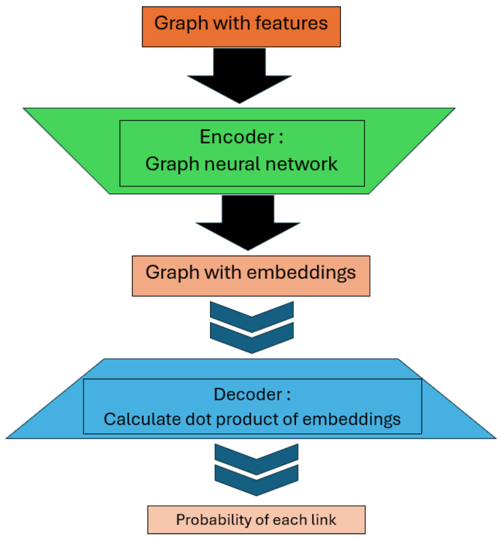

Figure 5: Diagram of a GAE

The graph neural network used is a three-layer graph convolutional network (GCN). We use dropout (50%) on the intermediate layers to try to help generalisation. The loss used is the binary cross entropy between the predicted probabilities and the true edges. The learning rate is set to 0.01. During training, the edges are separated into training edges (90%) and testing edges (10%).

3.2 Results

The results obtained are notably weak, with poor accuracy observed. Notably, there appears to be a slight decline in accuracy on the testing set as training progresses, while it approaches unity on the training set, indicative of overfitting. Similar outcomes were observed when employing a Variational Autoencoder (VAE) from the Torch Geometric library (as we were unable to locate a straightforward Graph Autoencoder (GAE) implementation). Despite rigorous code review efforts to identify potential issues in the datasets, no improvements were observed. Additionally, attempts to mitigate overfitting by introducing a regularization parameter (weight decay) to the optimizer yielded no significant changes. Furthermore,

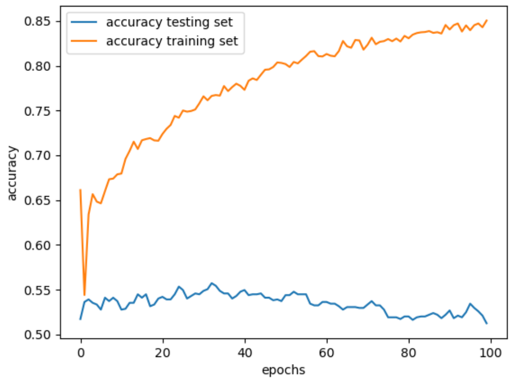

Figure 6: Accuracy changes with training

integrating the GAE into the voting classifier alongside traditional machine learning models failed to enhance accuracy, rendering the GAE impractical for our purposes.

4 Appendix

Table 1: Feature Descriptions

Feature Name	**Description**
Degree Centrality (Source/Target)	The degree centrality of the (source/target) node, i.e., the fraction of nodes the (source/target) node is connected to.
Betweenness Centrality Difference	The difference in betweenness centrality between the target and source nodes.
Preferential Attachment	The product of the degrees of the source and target nodes.
Jaccard Coefficient	The ratio of the number of common neighbors to the total number of neighbors of the source and target nodes.
Clustering Coefficient (Source/Target)	The clustering coefficient of the (source/target) node, i.e., the fraction of triangles the (source/target) node participates in.
PageRank (Source/Target)	The PageRank score of the (source/target) node, reflecting its importance in the network.
Neighborhood Overlap	The number of neighbors shared between the source and target nodes.
Common Neighbors	The number of common neighbors between the source and target nodes.

Continued on next page

Table 1 – continued from previous page

Feature Name	**Description**
Co-occurrence Keyword	The dot product of text-based features of the source and target nodes.
Hub Score (Source/Target)	The hub score of the (source/target) node, indicating its authority in directing traffic to other nodes.
Authority Score (Source/Target)	The authority score of the (source/target) node, indicating its importance as an information source.
Eigenvector Centrality (Source/Target)	The eigenvector centrality of the (source/target) node, considering the importance of its neighbors.
Load Centrality (Source/Target)	The load centrality of the (source/target) node, reflecting its importance in transporting information in the network.
Current Flow Closeness Centrality (Source/Target)	The current flow closeness centrality of the (source/target) node, measuring its centrality in the network flow.
Current Flow Betweenness Centrality (Source/Target)	The current flow betweenness centrality of the (source/target) node, measuring its importance in controlling the network flow.
Harmonic Centrality (Source/Target)	The harmonic centrality of the (source/target) node, reflecting its centrality based on harmonic mean distances to other nodes.
Adamic-Adar Index	The Adamic-Adar index between the source and target nodes, measuring their similarity based on shared neighbors.

Table 2: Model Parameters

Model	**Parameters**

Logistic Regression	C=0.01,
	max_iter=10000,
	penalty='l1',
	random_state=42,
	solver='liblinear'

Support Vector Classifier	C=100,
	probability=True,
	random_state=42

Random Forest	max_depth=20,
	min_samples_leaf=2,
	min_samples_split=5,
	n_estimators=80,
	random_state=42

XGBoost	colsample_bytree=0.8,
	gamma=0.1,
	learning_rate=0.1,
	max_depth=7,
	min_child_weight=5,
	n_estimators=100,
	random_state=42,
	...

References

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

Muhan Zhang. *Graph Neural Networks: Link Prediction*, pages 195–223. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-16-6054-2. doi: 10.1007/978-981-16-6054-2_10. URL `https://doi.org/10.1007/978-981-16-6054-2_10`.