# Introduction

The below suggested method has been used for detecting missing links in a co-authorship network/graph. The missing link problem has been dealt with as a supervised classification problem. Wide range of classifiers and techniques such as Logistic Regression, Gradient Boosting, Random Forests, Node Embeddings, LSTMs have been employed to find the best fit.

# Approach

Converting the given problem into a classification problem starts with producing positive and negative samples for binary classification.

To establish a baseline model, features given in **train.txt** and **nodes.json** such as Common Neighbor count, Intersection in keywords and venues, Cosine Similarity between keywords and venues have been used with Logistic Regression.

The inspiration for the final chosen approach comes from sequence classification. Each edge’s connecting nodes are taken as sequence with node embeddings using node2vec (Grover Aditya, 2016) as weights. These features are then fed to a bidirectional LSTM to extract meaning behind the pattern. Once trained, the same technique is used to produce test features and predict.

## sampling

* As the given graph already had a few links removed all the edges provided have been considered as positive samples. These samples have a label “1”.
* Negative samples are chosen if there exists no direct and indirect link between the chosen nodes or if a link exists, the shortest path length is more than 3. To deal with class imbalance, samples twice as large as the positive samples have been randomly sampled (Leskovec J, 2006). These samples have a label “0”.
* The dataset has been split into standard 80-20% to produce training and validation data.

## feature selection

* For preliminary analysis, number of common neighbors, number of total neighbors, cosine similarity between keywords and venue have been used.
* The given author graph is used to produce embedding vectors using node2vec. Node2vec produces embeddings using random walks as skip grams to train Word2Vec model. These embeddings act as weights for each node.
* For each edge present in positive and negative samples, all neighbors of two nodes are chosen as features. Each feature vector is padded with a key which has zero weight in the embedding space to produce vectors of consistent length.

## model architecture

* Tf.Keras has been used to develop the said LSTM. With Embedding layer at the top, a bidirectional LSTM (Martin Sundermeyer, 2012) to cover nodes emerging from both sides of the edge from graph, the results are subjected to MaxPooling and AveragePooling layers. The results are concatenated and flattened. After flattening, the output is mapped to a Dense layer with relu activation. The output is further squeezed into a dense layer of size to with sigmoid activation function.
* Binary\_crossentropy is chosen as the loss function and Adam (Diederik P. Kingma, 2015) has been opted as optimizer function.

# Alternate Approaches

Apart from above mentioned approach, the following approached have been tried

* Features specific to a graph such as common neighbors, Jaccard coefficient, node cosine similarity, Preferential Attachment, Adar-Adamic Index, Resource Allocation, Neighborhood Distance and Community Belonging (cdiscountdatascience, n.d.) (Michaela Hoffman, 2015) have been used as features. Classifiers such as RandomForestClassifier, GradientBoostingClassifier, XGBClassifier has been used to training. GridSearchCV has been used for hyperparameter tuning (JAIN, 2016 )and cross validation. Surprisingly, the AUC score never crossed 0.7. Feature correlation has also been checked to avoid redundant features.
* Edge embedding using HadamardEmbedder have been used to produce edge features and the same models mentioned above along with Deep Dense Net with 4 hidden layers has been used. The yielded AUC score ranged between 0.89 - 0.91
* Features from nodes.json along with XGBClassifier produced an AUC score of 0.87 despite the simplicity of the features.

# Key Takeaways

The final approach taken produced an AUC score of 0.92782. It can be understood from it that while it was successful in capturing pattern from most of the edges, it failed to extract minor outliers.

The score could have improved by trying an alternate architecture and using hyperparameter tuning on neural networks, but it is very computationally expensive.

There may have been a variation in score if the data is sampled based on structure, namely Barabási-Albert preferential attachment model or Uniform Random Sampling.

# References

cdiscountdatascience. (n.d.). *hackernoon*. Retrieved from Link Prediction In Large-Scale Networks: https://hackernoon.com/link-prediction-in-large-scale-networks-f836fcb05c88

Diederik P. Kingma, J. B. (2015). Adam: A Method for Stochastic Optimization. *3rd International Conference for Learning Representations.* San Diego.

Grover Aditya, L. J. (2016). node2vec: Scalable Feature Learning for Networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

JAIN, A. (2016 , March 1). *Complete Guide to Parameter Tuning in XGBoost with codes in Python*. Retrieved from analyticsvidhya: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Leskovec J, F. C. (2006). Sampling from large graphs. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, (pp. 631-636). New York.

Martin Sundermeyer, R. S. (2012). LSTM Neural Networks for Language Modeling. *INTERSPEECH 2012*, (pp. 194-197). Portland.

Michaela Hoffman, D. S. (2015). A Note on Using the Adjusted Rand Index for Link Prediction in Networks. *Soc Networks.*, (pp. 72–79.).