Link Citation Prediction

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Evangelos Andreas Venetsanos • ◦ **3180019**
Dimitris Alexakis • ◦ **3160003**
Petros Chanas • ◦ **3170173**

Abstract—In Network Theory, Link Prediction is the problem of predicting the existence of a link between two entities in a Network. Examples of Link Prediction include predicting friendship links among users in a Social Network, predicting co-authorship links in a Citation Network, and predicting interactions between genes and proteins in a Biological Network.

The goal of this report is to present a Model for predicting Citation Links in a Citation Network and the process by which we developed it.

All relative files can be found in this repository.

1 Introduction

1.1 Description of the Challenge

In this challenge, you will deal with the problem of predicting whether a research paper cites another research paper. You are given a citation network consisting of several thousands of research papers, along with their abstracts and their list of authors. The pipeline that is typically followed to deal with the problem is similar to the one applied in any classification problem; the goal is to use edge information to learn the parameters of a classifier and then to use the classifier to predict whether two nodes are linked by an edge or not.

Next, we also provide a simple approach, on which you can work. The key component is how we create our training and test data. For training, you may first create a numpy array of zeros with 2m rows and n columns (where m is the number of edges in the edgelist file, and n the number of desired features). We then need to populate this array to create the training features. A way to do that is to iterate over all edges in the edgelist file, and if we want to have 3 features: we get the degree of the first node of the edge as the 1st column, the degree of the 2nd node of the edge as the 2nd column and their sum as the 3rd column. For the remaining m rows, we select randomly nodes and get their degrees and sum as features as well. Now we have our matrix for training. We also need to create the y classes. We create a vector of zeros with size 2m. The first m elements represent the edges that are present in the edgelist. Thus we make them 1. The remaining need to be 0, as they represent non-existing edges. We are now ready to give the features and classes to a scikitlearn classifier via the 'fit' method. Afterwards, we only need to create a similar feature matrix for the test file. We then need the probabilities of our predictions. Remember to get the probabilities only for the positive class. Please, check the 'submissionsrandom' file to check if the format matches your output. The 1st column in the submissions file represents an id integer of the edges in test file. 0 points to the first edge of the test file and so on.

1.2 Evaluation of the Model

Given a pair of nodes, your model should predict whether the two nodes are linked by an edge or not. The evaluation metric for this competition is the log loss. This metric is defined as the negative log-likelihood of the true class labels given a probabilistic classifier's predictions. Specifically, the log loss is given by:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

where N is the number of samples, y_i is 1 if sample i belongs to class 1 and 0 otherwise, and p_i is the predicted probability that sample i belongs to class 1. The lower the score, the better.

1.3 General Approach

Our Data-set can be split in three distinct parts: the Edgelist , the Abstracts and the Author lists.

- **edgelist.txt**
  Every Citation (edge) between two papers (nodes). Each edge is represented by two paper ids, separated by a comma, for example **id1, id2**.

  We'll be referring to Paper id1 as the First Paper and Paper id2 as the Second Paper of that Edge.

- **abstracts.txt**
  The Abstract of every paper. We can apply a pre-proccessing technique to each one and then map it to it's corresponding Paper. We'll be presenting the technique we used later in this report.

- **authors.txt**
  The Author list of every paper. We can map each one to it's corresponding Paper.

In order to improve the Accuracy of our Model we need to take advantage of these three different aspects of our Data-set by combining them.

2 Development of the Model

2.1 Baseline Solution

Our first attempt at arriving at a better solution was tweaking the Baseline Solution provided to us by using another Model that's available through Python's Scikit-learn. Some of these Models and their corresponding Accuracies when Trained and Tested on the Features included in the Baseline Solution can be seen in Table 1.

Model	**Accuracy**
Linear Regression	0.7146276956214352
Logistic Regression	0.7174593115473518
Random Forest Regressor	0.7006692577356597
KNeighbors Regressor	0.6827436270325911
Decision Tree Regressor	0.6990153449406387
Xgboost Regressor	0.6731901285399778
LassoCV	0.717446490517933
Voting Regressor	0.7065284681800366
Average Accuracy	**0.70146004051**

TABLE 1: **Models and their Accuracy**

Note: The list of Regressors used by the Voting Regressor contains the following Models: Linear Regression, Random Forest Regressor, KNeighbors Regressor, Decision Tree Regressor, LassoCV and Xgboost.

Regardless of the Model that was used, the Accuracy remains similar. Therefore, transforming the Training Data we use is necessary in order to improve the Model.

2.2 Extracting Features from our Data-set

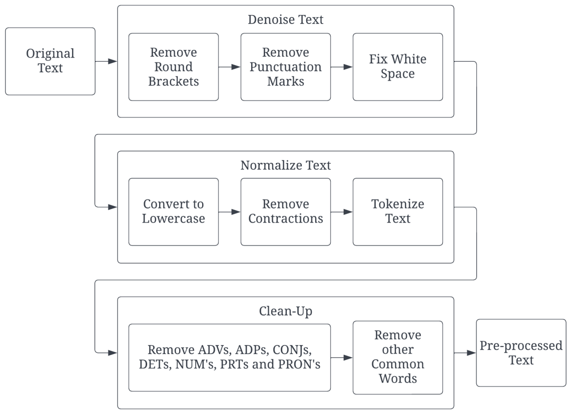
Firstly, we attempted to improve our Model by extracting additional features from our Data-set. For example, we calculated to number of common authors between each pair of papers.

We were able to use the edgelist to come up with most of the Features we ended up using in the final version of our Model. Throughout the development phase of our project we tested different combinations of those features. For example, calculating the PageRank only for the Second Paper for each Citation Link but we ended up finding that including the PageRank of both Papers produces better results.

However, not all features we were able to extract helped improve our Model. We calculated the shortest path between each pair of Papers and we also attempted to use Pyhton's node2vec. In both of those examples we saw that the Accuracy of our Model diminished.

As we've already mentioned, the Abstracts included in our Data-set need to be pre-processed. The reason why this approach is necessary is the size of our Data-set and the fact that certain common words that appear in most Abstracts would make it harder to accurately gauge the similarity of pair of them. In essence, using the raw version of Abstracts would be time consuming and it would make our Model less accurate.

We can overcome this obstacle by taking advantage of Python's Natural Language Toolkit (NLTK). The pre-processing pipeline we developed can be seen below.



Now we will present an example of the technique we described. We'll be applying those steps on the sentence *"The development of an automated system for the quality assessment of aerodrome ground lighting (AGL), in accordance with associated standards and recommendations, is presented."*. The pre-processing pipeline can be seen in Table 2.

Original Text	The development of an automated system for the quality assessment of aerodrome ground lighting (AGL), in accordance with associated standards and recommendations, is presented.
Remove Round Brackets	The development of an automated system for the quality assessment of aerodrome ground lighting , in accordance with associated standards and recommendations, is presented.
Remove Punctuation Marks	The development of an automated system for the quality assessment of aerodrome ground lighting in accordance with associated standards and recommendations is presented
Fix White Space	The development of an automated system for the quality assessment of aerodrome ground lighting in accordance with associated standards and recommendations is presented
Convert to Lowercase	the development of an automated system for the quality assessment of aerodrome ground lighting in accordance with associated standards and recommendations is presented
Tokenize Text	the development of an automated system for the quality assessment of aerodrome ground lighting in accordance with associated standards and recommendations is presented
Remove ADVs, ADPs, CONJs, DETs, NUM's, PRTs and PRON's	['the', 'development', 'of', 'an', 'automated', 'system', 'for', 'the', 'quality', 'assessment', 'of', 'aerodrome', 'ground', 'lighting', 'in', 'accordance', 'with', 'associated', 'standards', 'and', 'recommendations', 'is', 'presented']
Remove other Common Words	['development', 'automated', 'system', 'quality', 'assessment', 'aerodrome', 'ground', 'lighting', 'accordance', 'associated', 'standards', 'recommendations', 'is', 'presented']

TABLE 2: **Application of pre-processing technique**

Note: The meaning of the NLTK Tags mentioned in Table 2 can be seen in Table 3.

2.3 Testing Models

Utilizing Python's Scikit-learn, we were able to split our Data-set into Training and Testing data. That way we can figure out which Model works best with our Features. Throughout this report we've mentioned a number of Models that we took advantage of while developing

NLTK Tag	Meaning
ADV	Adverb
ADP	Adposition
CONJ	Conjunction
DET	Determiner, Article
NUM	Numeral
PRT	Particle
PRON	Pronoun

TABLE 3: **NLTK Tags and their meaning**

our Model. Furthermore, by submitting predictions to Kaggle we're also able to compare each Model and their combinations (eg., Voting Regressor).

3 Final Model

As we discussed beforehand, the Final Model we developed is based on the Baseline Solution. The set of features we used as well as the specifications of our Model will be presented below.

3.1 Features

The set we used is a combination of Features we extracted from the Edgelist, the Abstracts but also the Author lists.

- **Semantic Similarity of Abstracts**
  Word2vec (w2v) is a Natural Language Processing technique used to produce Word Embedding. The model represents each word by a vector and therefore can calculate the level of Semantic Similarity between those words.
  If we collect every vector that represents a word in a text and then calculate their mean, we will end up with a vector representing the whole text. We can then repeat the process one more time and then by calculating the Cosine Distance of those Vectors, we can determine the Similarity of the two texts.

- **Jaccard Index of Abstracts**
  Calculating the Jaccard Similarity Coefficient of two Abstracts requires pre-processing them, similarly to calculating their Semantic Similarity.

- **Jaccard Index of Author Lists**
  Each Author List is converted into a Set, allowing us to calculate their similarity.

  Note: Jaccard Similarity Coefficient
  The Jaccard Index can be used to gauge the similarity and diversity of two sets.

- **PageRank of First/Second Node**
  By utilizing Python's NetworkX we can calculate the PageRank for each Node in our Graph.

- **Hubness of First Node**
  Hubness of the 'citing' paper for each citation.

- **Authority of Second Node**
  Authority of the 'cited' paper for each citation.

  Note: Hyperlink-Induced Topic Search
  In HITS algorithm two values are calculated for each Node of our Directed Graph.

  – **Authority**: Equal to the sum of the Hubness scores of each node that points to it.

  – **Hubness**: Equal to the sum of the Authority scores of each node that it points to.

- **Out-Degree of First Node**
  Out-Degree of the Starting Node for each Edge in our Graph.

- **In-Degree of Second Node**
  In-Degree of the Ending Node for each Edge in our Graph.

- **Max k-Core for the First Node**
  k-Core number of the First Node of each

Edge.

- **Max k-Core for the Second Node**
  k-Core number of the Second Node of each Edge.

  Note: k-Core
  A k-Core graph is a maximal subgraph that contains nodes of degree k or more. The k-Core number of each Node is the highest k for which a node is present in that graph.

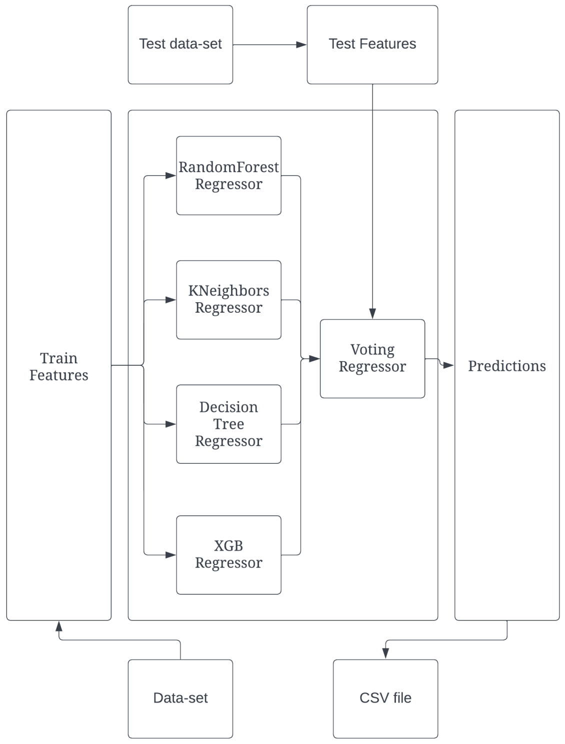
- **Number of Common Neighbors**
  The number of common neighbors between the two Nodes.

3.2 Model

After some experimentation, we decided to utilize a Voting Regressor with a list of Regressors that includes the following models:

- Random Forest Regressor
- KNeighbors Regressor
- Decision Tree Regressor
- XGB Regressor

Below we can see the way each aspect of our Model interacts with each other.



4 Further Exploration

As always we approach the 3 distinct aspects of our Data-set separately. Firstly, extracting more relevant and useful features from the graph we construct through the edgelist provided to us would also improve our Model significantly. Regarding the Abstracts, tweaking the parameters used when building the w2v Model or improving the pre-processing technique we developed would help us determine their Semantic Similarity more accurately. Lastly, we could utilize the Author Lists more. For example being able to extract ′friend′ groups from our set of Authors would help us more accurately gauge the possibility of a Citation Link existing between two Papers co-written by them.

5 Closing

Predicting whether a Citation Link exists between two papers is a significant analysis task and a fundamental link prediction problem. The approach we utilized prioritizes an increase to the Accuracy of the Model by using a combination of Features from each aspect of our Data-set. The final version of our Model and all relative files are available through our Github Repository.