Antoine CADIOU
antoine.cadiou@ens-paris-saclay.fr
Léo TRONCHON
leo.tronchon@ens-paris-saclay.fr

ALTEGRAD 2021
Challenge: Citation Prediction
Team: A&L

04/22/22

# 1 Introduction

In this project, we are asked to predict whether a paper has cited another paper. To do so, we are given information about the abstracts and authors of the paper, as well as a graph of citation between papers with the test edges missing.
Predicting whether a paper will cite another could give information, in advance, about the potential impact of a publication in the scientific community. This problem is also very interesting because it mixes two different modalities, Natural Language Processing data (knowing the authors & abstracts of each paper) and Graph data.
Our objective here will be to design features and a neural network architecture for link predictions on the Citation Graph.

# 2 Analyzing baselines

The two given baselines gave us the global direction to follow and all our proposed solution will be based on designing a vector of features corresponding to an edge, before applying a classifier in the form of a logistic regression or a Neural Network for link prediction.

- Text baseline
  Given 2 publications' abstracts we name $s_1$ and $s_2$ the two sets of words in each abstract. 3 features are computed:
  - the sum of lengths for $s_1$ and $s_2$
  - the length of the union between $s_1$ and $s_2$
  - the length of the intersection between $s_1$ and $s_2$

- Graph baseline
  Given a graph of publications in which nodes represent an abstract and edges represent citations between 2 publications, for each edge composed of 2 nodes $n_1$ and $n_2$, 2 features are computed:
  - the sum of degrees for $n_1$ and $n_2$
  - the difference of degrees for $n_1$ and $n_2$

The first idea that came in mind was to simply concatenate the features in the two baselines, so we have 5 features for each edge. This achieved a good improvement (see 10).
Nevertheless we see quickly that some important information is missing, such as the authors (we are not using them). Also, we can improve the features by adding the embedding of abstracts or nodes. We will see in the next part what is our solution for that.
Note that in the next parts we will always concatenate these 5 features with the others.

# 3 Abstract Embeddings

## 3.1 Doc2Vec

We knew that it could be useful to extract meaningful information from the abstracts. For that we had to find a way to compute their embeddings. Initially we thought of Word2Vec but we faced an issue: abstracts are composed of multiple sentences, so we needed to find an embedder that was capable to encode this format. Initially, we chose Doc2Vec [4] which is doing exactly what we were looking for.
Finally, after having trained our model, we were able to find the embedding for a given node.
The last step was simply to concatenate the 2 embeddings corresponding to the 2 nodes from the edge we try to predict.
For instance, let $D_{d2v}$ be the dimension of the latent space for an embedding obtained with Doc2Vec, we have $2 * D_{d2v} + 5$ features for each edge in $X$ (the final matrix on which to run the logisitic regression).

### 3.2 Specter

In an effort to get embeddings more informative and tailored to our task, we found the paper SPECTER [1]. SPECTER is a recent method which can generate document-level embedding of scientific documents based on pretraining a Transformer language model on a powerful signal of document-level relatedness: the citation graph. The fact that the model was pretrained on scientific articles and more specifically on their citation graph is crucial, and also the main reason for its good performance in our task.
Therefore we used the HuggingFace implementation of SPECTER to compute the embeddings of each abstract in the dataset.

### 3.3 Neighborhood Embeddings

Using the abstracts embeddings of the nodes of interest is important, but it is a bit limiting. The neighbors of each node of interest contain important information as well to consider. However, their number can vary and using the embeddings of each neighboring node would require too much RAM anyways. Therefore we must aggregate the embeddings coming from neighboring nodes. There are 3 common ways of aggregating embedding information: using the mean, using max-pooling, and using the sum. The sum led to a strong overfit on the training/validation graph, and so did the max-pooling to a lesser extent. However, the mean pooling did not display the same behavior and allowed us to beat our best model.
However, one has to be careful when using information from the neighboring nodes. When computing those embeddings, we do not want to include information about the existence of a link or not between the nodes of interest, and therefore we exclude their embeddings from the pooling operation to avoid overfitting the training/validation graph.

## 4 Nodes Embeddings

### 4.1 Node2Vec

On top of using text embeddings obtained from the abstracts, we wanted to get embeddings corresponding to the graph structure. Therefore we decided to make use of the paper Node2Vec [2].
As for the abstracts' embeddings, we had to think of a way to merge them. We thought of using the Hadamard transform but it was not as good as expected. After that we tried two approaches: concatenate them or use the mean of the 2 vectors. We finally kept the last one because the results were better.
Be $D_{n2v}$ the dimension of the latent space for an embedding obtained with Node2Vec, we have $D_{n2v}$ features for each edge in the dataset $X$.
The score was definitely not as good as our current solution (with Specter), but we have observed animportant detail: the predictions are either very small (close to 0) or very high (close to 1). That means the model, has overfitted the training/validation set, but fails to generalize well to the graph structure when the test edges are added.

### 4.2 Graph Autoencoders

#### 4.2.1 KMeans on node embeddings

Our very first idea (even before using baselines and doing supervised learning) was to use Graph autoencoders to obtain embeddings of a node. Then, we grouped the nodes by clusters with KMeans and predicted the existence of a link between 2 nodes if they belonged to the same cluster.
Obviously this technique has many drawbacks, for example we had to choose a number of clusters "k" and we had no idea how to initialize it. To fix the k , we wanted to try to visualize the graph, but as it was too large to draw with networkx, we tried with the Gephi software. We were hoping to be able to visualize the graph and see that there were, for example, 20 clusters (to set k to 20) but it never worked. So we set k to 100.
In the end, the score obtained by this method was much worse than the one obtained by the basleines, so we decided not to continue in this direction.

#### 4.2.2 Graph Convolutional Networks for Node Features

Using graph autoencoders to obtain meaningful features about the nodes seemed like a very reasonable thing to do as it should allow us to use the graph structure's information to make predictions. Therefore, we used Pytorch Geometric and tried to combine GAEs and VGAEs' [3] embeddings with those obtained with SPECTER. However, they failed to improve the results, regardless of the Graph Convolution tested (GCN, GraphSageConv, TransformerConv ...), the embeddings initialization used (SPECTER embeddings, random embeddings), or the

size of the embeddings (64,128,768). Therefore we did not end up using those embeddings to train our best models.

# 5 Authors

We have information about the authors of each abstracts, so we tried to leverage it. We made a weighted graph with links based on the authors. Each abstract was a node, each link meant the abstracts had authors in common, and each weight corresponded to the number of authors in common. However, this graph had about 8 times more edges that the citation graph and therefore it was computationally demanding to get embeddings. We tried to use GAEs and VGAEs, on a subset of the graph, but it didn't help. Then we tried to use the standard Node2Vec implementation, which failed due to RAM constraints. Finally we found a very efficient implementation of node2vec from a package called fastnode2vec, but it failed to produce any improvement when combined with SPECTER embeddings and therefore was not used in our best model.

In the end we used only 2 handcrafted features related to the authors: the 2 nodes' number of authors, and the number of authors they have in common.

# 6 Graph Features

In the paper [5], the authors define multiple graph features that we added to the ones from the graph baseline. e have implemented the following ones:
- Common neighbors of degree 1
- Katz (degree 2), wich is based on the same principle of common neighbors but with a penalization on the depth(degrees) of the neighbors (for instance, common neighbors of degree 5 will not be as important as the ones of degree 2). We took $\gamma = 0.5$.

$$\sum_{d=1}^{degree\_max} \gamma^d \, . \, |neighbors_{n1}^{degree=d} \cap neighbors_{n2}^{degree=d}|$$

- Jaccard
- Preferential attachment
- Cosine similarity

The results improved to different degree depending on the features they were concatenated for both methods. They are visible on Table 10.
We can conclude that these tabular features are extremely important (especially Katz)

# 7 Dataset

When creating our dataset, we created an edge for every edge in the graph to which we attributed the label 1, as well as a negative edge which was a random edge to which we attributed the label 0. We tried to increase the diversity of negative samples by multiplying by 5 the number of negative edges, and using a Weighted random sampler to sample from the dataset. Therefore, the negative edges were more diverse but they were sample such that the ratio negative/positive was still 1. We also tried to augment the dataset by adding noise to the embeddings.
Unfortunately, none of those augmentation techniques yielded significative improvements to our best solutions.

# 8 Classification Neural network

## 8.1 Architecture

Once the features are concatenated, they are fed to a neural network to predict whether an edge is present or not. In our project, this network is a multi-layer perceptron with 3 intermediate layers and a classification layer. Each layer contains 4096 neurons.
Between each intermediate layers, we add a batch normalization, a ReLU activation function, and a Dropout with $p = 0.4$. Finally, after the classification layer, we use the sigmoid activation to get a score between 0 and 1. The network is quite simple but it is enough since the features we feed it are relatively easy to interpret.

### 8.2 Optimization

In our best model, we use the RAdam optimizer with a learning rate of $lr = 3e-4$ and $weight\_decay = 1e-7$. We observed a training a bit more stable with the RAdam optimizer over the Adam and AdamW optimizers. However, the difference in the loss across them was minor.

## 9 Ensembling

Finally, we have computed the mean between our solution combining SPECTER with neighborhood embeddings' means as well , and our model using Node2Vec which, when used on its own, had overfitted the training/validation set.

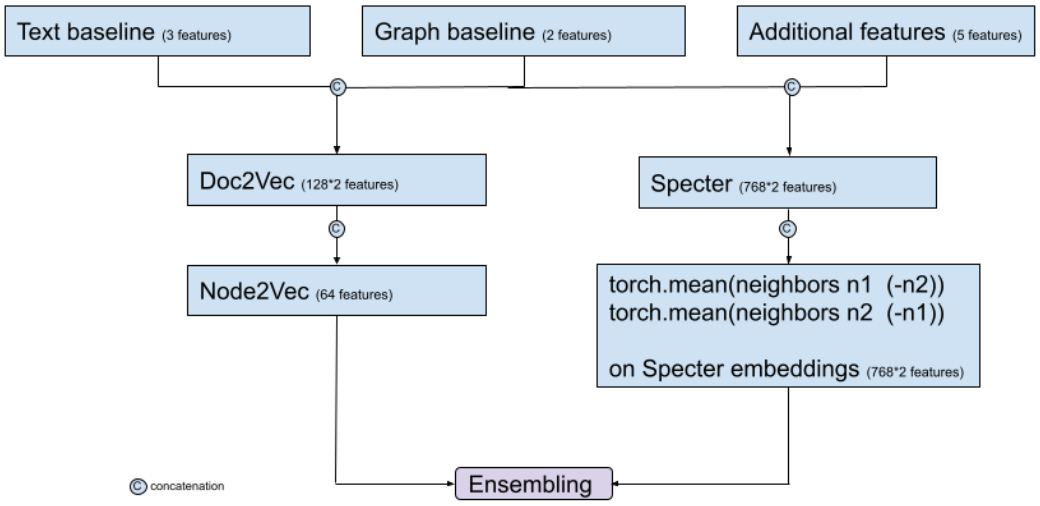


Figure 1: Final solution architecture

## 10 Results

| Features used | Score on 50% of Test Set |
|---|---|
| (1) Text Baseline | 0.55576 |
| (2) Graph Baseline | 0.48023 |
| (3) Text+Graph Baseline | 0.35967 |
| (4) Doc2Vec + Features Baselines Authors Graph | 0.13669 |
| (5) SPECTER + Features Baselines Authors Graph | 0.10254 |
| (6) Doc2Vec + Features Baselines Authors Graph + Node2Vec | 0.18010 |
| (7) SPECTER + Features Baselines Authors Graph + Node2Vec | 0.2187 |
| (8) SPECTER + Features Baselines Authors Graph + Neighborhood | 0.08833 |
| Ensemble:(6) and (8) | **0.08541** |

## 11 Conclusion

We tried to make use of a wide variety of features, with different embeddings encoding information about either the graph, the abstract, and even the authors of the papers used. The information regarding the abstracts was the easiest to improve, as each improvement on the training/validation set led directly to an improvement on the test set. On the other hand, the information coming from the graph was more tricky. Indeed, since the graph we had was composed of the training/validation edges, a better understanding of the graph structure could very easily lead to an overfit when trying to generalize on the test set. Finally, the information about the authors turned out to be much less useful than we hoped.
Overall, we managed to get very competitive results by using an ensemble on a model that understood the graph structure very well, but overfitted the train/validation set, and a model that had very rich information about the text embeddings, neighbors, and rough information about the graph structure.

# References


[1] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. SPECTER: Document-level Representation Learning using Citation-informed Transformers. In *ACL*, 2020.

[2] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

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

[4] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[5] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *CoRR*, abs/1802.09691, 2018.