CentraleSupélec

MLNS
Kaggle Challenge Report - Link Prediction

DAO Elisabeth - LAHRICHI Zineb - OUBAIK Ilyass - RIO Timothée

March 2022

Contents

1 Introduction

This Kaggle competition is a link prediction challenge. Here, we will try to predict is a research paper cites another one or not. To address this task, we are provided with a data set containing pair of nodes (papers) with a label about the existence of a link between the nodes (0 or 1). We also have access to some information about the papers (publication date, authors list, publication journal, title and abstract). The test set on which we are supposed to make the prediction is made of existing links (label: 1) that have been removed from the initial graph, as well as non-existing links (label: 0). The performance of this binary classification task is evaluated with the F1-score metric.

In the first section 2 we present the data pre-processing pipeline that we implemented. In the second section 3 we present the different features that we computed for the task and which are based on the three graphs that we created. In the second section 4 we present the models that we experimented, the influence and the the selection of the various parameters as well as the voting scheme between models that we implemented.

The code is available in the following github: *github.com/timrio/MLNS$_K$aggle$_C$hallenge*

2 Pre Processing

Before computing any features, we needed to pre-process the data sets.

2.1 Authors unique identification

At first we noticed that some authors were named differently across papers (Jean Dupont in paper A could be named J. Dupont in paper B). We had to deal with this issue otherwise it would have been almost impossible to create graphs based on authors. We found a package online called NameMatcher that scores the similarity between names based on the Levenshtein distance. We associated each author name in the dataset to a "representative name" (it is the longest name that designates the author in the data set) if the score given by NameMatcher was above a threshold and if the first letter of the two names were the same. After this step, we replaced each author name in the data set by its representative name and by doing so we considerably reduced the mutli-naming problem. Of course we did not check every names, so we can assume that some authors might still be designated by different names in the data set.

2.2 Titles and abstracts

Concerning the title and the abstracts we applied a regular NLP pre-processing. We convert everything to lower case, we removed the stop words and the punctuation and we lemmatized the text.

3 Feature Engineering

In this section we present the main features that we created for the task. We computed features for nodes and for edges. To do so we started by creating three graph to help us creating insightful features. All the features that we created can be found in A

3.1 Graphs Creation

For this task we wanted to take advantage of the graph structure of the data. We thus created 3 graphs: an article citation graph, an author co-authorship graph and an author citation graph. Our idea was to be able to re-create the environment in which authors work and in which articles are published.

3.1.1 Article citation graph

This first graph ($G_{articles}$) is the most straightforward because it is already included in the data set we are provided with. In this directed graph, the nodes are the papers and a link in present between article A and article B is article A cited article B. To compute the graph, we used the edges labeled 1 in our training set. This graph was created to help us gain more insight about the links between the articles and to help us compute powerful embeddings that represent the subject and this environment of the articles.

3.1.2 Authors co-authorship graph

We also created a graph to represent the collaboration between authors (G_{colab}). We also noticed that very frequently, authors that have worked together tend to cite each other in the future. We computed an undirected graph where nodes represents the authors and an edge is present between two authors if they have written an article together. The weight of a link is the number of collaboration between the authors.

3.1.3 Authors citation graph

Since authors are responsible to cite another author, we felt that it made sense to add another graph to represent the co-citation links between authors. We thus created a new directed graph (G_{author}) in which the nodes are once again the authors and an edge in present between author A and author B if author A has already cited author B. The weight of the edge is the number of time author A has cited author B.

3.2 Nodes Embedding

An efficient method to represent nodes similarities and predict a possible edge between them is to compute nodes embeddings and to use those embeddings to compute distance metrics such as cosine distance or a mere euclidean distance.

3.2.1 Abstracts and titles embeddings

The abstracts and the titles contains a lot of information about the content of the paper. To retrieve efficiently this information we used a sentence transformer to compute a title embedding and an abstract embedding for each paper. Since abstracts are relatively short, it made sense to use a sentence transformer on them. We found Specter [3], a pre-trained sentence transformer that had been trained "to map the titles abstracts of scientific publications to a vector space such that similar papers are close". We used this model to compute an embedding for the tile + abstract and for the title alone.

3.2.2 Articles embeddings

We then decided to compute an embedding for each article based on its position in $G_{articles}$. We started by using the notorious Node2Vec model [4], which implements a skip-gram approach on "sentences" made of random walk paths. After several experiments with the parameters, we decided to use the following parameters: a number of walks per node of 10, a walk length of 15, a dimension of 128 and a window size of 5.

We also computed Walklets embeddings [5]. They are relatively similar to Node2Vec embeddings but instead of sampling whole path, it skips over steps in each random walk and generates a corpus of node pairs which are reachable via paths of a fixed length. A skip-gram approach is then computed of this corpus to generate the embeddings. We use the following parameters: a walk number of 10, a walk length of 80, dimension 64 and a window size of 5.

3.2.3 Authors embeddings

For each individual authors we also computed embeddings based on Walklets and Node2Vec on the two available authors graphs (G_{author}, G_{colab}). We thus obtained 4 embeddings per author. Since most articles have been written by several authors, we needed a way to combine those embeddings. We experimented several possible combination (mean, max, sum, etc.). We did not observe any significant difference depending on the combination so we decided to use the mean. After this step we obtained 4 embeddings for each articles based on their authors.

3.3 Edges features

We started by computing simple features such as the time difference between the publications of the papers, the number of common words in the title and in the abstracts (post pre-processing), the number of common authors, if both articles were published in the same journal.

We then added metrics based on the previously computed embeddings. For each embeddings (abstracts, article and authors) we computed the cosine and euclidean distance between the embeddings.
Finally we also added some notorious and widespread features used in the context of link prediction such as Jaccard Coefficient, Preferential Attachement and Academic Adar Index.

3.4 Nodes Features

We added features for both nodes such as the Pagerank index and the degree of each nodes. With those features we wanted to give the model for understanding of the importance of each nodes.
In order to give for room for interpretation to the model, we also directly added a contracted version (obtained with a PCA and 5 components) of the Node2Vec embeddings of both articles.

4 Model Tuning and Comparison

In this section we present the different models that we have experimented as well as the fine tuning strategies that we implemented.

4.1 Cross validation strategy

Just after the pre-processing step and before any graph computation, we created 5 training/validation sets (with disjoints validation sets). The validation sets are made of 10% of the initial train set. We thus computed 15 graphs (3 on each train set).

4.2 Models

We fine tuned several models on each of the five training/validations sets we created after having scaled the data. Namely we experimented several classifiers such as Random Forest, SVM, XGBoost , CatBoost as well as simple Multi Layer Perceptron with a few layers only. Fine tuning one model was however extremely slow (around 10 hours for one model). For this reason we started by assessing the potential of each model (in term of performance and computational time). After a few experiments we realised that XgBoost and the MLP models outperformed the three others. So we decided to fine tune and use those 2 models only.

4.3 Hyper Parameters Tuning

We compared the performances of those models by monitoring the F1-score on the validation sets. We fine tuned the two models on each training/validation sets, and ended up with 10 fine tuned models. In order to accelerate the grid search we ran the XgBoost model of GPU. Even though some optimal parameters slightly differed between the sets (for instance the number of trees for XgBoost), we observed that most of the best parameters stayed the same on the 5 sets. The best performances with XgBoost were obtained with the following parameters: $Number\ of\ Trees$: *between*70 *and* 150, $Max\ Depth\ Candidates$: 15, $Learning\ Rate$: 0.05, $Minchildweights$: 5.
For the MLP model we achieved the best results with: $Activation$: $ReLU$, $LearningRate$: $adaptive$, $Optimiser$: $Adam$, $Number\ of\ hidden\ layers$: 2.

The average performances of the trained models on the validation sets are summarized in the following table (only MLP and XgBoost were properly fine tuned, the results for the other models come from some "hand made" experiments).

	MLP	*XgBoost*	*SVC*	*Random Forest*	*Cat Boost*
Average validation F1 score	*97.2*	*97.1*	*96.3*	*96.6*	*96.4*
Average training F1 score	*97.4*	*97.3*	*96.7*	*96.8*	*96.6*

We can notice that overall the overfitting of the models remains small and this was confirmed by the scores we obtained after submitting our predictions to Kaggle which were always very close to our validation scores.

4.4 Voting and final submission

As mentioned in the previous section, we obtained 10 fine tuned models. Before the final prediction, we added a step were we optimised for each model the threshold above which a prediction is considered positive. For each model we implemented a small gridsearch where we looked for the highest F1-score. We now had 10 fined tuned models and their respective thresholds for positive predictions.

We computed a prediction on the test set with each model (10 predictions) and implemented a simple majority voting between them to obtain the final prediction. This result achieved 97.047 F1-score on the public data set on Kaggle.

4.5 Additional Experiment

After this first submission we were surprised by our relatively "low" score. We then tried to retrain the models with various subsets of features to see if some of them could be deteriorating our performances. However using only a subset of our features was just detrimental to our results.

5 Conclusion

For this link prediction task, we leveraged the graph structure of the data as much as possible by creating two extra graphs. We computed a lot of different features, and embeddings sometimes using advanced models such as sentence transformers.

If we had more time, or more available computational resources we could have tried more parameters combinations for the various embeddings computation. We could also have retrained the sentence encoder on our dataset so that it perfectly fits our task.

A Features

For this task we computed 20 features (if we do not consider each dimension of the embeddings as a feature). They were all computed using the following metrics:

Academic Adar

The Adamic-Adar index (AA) [1] is defined as:

$$AA(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log|N(u)|} \tag{1}$$

Where $N(x)$ is the set of x's neighbors. AA "counts" the number of common neighbours between two nodes and penalizes the very popular neighbours.

Jaccard Coefficient

Jaccard coefficient (J) of nodes x and y is defined as:

$$J(x,y) = \frac{N(x) \cap N(y)}{N(x) \cup N(y)} \tag{2}$$

Where $N(x)$ is the set of x's neighbors. The coefficient computes the similarity between both sets of neighbors.

Preferential Attachement

The preferential attachment index (PA) is defined as:

$$PA(x,y) = |N(x)||N(y)| \tag{3}$$

Where $N(x)$ is the set of x's neighbors.

Resource allocation index

The Resource allocation index (RA) [6] is defined as:

$$RA(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{|N(u)|} \tag{4}$$

Where $N(x)$ is the set of x's neighbors.

Degree

The degree of a graph is the number of edges that are incident to the node.

Pagerank

PageRank [2] computes a ranking of the nodes in the graph based on the structure of the incoming links. The underlying idea is that a page is only as important as the pages that link to it.

Cosine between embeddings

The cosine distance between the embedding of node A (e_A) and embedding of node B (e_B) is defined as:

$$cosine(e_A, e_B) = \frac{e_A . e_B}{\|e_A\| \times \|e_B\|} \tag{5}$$

Euclidean distance between embeddings

The euclidean distance between the embedding of node A (e_A) and embedding of node B (e_B) is defined as:

$$euclidean(e_A, e_B) = \|e_A - e_B\|_2 \tag{6}$$

References

[1] Lada Adamic and Eytan Adar. Friends and neighbors on the web. Social Networks*, 25:211–230, 2001.*

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. Computer Networks*, 30:107–117, 1998.*

[3] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. SPECTER: document-level representation learning using citation-informed transformers. CoRR*, abs/2004.07180, 2020.*

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

[5] Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. CoRR*, abs/1605.02115, 2016.*

[6] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. The European Physical Journal B*, 71(4):623–630, Oct 2009.*