Link Prediction in a Citation Graph

Raphael Montaud & Gabriel Misrachi

***Abstract*—The object of this project is a directed graph where the nodes represent scientific articles and an edge indicates that its tail cites its head. Given that some of the edges of the graph have been removed, the purpose of our project is to be able to retrieve them. This can be seen as a classification problem which takes a pair of articles as input and outputs one if there should be an edge between them, zero otherwise. To solve this problem we will build a model based on metadata related to the articles, textual information from the titles and abstracts as well as graph related features.**

I. Introduction

The full code for the project is available here. For a quick overview of the structure of our code and for basic instructions on how to use it you can refer to the Readme in the code folder. Some functions require substantial computational power. For your information, in order to compute some results in a reasonable time, some computations were multi-processed on an Ubuntu 16.04 LTS with 16 Intel I7 cores.

II. The dataset

First of all we would like to say that the problem of link prediction is generally a complex issue. It can find many interesting applications notably in social networks or terrorists networks. The problem presented here is not a real problem since we know that approximately half of the pairs in the testing and training data are linked. So the testing set is actually composed of linked pairs and pairs chosen at random. So this is a rather easy problem compared to what you might come across in the real world. It would be like taking two brothers and a random person on earth and asking to someone ”who are the two brothers ?”. It is quite easy to reach a great accuracy since most of the examples will be very easy. But sometimes two brothers can look nothing alike and sometimes two persons chosen at random can look alike. Those are the examples that are going to be challenging.

III. Feature Engineering

A. Metadata

Date Difference: Of course this piece of metadata is important. No one can cite an article that has not been released yet. Indeed after exploring the dataset, we find out that among pairs of articles that have a negative date difference, only $\frac{1}{100}$ cite each others.

Common Authors: This feature contains substantial insight since there are chances that articles with common authors cite each other (authors usually write on related subjects and of course they tend to cite their previous articles). It can be looked at as a sufficient condition, but not a necessary one. Indeed, we found out that among pairs of articles with one or more common authors, there is a link in 98% of cases. But among articles that cite each other, only 10% have common authors.

Journal Similarity: The journal names are given with a strange format that appears to be nested. For example we can see ”Phys.Quant” and ”Phys.Elec”. We decided to parse this recursively and attribute 1 point if the first layer matches, 2 points if the first two layers match, etc. If the first layer does not match, we count zero. If one of the journals is unknown, we count -1. Here are some examples:

- ”Phys.Quant.Rel” and ”Phys.Quant”: 2 points
- ”Phys.Quant” and ”Phys.Elec”: 1 point
- ”Phys.Quant” and ”Fin.Quant”: 0 point

Partial Conclusion: In the following tab we present the results obtained by fitting a Random Forest model with the textual features (we consider the average test score on a 5K Fold). It should be noted that the baseline result is 70%.

TABLE I
Comparison of Metadata Features.

Feature	Date	Authors	Journal	All Metadata F.
Test score in %	80.5	70.5	70.5	80.9

We can already say that the most useful metadata is the date. The author features do not seem to be very relevant for our problem. We manage to have a satisfying score only by considering the date and there is no machine learning needed for this. Indeed, to double check our results we hard-coded a classifier that is always predicting that there is a link, except if the date difference is negative. This classifier obtained a f1 score of 80%. As we said in I, this problem is not difficult and we should not be surprised to have such a good score with just a few features.

B. Textual Features

Overlapping words in Title: This feature is interesting since it is going to give an approximation of the content-wise closeness of two papers. Of course, it is one of the most simple textual feature one can imagine.

Overlapping words in Abstract: This is already more interesting since two articles that cite each other might have a lot of similar words in their abstracts. When computing this feature, we stemmed the words in order to count as equal ”logarithm” and ”logarithmic”, for example.

Cosine Distance: The problem with the number of overlapping words is that it takes into account common words although they probably aren’t relevant. This is why we built a TF-IDF representation of the abstracts. Given this representation, we then want to approximate the similarity of two texts and according to [5], it can be done with the cosine distance. The formula to compute it is:

$$\frac{d.q}{||d|| * ||q||}$$

where d and q are the "bag of words" representation of the abstracts.

Score (1, 2): According to [5] one of the best function for query search in a set of documents is the following function:

$$score(d,q) = \sum_{t \in d} idf_t * \frac{tf_{t,d} * (k+1)}{tf_{t,d} + k_1 * (1 - b + b * \frac{len_b}{avglen})}$$

Since this works well for queries, we assumed it w could also be interesting for document similarity. This score gives information on the closeness of two texts. Since this feature is not symmetric in d and q, we decided to compute also Score(q, d) in order to be able to tell which one is the most relevant.

Partial conclusion: In II we present the results obtained by fitting a Random Forest model on the textual features (one by one).

TABLE II
COMPARISON OF TEXTUAL FEATURES.

Feature	Title	Abstract	Cosine	Sc(1, 2)	All Textual F.
Test score in %	61.2	76.3	81.1	78.7	81.35

First of all there might be an issue with the feature "overlapping words in Title" because we are getting even worse results than the baseline classifier. Another conclusion is that score(1,2) and score(2,1) have the exact same results. To conclude, the cosine distance is probably the most interesting feature among the textual features, we'll definitely keep an eye on it.

C. Graph Related Features

Since we are doing this challenge for a course on Natural Language Processing, we first focused on the textual features. But we discovered afterwards that graph related features are even more important in this challenge. Most of our features are based on papers [1] and [3]. [1] is an article on co-authorship prediction, which is actually very close to our subject. They discovered that the shortest path is a very significant feature. They also implemented the sum of neighbours and the sum of secondary neighbours which gave us some ideas for our own features.

1) Bidirectional Graph: At first, we consider the graph as bidirectional and compute features according to that assumption.

Common neighbours: The number of common neighbours gives insights on the closeness of two papers and the links that might exist between them. For example, if A cites B and B cites C then A is likely to at least know about the article C and might cite it.

Preferential Attachment: The preferential attachment is the product of the number of neighbours of each node. Indeed, if we make a reference to the social networks like Facebook, people with a lot of friends are likely to create more connections.

Adamic/Adar: The Adar coefficient is computed as follows:

$$\sum_{z \in N(a,b)} \frac{1}{log(|N(z)|)}$$

where $N(z)$ are the neighbours of z and $N(a, b)$ are the common neighbours of A and B. Again, let's take a social network point of view. The fact that two people are both connected to a popular person is not very relevant. But the fact that two persons are both connected to a person that has less than then 10 Facebook friends is very relevant. This can reveal a strong bond between two persons. The Adar coefficient takes this into account by counting the number of common neighbours and normalizing by the number of neighbours of each neighbour.

With the same idea, we also computed the same feature, but without the logarithmic smoothing. It is called the "Resource Allocation Index".

Jaccard: The Jacard coefficient can be useful because it takes into account the number of common neighbours but it is normalized by the number of connections that the two nodes make.

$$jaccard(a,b) = \frac{|N(a) \cap N(b)|}{|N(a) \cup N(b)|}$$

2) Directed Graph: Now we consider that the graph is directed to compute new features.

Shortest Path: The shortest path in a directed graph is obviously an interesting feature. For example if A cited B and B cited C then A is likely to at least know about the article C and he might cite it. In [1] it is one of the best features to solve a problem of co-authorship prediction.

Katz: The katz coefficient is interesting because, like the shortest path, it can give a lot of insights on the closeness of two nodes, but it is more stable. Indeed, it computes the k shortest paths and weighs them in a sum according to the size of the path. The beta coefficient enables to control the influence of the length of the path.

$$katz(a,b) = \sum_{p\ in\ k\ shortest\ paths} beta^{|p|}$$

Out neighbours: Since we want to compute the likelihood that node 1 cites node 2, the number of articles cited by node 1 can have an impact. Indeed, if the article 1 cites none, it is unlikely that it cites article 2.

In neighbours: This is a very interesting feature. Indeed, some articles are more popular than others. If an article is very unpopular (nobody cites it), it is very unlikely that it will be cited.

Popularity: We had the idea of this feature after considering the "in neighbours" feature. Since we were talking about popularity, we wanted to consider the fact that being friends with someone popular usually means that you are popular. This is why we computed the sum of the count of in-neighbours of the in-neighbours.

Common successors: We look at all nodes that were cited both by paper 1 and paper 2. This can give insights on the closeness of two articles.

Common predecessors: We compute the number of nodes that cite both articles. This should give insights on the closeness of two articles.

3) Author's Directed Graph: For now we only considered the graph of scientific paper's citations, but from this data, we can create the graph of authors citations. We first create a node for each author in the dataset. Then, for each example of citation, we create a link with weight 1 between all authors of paper 1 and all authors of paper 2. If the link already exists, we increment the weight of the link. So at the end, if there is a directed link of weight 2 between author A and author B, this means that author A wrote two papers that cited papers of author B. Now, in order to compute features for an example (paper 1, paper 2) we had to consider all edges between each possible pairs of authors. We aggregated the weights computed with three different functions: sum, mean and best. (mean is the sum divided by the product of the number of authors). This is pretty much what [1] did in a problem of link prediction in a co-authorship graph.

In neighbors: We computed the in neighbors for the authors of paper 2. This feature can represent the popularity of authors so this can give good insights on the likelihood of a link.

Previous citations: We computed the sum of all edges between authors of paper 1 and authors of paper 2. This can give good insights on the previous citations, and an author is more likely to cite another when he has already cited him.

4) Co-authorship Graph: With the data provided, we can also generate the graph of co-authorships. Just like for the author's directed graph, we computed weighted edges that represent how many times two authors co-wrote an article. With this graph, we computed only one feature which is a co-authorship score between the set of authors of paper 1 and the set of authors of paper 2. Intuition tells us that an author is more likely to cite another if he has already collaborated with him, so this feature should be interesting.

5) Partial conclusion: In the following tab we present the results obtained by fitting a Random Forest model with the graph related features (with a 5 fold cross validation).

TABLE III
COMPARISON OF GRAPH RELATED FEATURES.

feature name	train score	test score
normalized authors citation	75.39	75.39
jaccard	95.92	95.92
adar	96.17	96.16
preferential attachment	78.57	78.55
resource allocation index	96.27	96.27
common neighbors	96.27	96.26
out neighbors	75.53	75.53
in neighbors	79.24	79.20
popularity	77.98	77.95
shortest path	87.74	87.74
common successors	75.53	75.53
common predecessors	79.24	79.20
paths of length one	77.96	77.93
coauthor score	70.50	70.50
authors in neighbors	70.81	70.81
all together	97.29	96.95

The first conclusion that we can make is that graph related features are the most important features, indeed the results are way better than with textual features. It looks like the most important features are the ones related to common neighbours. Indeed, the resource allocation is also a feature related to common neighbours. We were quite surprised by the results of the shortest path feature since it is usually one of the most important features in link prediction problems ([1]). Actually, as we said in I, the problem is usually easy but what we need is insight on the nodes when the problem is hard. A hard problem means that the two nodes are close and usually they will have at least one common neighbour, so a shortest path of 2. But the shortest path cannot distinguish between two very close nodes (with a lot of common neighbours) or two relatively close nodes. We think that this is why the common neighbours features are getting better results than the shortest path, it is giving more information for the tough predictions.

IV. Feature Selection

For this part we decided to use a greedy forward selection process as described in [4]. The model we used for this section is a Random Forest Classifier. IV presents the results of this feature selection with the test score of the model taking into account the accumulated features (5 fold).

TABLE IV
TEST SCORES DURING THE FEATURE SELECTION PROCESS

feature number	feature name	test score
1	common neighbors	96.27
2	cosine distance	96.76
3	date diff	97.22
4	authors citation	97.39
5	shortest path	97.50
6	preferential attachment	97.57
7	in neighbors	97.64
8	common author	97.67

What is interesting is how features tend to complete each others. The first feature selected is the one that gives the best results alone: common neighbors. But although some features have very good results in the graph related features, the second feature chosen is a textual feature and the third one is the date difference which is from metadata. So the features that bring the most improvement to the model are the ones that give different information. If we look at the two next features, once again they are chosen from different sets of features (authors citation comes from the author citations graph and shortest path is from the directed graph of citations). Even though it is very useful for computational reasons, the outcome of this process can't be considered as the absolute best features for our problem but rather as a suboptimal solution. Indeed, imagine you were to model a classification process using a logistic regression. There could very well be two features that are alone useless but completely explain the target when you combine them. The afore described process might not detect such a behaviour and therefore yield unsatisfactory results.

In the following, we will check the validity of this feature selection process and use its advantages to perform wider and finer parameter tuning.

V. Model comparison and parameter Tuning

As an introduction to this section, let us first take care of validating the feature selection process we described in the preceding section. To convince ourselves we could safely use the resulting subset of features we compared the results of a fast learner, Random Forest, on said subset against that of the same learner on the whole set of features. As adding more features did not increase our score in a significant and stable manner we deemed it was fine to work exclusively on the significant subset for the aforementioned learner. We extended this validity hypothesis to the SVM because we absolutely needed to reduce the training time of that costly model.

N.B. For reproducibility reasons, as it is very long to compute, we did not use the shortest path feature in the modeling phase.

In the following, we describe all our models as well as the tuning of their hyperparameters. For scoring purposes, we systematically used a five fold stratified Kfold through its scikit-learn implementation. We do not mention the exact scores of each model as they are not very interesting compared to the baseline we had with our greedy forward feature selection process. Rather than that, we decided to describe the build up towards our best submission that was yielded by combining all of them.

A. SVM

On the one hand, SVMs are very good at separating non-linearly separable samples so they're definitely something we would want to look at. On the other hand, they are very costly to train and suffer a lot from the curse of dimensionality. Our feature selection is going to help with that but it remains a greedy approach compared to others. Thus, to further reduce the dimension of our problem we decided to create a scikit-learn Pipeline coupling a selection algorithm with our classifier. We then used the GridSearchCV algorithm from the same library to simultaneously choose between two feature selection algorithms (PCA and Kbest on entropy), optimize their parameters (the number of selected features) and the hyperparameters of the SVM (kernel and regularization factor inverse C). The results of this search suggested us to use all the pre-selected features and input them in a linear SVM with $C = 0.1$. For the sake of exploration, we also considered using Bayesian optimization to determine an optimal regularization factor but the method is not tractable given the size and dimension of our dataset. We did not investigate further as we didn't feel that SVMs would be the key to our problem.

B. Random Forest

From the beginning, our best bet was placed on tree methods. With features like date difference that allow us to derive obvious decision rules we already knew that tree methods would perform well. Our first approach in that area was to use a Random Forest. Even though we finally used the preselected feature set we also looked at other configurations to convince ourselves that we were right to do so. As this model is not very expensive to train we were able to do multiple grid searches while each time refining our parameter grid. That allowed us to get precise results as to what values we should set for the hyperparameters. We operated this way for the most important parameters, that is : max_depth, min_samples_leaf and n_estimators. Theses parameters are crucial when trying to best fit a random forest as they are the best way to control overfitting. The results of our search led us to consider a forest with : max_depth = 20, min_samples_leaf = 10, n_estimators = 150. To further evaluate the pertinence of the features we handcrafted we also looked at feature importances within the random forest fit. The results are drawn in the following figure.

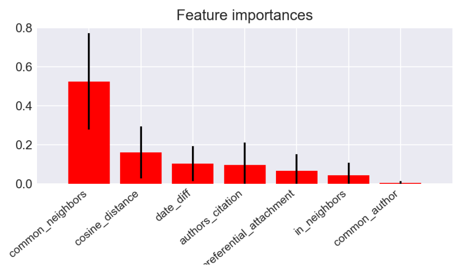

Fig. 1. Feature importances in RF

The error bars represent the std of each feature importance through the 150 trees in the forest. The results completely back our greedy forward feature selection. Results are also compelling as they clearly give a decisive importance to the common neighbours feature.

C. Neural Networks

To achieve our modeling effort we tried two neural network architectures. We used the scikit-learn wrapper on the Keras library in order to be able to tune them using scikit-learn's optimized grid search. We did not use them directly for submission but they were part of our final effort described in the following and last section.

N.B. As they were last minute efforts, we did not bother to verify the validity of our feature selection on our Neural Networks and used the whole set instead (greedy features aside).

D. LightGBM

As as substitute to Random Forest we decided to try out a LightGBM classifier. It is a very popular ensemble boosting method that grows trees leaf wise instead of depth wise. It is widely used because it is efficient and very fast to train even when dimension is high.

N.B. As this approach is very fast and does it's own version of feature selection we directly used it on the entire feature set.

Once again, we tuned the hyperparameters using a cross validated grid search. We tuned the most important parameters and set the others to common values for the task at hand. To further illustrate our approach we drew the metrics computed during the tuning process in the following figure.

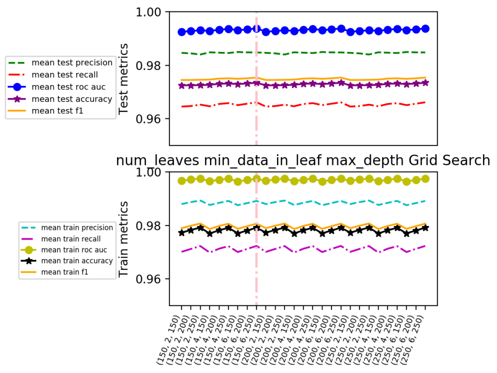

Fig. 2. Grid Search metrics per fold for LightGBM

We here witness how robust the LGBM classifier is through the parameters set. This shows that we have found the good features and that no matter what parameters we try out LightGBM yields approximately the same results.

VI. Stacking

In order to combine all our fine tuned models we decided to stack their results. What that means is that we took the mean predictions on the test set over 5 folds of cross validation training and used those as features. More precisely, we had five features each corresponding to the mean prediction of one of our models (linear SVM, Random Forest, LightGBM, Neural Network 1, Neural Network 2). On top of those we fitted a Pipeline of feature selection techniques (Kbest with χ_2 statistic) and classifiers (Logistic Regression and Random Forest). For tuning purposes, we used a grid search on the pipeline instance. It is convenient to tune the models and the stack separately but there is a bit of data leakage when doing so. Indeed, while performing the grid search on the stack model we use features which were computed with parameters chosen on folds that, in total, covered the entire training set. Thus, the data leakage. The right way to train the stack would be to tune all the hyperparameters at once. Still, given that the leakage is tenuous and the tuning of the whole architecture quite costly, we decided to proceed anyway. We were rewarded as the stack yielded our best submission for the private leaderboard: 0.97707. Sadly, we mistakenly didn't select that submission for the final scoring.

VII. To conclude

As a conclusion, we felt throughout our study that the crucial step to nail this problem was feature engineering. Notably, it appears that for Link prediction problems, the graph related features are extremely important, and what is surprising is that they give much more insights than the information extracted from the abstracts. Indeed, we were able to reach a performance very close to our best with as much as a single one of those features. We managed to improve this performance by adding features that were adding complementary information and by thinking outside the box and computing original features like those from the author's graph. Still, to be able to distinguish ourselves on the leaderboard we decided to put some great effort into the modeling phase. Even though it did not perform a lot better than the basic model with the right features, our final stacking effort was decisive in securing the third place on this competition and was therefore quite justified.

References

[1] M. Al Hasan, V. Chaoji, S. Salem, M. Zaki "Link Prediction using Supervised Learning" *Rensselaer Polytechnic Institute, Troy, New York* 12180

[2] http://be.amazd.com/link-prediction/

[3] M. Al Hasan, M. J. Zaki "Link Prediction in Social Networks" *eBay Reseach Center, San Jose CA and Rensselaer Polytechnic Institute, Troy, New York* 12180

[4] Kdeng "Feature Selection" *Carnegie Mellon University of Computer Science* https://www.cs.cmu.edu/ kdeng/thesis/feature.pdf

[5] M. Vazirgiannis INF 582: Text Mining and Natural Language Processing *Ecole Polytechnique*