ML Project : Citation Network Prediction

Kaggle : BarthDoe

Made by:
Mathieu Rita - mathieu.rita@polytechnique.edu
Barthélémy Meynard (barthDoe) - barthelemy.meynard@polytechnique.edu
Jacques Boitreaud - jacques.boitreaud@polytechnique.edu

Contents

1 Introduction **2**

2 Feature engineering **2**

2.1 Features 2

2.2 Features selection 4

3 Comparing various approaches **5**

4 Achieving our best score: Random Forest tuning and training-set balancing **7**

4.1 Random forest fine-tuning 7

4.2 Refining the training set 7

1 Introduction

In this project we tried to predict links between articles in a citation network. The citation network is a directed graph G(V, E), where nodes represent scientific papers and a link between nodes u and v denotes that one of the papers cites the other. Each node in the graph has attributes such as the paper's abstract, the year of publication, the authors' names, the journal's name and the paper's title. Using these attributes and the set of citation links we were provided with, we designed meaningful features to quantify the similarity of two papers and their proximity in scientific research. These features we designed are of three types: topological, textual and directly related to the papers' attributes. We then explored several machine learning and deep learning approaches to predict whether a paper cited another or not. In the first part, we will present the data preprocessing and features design. We will then compare the different approaches we tried. Finally, we will focus on the optimization of a random forest classifier, which outperformed all other approaches.

2 Feature engineering

2.1 Features

In this section, we present how we selected the features for this problem. The underlying issue is : what is the mechanism behind citations ? A lot of the features we chose are inspired by [1] since this article presents clever ways to benefit the measure the similarity of papers. For the following section, we call the first element of the pair in the dataset the *from paper* and the second element of the pair the *to paper*.

Features provided by the baseline: Since the features given in the baseline made sense and gave some results, we decided to keep them:

1. **Number of overlapping words in the titles**: To compare two articles, we calculate the number of overlapping words in the *from* and *to* titles.
2. **The date difference**: This feature is just the difference between the date of publication :

$$dif_{date}(from, to) = publication\_date(from) - publication\_date(to) \tag{1}$$

   We especially thought that a searcher would rather cite recent works.

3. **Number of common authors**: numbers of authors who have taken part in the writing of both the *from* article and the *to* article.

Intuive features: We add to the baseline some features that rely on the basic description of the different articles: authors, journal, date, number of citations :

4. **Is self citation**: return 1 if the there is at least one common author, 0 otherwise

5. **Is same Journal**: Author publishing in the same journal may know each other better and work in the same field.

6. **Number of common words in the journal**: It can brings some links between the articles in term of research fields : we hope it will bring some marginal effects.

7. **Earliness : Number of citations of the "to" article weighted by the date difference**: According to [2], a factor that helps to determine the probability of citation is how immediately a paper gets cited in its early years: if a paper gains an early boost, its visibility makes it more likely to pick up more citations. Thus, we decide to build a feature that sums the number of citations weighted by the date difference :

$$score(to) = \sum_{paper \in in-links} w_c(paper) = \sum_{paper \in in-links} \frac{1}{1 + (date_{to} - date_{paper})} \tag{2}$$

8. **The number of in-Links of *to* paper**: this is the articles which refer to *to*. It is supposed to represents the academic influence of an article.

9. **Author connections**: Because Scientific fields are growing at an incredible rate, searchers can not read everything that is published and therefore they might be more familiar with the work of some others searcher. This feature is the ratio between the number of article where an author from *from* cites an author from *to* with the total number of articles written by the authors of *from*.

Topological features:

The structure of the citation network spurs us to use a graph architecture. All the features coming from this graph are called topological features.

10. **Number of common neighbours**: counts the number of article who are connected to *to* and to *from*

11. **Similarity features**: The idea is to evaluate the connection between two articles. We chose two different similarity measures that are used in graph analysis. (notation : Γ represents the connected nodes)

    2.1 Link-based Jaccard coefficient; it represents the the common neighbours of *from* and *to* over their total numbers of neighbours.

    2.2 The Adamic-Adar Index

$$Sim^{AA}_{from,\,to} = \sum_{z \in \Gamma(to) \cup \Gamma(from)} \frac{1}{\log k_{out,z}}$$

12. **Difference in Pagerank** : pagerank is an algorithm made by Google in order to rank website through graph analysis. We admit that this ranking can be transposed to our problem.

13. **Betweenness centrality difference**: the betweenness centrality is a classical feature in graph analysis. It aims at saying if a node is at a border of a graph or at its center.

14. **Is in the same cluster**: The graph structure of our data enables us to make clusters of article which cite each other a lot. This a way to gather articles about the same subject. This feature return 1 if the two articles belong to the same cluster, 0 otherwise.

15. **Shortest-path** It counts the shortest number of edges needed to go from *from* to *to*.

Textual features:

We had a lot of textual information to use : title, abstract, authors, journal. If two papers are about the same topic they are likely to use the same word. Thus, we built similarity measures over this textual informations : we perform the classical method (preprocessing and applying the Term Frequency–Inverse Document Frequency ; or count the number of common word).

16. **number of common words in the title**

17. **number of common words in the abstract**

18. **cosine similarity on the titles**: Once the word have been detected and the TF-IDF computed we can measure the similarity in the keywords of titles through cosine similarity.

19. **number of common words in the abstract**: Count the number of common word in the abstracts after the preprocessing (the formula can be seen in [1]).

20. **cosine similarity on the abstracts**: the same can be done on the abstracts.

2.2 Features selection

The features selection is a key issue to obtain the best result. There are some algorithms that help to select the features. Here, because the number of features is not too big, we decided to perform a handmaid selection :

Results

First, we compute scores that tell us what are the most important features (SVM, LR and RF weights). Here is what we obtained :

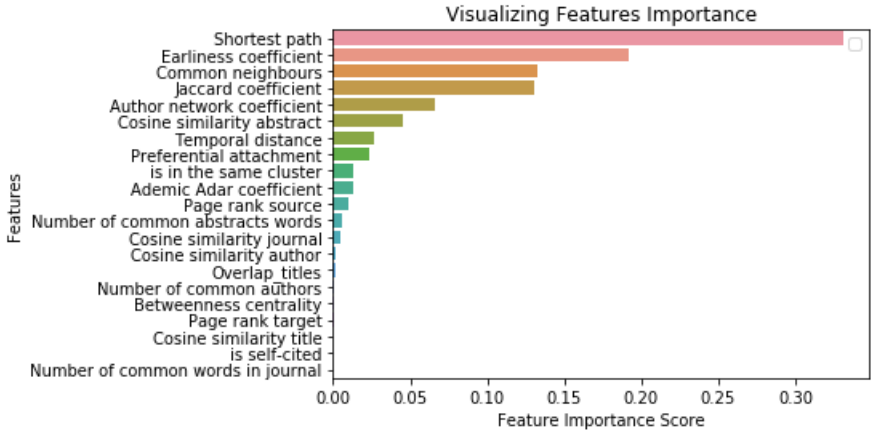

Figure 1. Importance feature scores obtained with a random forest

Then, we compare the results with or without each of the most important features. It appears that the "shortest path" and the "author network coefficient" enhance considerably the results for the training set but not for the testing set : they overfitted the training set because we built the graph with edges that are not supposed to be known for the prediction.

We continue to test combinations of features by removing one by one the other features until we obtained the best selection. At the end, "is in the same cluster", "betweenness centrality" and "is self-cited" were remove from our set of features.

After this step, we got our final set of features on which we trained our models. We can notice that the coefficient we built after reading [2] became the most meaninful features of the problem. Moreover, the features which derive from the graph are important for the predictions.

3 Comparing various approaches

Although the literature substantiated our finding that SVM were the best performing model for this problem, we also tried out tree-based approaches and a multi-layer perceptron. We made sure the hyperparameters were adapted to the dataset by performing a fast grid search, and compared the mean cross-validation score of these models to have a rough idea of their level of performance. To save time, training was done on a random selection of 10% of the training set.

Support Vector Machine We first trained SVM since the literature we used to compute our first set of features [1] suggested it outperformed other approaches. We achieved best results with a radial-basis function kernel and grid searched for the C penalty (powers of 10 between -2 and 2). We achieved our best score with a C penalty of 1.0, and obtained a 0.962 score on the public leaderboard with this model. We were unable to understand why our models always scored a bit worse on the test set than they did using cross-validation on the training set. Eventually, rebalancing the training set mitigated this issue as shown in part 3.

Table 1: Indicative scores for the classifier selection

Classifier	**Mean three-fold cross-validation f-score**
Random Forest	0,992
SVM	0,983
XGBoost	0,972
Multi-layer perceptron	0,971
Logistic Regression (L2 regularized)	0,968
AdaBoost Decision Trees	0,966

Random Forest Classifier Without tuning parameters, setting the number of trees to 200 and no maximum depth, we achieved a great score with this model. This triggered our curiosity and we explored the influence of the number of trees and tree depth. We found out that increasing the number of trees did not increase the F-score by more than 1e-3. However, setting a maximum tree depth of 5 significantly cut the training time but also affected the performance, as it alters the discriminative power of each tree. With a max tree depth of 10, we were able to obtain the same mean f-score as with no max depth. Like with AdaBoost, it shows that 10-levels deep trees are complex enough for this dataset.
In the next part, we will delve into the optimization of the model's parameters and training set refining which enabled us to obtain our best score on the public leaderboard.

AdaBoost with Decision Trees as weak learners Since the Random Forest Classifier performed quite well, we were interested in trying out the AdaBoost approach, as we had seen in the labs it could improve tree-based approaches. We took inspiration from the lab to grid search the optimal tree depth and number of learners, using three-fold cross-validation to validate our choice. An interesting finding is that contrary to the random forest, setting a maximum tree depth of 8 does not affect the f1-score: It increases by only 3E-4 if we set the maximum tree depth to 15. This finding is pretty much in line with what we found during the AdaBoost lab, where the optimal tree depth was also lower than 10 s.

Extreme Gradient Boosting We decided to try this approach as it is known for achieving good performance in Kaggle challenges. We grid searched parameters affecting model complexity: the max tree depth, minimum child weight, number of trees and the gamma factor (decrease in loss required to form a new split). We obtained our best results with a gamma factor of 0.2, a minimum child weight of 3 and a maximum tree depth of 12. We did not try more than 500 trees as the literature suggested there was little benefit in doing so for a high increase in training time, but the 500 trees model outperformed those with 100 and 200 trees.

Logistic Regression with Stochastic Gradient Descent . We implemented logistic regression with stochastic gradient descent over iterations. We grid-searched for the optimal learning rate, using again three-fold cross validation. We obtained our best score with a learning rate of 1e-3. We also used one of our Kaggle submission to test this model, which was trained with the first, less optimized features we designed, and obtained a 0.955 score. Although this shows logistic regression could perform quite well on this data, we found out that the training time was significantly higher than that of tree-

based models, and the convergence was only achieved after a high number of iterations (more than 5000).

Multi-layer perceptron We tried multilayer perceptrons and obtained satisfying results with these, but found them harder to optimize than other models. We performed a fast grid search on the number of layers and the number of neurons per layer, without being able to achieve as good results as the ones obtained with previous approaches.

4 Achieving our best score: Random Forest tuning and training-set balancing

4.1 Random forest fine-tuning

As the previous comparison suggested, we decided to further optimize the random forest. We grid searched more model parameters which control the model's complexity:

1. The minimum number of samples per leaf: Setting higher values reduces the complexity of the model by controling the number of leafs, and thus prevents overfitting. In our case, setting higher values than the default one (1) resulted in a weaker performance, probably because the complexity and size of the dataset makes it better adapted to complex models.

2. The minimum samples for a split : It has the same effect as it controls the number of internal nodes

3. The number of features considered when looking for the best split: The smaller, the less likely to overfit. For practical reasons, we only tried log2(nb of features) and sqrt(nb of features) which seemed the most popular in the literature. The log option was best in all cases.

After optimizing these parameters, we compared the performances of our models under two decision criteria for choosing the best split: the gini criterion and the entropy one. The difference in the scores obtained were minor, but the entropy criterion worked better on the test set. We invesigated the literature to understand the difference between both criteria. We found out that there was usually little difference between both, but gini criterion enabled to cut computation time because it doesn't need to compute log functions. We weren't able to notice a significant difference in computation time.

4.2 Refining the training set

The last optimization we perform was a refining of the training set. During our tests, we evaluate some statistical figures : true positive, false positive and false negative (the proportions were round tp=6500 ; fp=100 ; fn=200). Therefore, we decided to rebalanced the dataset in order to modify these proportions (increasing tp while keeping the same proportion of fp+fn). Empirically, it appears that removing between 0% and 20% of the 0-labeled elements improve slightly our F1-score. It helped us to turn from 0.97304 to 0.97506 on the private leaderboard.

References

[1] Naoki Shibata, Yuya Kajikawa, Ichiro Sakata, *Link prediction in citation networks.* Journal of the american society for informaiton science and technology

[2] R. Van Noorden, *Formula predicts research papers' future citations.* Nature News