Assignment 2: Predict missing links in citation networks Kaggle team: *Les Cloue-Porte*

Adib Baziz, Reuben Dorent, Samuel Joutard, Joël Seytre

{adib.baziz, reuben.dorent, samuel.joutard, joel.seytre} @ student.ecp.fr

This report describes our work related to the following Kaggle challenge: `https://www.kaggle.com/c/ngsa-2018/`. Our results are entirely reproducible from the attached code folder (see github `https://github.com/joelseytre/ngsa/`, which contains a `ReadMe` file that explains how to do that.
Our final F-score was **0.97668** on the public test set and we are currently ranked **2 / 19**.

Note: every time we talk about a "score" in this report, we refer to the F-score which is the error measure used for the Kaggle competition.

Note: the group who is currently ahead of us in the competition by 0.002 submitted 41 times so we really hope they overfitted the public test set strongly...

1 Feature engineering

We put in place the following features:

(i) The three default features provided with the baseline, which capture:
   (a) **Title overlap** (number of overlapping words in titles)
   (b) **Year difference** between articles
   (c) **Number of common authors**

   These features capture tendencies of articles to cite articles tackling the very same issues, if they are in the past, and the fact that people like citing themselves ("gotta work on that Google Scholar h-index").

(ii) The **TF-IDF** (Term Frequency - Inverse Document Frequency) between the source and target articles:
   (a) **TF-IDF distance between the articles' abstracts**
   (b) **TF-IDF distance between the articles' titles**
   (c) **TF-IDF distance between the articles' author names**

TF-IDF is a way of computing a similarity measure between textual information[1]. An interesting alternative would have been to compute the Word2Vec[2] distances but we saw that as redundant.
These features measure how similar the abstracts and titles are, assuming that similar topics and papers will tend to cite each other.
The measure of textual similarity between author names is a very pointless feature that we expect to have no impact on performance. It was very simple to add to our feature pool so we thought "huh, why not?".

(iii) The **number of times the target article is cited**
We computed this feature based off the intuition that some articles are much more cited than others, and that if an article is cited many times it is more likely that it will be cited again.

(iv) The **shortest path between the source and the target articles**

   (a) in the **directed** graph
   (b) in the **undirected** copy of the graph

We computed the shortest path (removing temporarily an existing direct link if needed for the training set) to measure how close the two articles are in the citation graph.
Measuring the shortest path on the **directed** graph captures how the source article might cite articles that cite articles... that cite the target article.
Measuring the shortest path on the **undirected** graph is another way of measuring how close the articles are by enabling to "go up" links and measure if the articles are in similar parts of the network.

(v) The **jaccard similarity coefficients** of the source and target articles
The Jaccard similarity[3] coefficient of two vertices is the number of their common neighbors divided by the number of vertices that are adjacent to at least one of them. This is another way of measuring how comparable the articles are within the graph.

2 Model tuning and comparison

First, we computed an overall training set of 10% of the total links available in the graph (5% was the number used in the baseline). We then divided that training set into 90% pure training set (i.e. $\sim 55,000$ links 9% of original link dataset) to train our models and 10% validation set (i.e. $\sim 6,000$ links or 1% of original link dataset) to test for overfitting and for the comparison of our models.
One important thing to note is that we normalized the datasets in unison, which means we substracted the mean of the training set and divided by the standard deviation of the training set for all the datasets. This means the model will operate similarly on the test set even though the test set's means and variance will not be exactly 0 (respectively 1).

Model comparison In order to obtain the best performance we compared the following methods, which were fine-tuned in the following ways:

(a) **Gradient Boosting**
We used a Gradient Boosting Regressor from the `lightgbm` library. At first the model

[1]Sparck Jones, K. (1972), "A statistical interpretation of term specificity and its pplication in retrieval", *Journal of Documentation*, Vol. 28, pp. 11–21.
[2]Mikolov et al., "Efficient Estimation of Word Representations in Vector Space"
[3]Huang, "Similarity Measures for Text Document Clustering"

showed strong overfitting, which we eventually reduced by using L2-regularization (grid search for hyperparameter tuning), and early-stopping. The number of estimators used is very high (10,000), but the training is stopped as soon as the validation score doesn't improve for 50 iterations (final results used ~ 130 estimators).

(b) **Support Vector Machine** (SVM)
We used `Scikit-Learn` SVM implementation with the standard hinge loss as suggested by the baseline. The best score on the validation set obtained for this method is **0.968**, which is already a correct score when comparing to more complex methods.

(c) **Random Forest**
Here again we used the `Scikit-Learn` implementation of the Random Forest algorithm. We tried to change the maximum depth of the trees built in the model as well as the number of trees generated, this method remains very robust and still achieves a rather high score of **0.975** without having to fine tune the parameters any further.

(d) **Logistic Regression** We used Regularized Logistic Regression from the `Scikit-Learn` with both L1 and L2 regularization. Both types of regularization seem to perform equally well, reaching a score **0.969** for the best regularization parameter.

The best results were obtained with Gradient Boosting with a validation score of **0.976**.

Feature comparison To compare how useful the features are we compare the score of the best method (Gradient Boosting) on the validation set when removing certain features. Our results are presented in Figure 1

Feature group	(i)				(ii)			
Feature removed	(a)	(b)	(c)	**all**	(a)	(b)	(c)	**all**
Validation Score	0.977	0.972	0.977	**0.972**	0.974	0.977	0.976	**0.971**

Feature group	(iii)	(iv)			(v)
Feature Removed		(a)	(b)	**all**	
Validation Score	**0.977**	0.975	0.975	**0.977**	**0.976**

Figure 1: The impact of the removal of certain features on the validation score. The numbers indicate the feature groups described in Section 1. We removed each component of the feature group, and the bold score is without that feature group altogether.
Reminder: with all features we achieved the validation score of **0.977**

We can see that the features that stand out the most using this protocol are the year difference of the article and the TF-IDF distance between the articles' abstracts. Both the first group of features, which was suggested by the baseline, and the group of TF-IDF distances features seem to be equally important. The features built from the shortest paths in the graph as well as the Jacquard similarity allow us to gain the few points which are necessary to ensure a high ranking in the Leaderboard.
One interesting thing to note is that getting rid of some of the features doesn't diminish the validation score that much: our features thus show some correlation and one missing feature can be explained by the remaining ones.
This and the high F1-score (close to 98%) makes us think we have captured most of the information that can be extracted from the graph.