Kaggle challenge report - Team Name

Ombeline Lagé - ombeline.lage@polytechnique.edu
Haris Sahovic - haris.sahovic@polytechnique.edu

January 2019

This report in organised in three sections.

Firstly, we discuss our feature engineering process. Secondly, we describe our learning model and its fine-tuning process. We also compare it to classical methods. Finally, we rapidly discuss possible extension of this work.

We also included a references section which lists the papers we used while building our features and our model.

Feature engineering

Pre-processing and raw data

Authors field

This field is formatted, for each node, as a comma separated list of names, each one corresponding to an author.

Some authors names also include various information about their university or company, usually between parenthesis. We decided to delete this information, in order to unify our authors list. Authors names also undergone a couple of unification steps, such as deleting spaces or hyphens.

For title ans abstracts, we experimented with stemming. As it did not allowed us to improve our accuracy, we decided not to use it in the final model.

Journal field

Similarly, journals names were reformatted, mainly by deleting parasite characters (', ", , /, (,), -).

Year of publication

Every year was converted to an integer.

Usefulness: this feature is probably the most important we have. A simple classifier with access to the year of publication of two nodes can reach an accuracy comprised between 75% and 76%.

NLP features

Naive similarity

Our first NLP feature was a naive measure of similarity between the titles and abstracts of two given nodes.

Given two nodes n_a and n_b, their abstracts a_a and a_b and their titles t_a and t_b, we can define sa_a, sa_b, st_a and st_b, the set of words or a_a, a_b, t_a, t_b.

We can then compute the proportition of words from sa_a present in sa_b and ta_b, and so on.

Usefulness: this feature is relatively efficient given its simplicity. Used with dates, it allows a simple classifier to reach 85% accuracy.

TFIDF

For each node, we compute its tfidf vector for its title and abstract, using the `sklearn` (Pedregosa et al. (2011)) built-in, fitted on our whole title and abstract dataset. We experimented using unigrams, bigrams and trigrams, and decided to use trigram tfidf in our final model, as it gave slightly better results once SVDed.

Usefulness: Not used directly.

TFIDF truncated SVD

Once our TFIDF vectors are computed for every abstract and title, we can apply a truncated SVD.

This feature necessited some fine-tuning when it came to settling on a dimension. We found 128 to be a good compromise between performance, memory usage and overfitting.

Usefulness: this feature is extremly powerful. A simple classifier with dates and a high dimensionnal SVD of TFIDF vectors (~512, which means more than 2000 input features per pair of nodes) reaches an accuracy between 94 and 95 percent.

Authors titles' SVD

For each node, we select all of its authors articles, and apply our above-defined truncated SVD on the agregation of their titles.

This allows us to have a vector representation of each node based on its authors activity.

Usefulness: while useful when used on its own, this feature does not benefit much a bigger model and actually leads to overfitting. Hence, it was not used in the final model.

Common words SVD

We apply our tfidf truncated SVD on the intersection of words between two nodes' abstracts.

Usefulness: as the authors title's svd, this feature is useful on its own but leads to overfitting; it was not used in the final model.

Word2Vec

We made multiples experiments with CNNs. All of them used word embeddings, and most of them used a variant of W2V.

The variants we experimented with are:

- Google's pretrained vectors
- Our own vectors, trained on our data. With these, we experimented with several dimensions. In the end, we found that 128 was once again a reasonnable choice.

Usefulness: as explained in the model section, our final model did not use word embeddings. However, we did get some interesting results with w2v while experimenting.

Glove

When exploring CNNs, we also tried using Glove word embeddings. We directly used the vectors trained by Stanford NLP. The results we got were not better than those obtained by using w2v, so we prefered concentrating on w2v.

Usefulness: less than w2v.

Doc2vec

We wanted to use doc2vec, but had trouble getting it to run in a reasonnable time.

If we had time to expand on our current model, this is probably one of the main features we would like to test.

Graph features

Author cooperation graph

We defined the graph of author cooperation as follows. Each author is a node. Two authors are linked by an edge if they have cooperated on a paper.

Once computed, we can study this graph. We used distances between authors in this graph in a set of features described in the nexrt section.

Usefulness: used with `author_set_comparison`

Author set comparison

Building on top of our author cooperation graph, we define for each pair of articles the following features:

- Number of authors of articles a and b
- Mean distance of authors of article a and article b (among those where such a distance is defined)
- Proportion of pairs of authors without a defined distance
- Minimal distance between two authors of article a and article b

These feature are returned by the function `author_set_comparison`.

Usefulness: When used with dates and naive similarity, this feature gives a boost of a couple percents to accuracy.

Nodes arity

We consider three graphs; the article graph, the journal graph and the author graph. Two nodes are linked by an (unweighted) edge from a to b if there is an instance of a that cites an instance of b.

For each node, author and journal, we compute its in-degree and out-degree.

For authors and journals, we also compute weighted (sum of weight going out of a node normalised to one) edges.

All of these values are fed into our model, for both considered nodes.

Usefulness: These features were added together when experimenting with our model, and allowed us to progress from a f1 score of 94.5 to an f1 score of 95.5.

Adjacency matrices

We build three adjacency matrices for each of our three graphs; one with directed edges, one with reversed edges, and one with undirected edges.

We then compute these 9 matrices to the powers of two, three and four.

The resulting matrices contains, for every entry i, j, the number of paths of lenght k from node i to node j, where k is the power of the matrix.

We can then feed, for each matrix of order two, three and four, given two nodes a and b, their a, b and b, a entries.

Usefulness: These features were extremely useful, and allowed us to reach a 97 f1 score

Limit of these features: Considering two nodes a and b, these features take into account an eventual edge from a to b. This is a problem for our undirected adjency matrix of order 3, as it significantly increases the expected value of the a, b entry.

One way to overcome this limit would be to substract the number of neighbours of b, squared, as it corresponds to the extra induced by the $a \to b$ edge. However, we ran out of time and did not include it in our feature engineering process.

Co-citations

We say that two articles are co-cited if they are both cited by the same node.

The number of co-citations between nodes is used as a feature.

Usefulness: this feature allowed us to marginally increase our accuracy; it was not tested before the introduction fo adjency features.

Graph TFIDF SVD

We create a vector representation of articles based on their position in the graph. To do so:

- For every node, we generate an array containing the list of nodes it cites
- We apply a tfidf vectorisation to these arrays
- We fit a truncated svd base on the resulting vectors

Then, given two nodes, we can return the truncated svd vectors obtained by the list of their citations after removal of a potential citation from one to the other.

This idea is vaguely inspired from Rossi, Zhou, and Ahmed (2017), W. L. Hamilton, Ying, and Leskovec (2017) and W. Hamilton, Ying, and Leskovec (2017), although much simpler.

Usefulness: when used with dates, learned journal embeddings and learned author embedding, this feature reaches an f1 score of 95.

Other features

Learned embeddings

In our neural network, we used embedding layers for journal and authors (for the authors, we fed the results into an LSTM in order to take into account their multiplicity).

These embeddings were low-dimensionnal (6 for the authors, 16 for the journals) and learned during the training of the encompassing neural network.

Usefulness: these features allowed us to take easily and efficiently into account authors and journals. Used with articles dates, these learned features allow us to reach an accuracy of 86%.

Model tuning

Pre and post training tuning

Our feature vectors were scaled to 0 norm and unit variance.

We experimented with dimensionality reduction of the input (SVD and PCA), but it led to significantly poorer results.

Post training, we optimised our threshold for class distinction to maximise our f1 score on the testing set. In this section, every reported f1 score takes into account this optimisation.

CNNs

Before switching back to a simpler neural network, we experimented with CNNs using Keras (Chollet and others (2015)), inspired by results such as Elbayad, Besacier, and Verbeek (2018) and Zhang and Wallace (2015).

Our model had 9 input layers. For both articles, their journals were two inputs fed to an embedding layer, their author lists were two inputs fed to an embedding layer then processed by an LSTM cell, the titles and abstracts (4 inputs) were fed to convolutionnal layers, and these 8 resulting layers were concatenated with a ninth input, containing all the other features we were computing.

We experimented with and without titles and abstracts SVMs as additional features, with and without embedding learning during the training, with Google's w2v vectors, our w2v vectors and Stanford NLP's Glove vectors. For our own w2v vectors, we also experimented with vectors dimension.

Overall, despite some promising results (95.5% f1 score without most graph features), the training was significantly slower to train than the model we switched to, and we recurrently failed to converge to better scores than those obtained with a simpler model.

We therefore decided to abandon CNNs and switch back to a more simple neural network.

Main neural network

Structure

The main model we used is a neural network whose structure is as follows:

- Two input layers for article journals, fed to an embedding layer
- Two input layers for article authors, fed to an embedding layer and an LSTM layer
- One `other_features` layer, containing the rest of our features
- A concatenation of the five resulting layers
- A number of dense layers, with a final 1-dimensional sigmoid output

We used Keras (Chollet and others (2015)) to implement it.

Fine-tuning

Here are the main parameters we experimented with:

- **Dropout value**: we experimented with dropout values comprised between 0% and 60%. For our final submission, we used a value of 40%.
- **Number of layers**: we tried neural networks with one, two and three hidden layers. We found that 3 hidden layers did not improve accuracy over 2 hidden layers - but 2 did over 1.
- **Layer dimensions**: we used different values, with dimensions ranging from as low as 8 units to 1024 units for every layer. In the end, we settled on a first layer size of 256 and 128 for the second one.
- **Activation function**: We tried *elu*, *relu*, *selu* and *tanh* functions. *elu* and *selu* led to better results than *elu* and *tanh*. We used *elu* in our model.
- **Batch size**: we tried batch sizes ranging from 16 to 4096. In the end, and after some addtionnal tweaking, a batch size of 320 was kept.
- **Data augmentation**: our data was scaled to ensure 0 mean and unit variance. We tried adding random noise to training data to reduce overfitting. Gaussian noise with variance ranging from 0.0001 to 1 was tested, without significant improvement in results. We therefore decided not to use it during training.
- **Optimizer**: we compared Adadelta, Adagrad, Nadam, RMSprop and SGD. RMSprop led to the best results, so we used it in the final model.

Comparison with sklearn classifiers

The reported accuracies are based on a 90 / 10 split of the training data.

- **Baseline neural network** : 97.55% accuracy
- AdaBoost: 97.45% accuracy
- Linear SVC: 96.38% accuracy
- Gaussian Naive Bayes: 90.80% accuracy
- Random forest - 10 trees: 97.44% accuracy
- Random forest - 30 trees: 97.73% accuracy
- Random forest - 100 trees: 98.09% f1 score
- Random forest - 400 trees: 98.14% f1 score

Once submitted, predictions obtained from random forests did not top the neural network. We suspect that this is due to the issue related to adjency matrices, discussed in the feature engineering section of this report.

Final model

After trying multiple ways to combine the output of multiple classifiers (including a Logistic Regression), we settled on the following method, which led to our best results and submission:

- Compute predictions by the neural network, the 100-tree random forest and the 400-tree random forest
- Sum the predicted probabilities of a citation existing
- Compute the optimal decision threshold for this sum
- Combining graph and text analysis in a single vector, as is done in Livne et al. (2013)

This method gives an 98.25 f1 score on the training set, and led to a 97.77 f1 score on the kaggle leaderboard.

Extensions

Some techniques that we did not take the time to explore and that might be efficient are:

- Doc2vec
- DeepWalk
- Node2Vec
- Learning node embedding directly and using text features, as in Ganguly and Pudi (2017)

References

Chollet, François, and others. 2015. "Keras." https://keras.io.

Elbayad, Maha, Laurent Besacier, and Jakob Verbeek. 2018. "Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction."

Ganguly, Soumyajit, and Vikram Pudi. 2017. "Paper2vec: Combining Graph and Text Information for Scientific Paper Representation." In *Advances in Information Retrieval - 39th European Conference on IR Research, ECIR 2017, Aberdeen, Uk, April 8-13, 2017, Proceedings*, 383–95. https://doi.org/10.1007/978-3-319-56608-5/_30.

Hamilton, William L., Rex Ying, and Jure Leskovec. 2017. "Representation Learning on Graphs: Methods and Applications."

Hamilton, Will, Zhitao Ying, and Jure Leskovec. 2017. "Inductive Representation Learning on Large Graphs." In *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 1024–34. Curran Associates, Inc. http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf.

Livne, Avishay, Eytan Adar, Jaime Teevan, and Susan Dumais. 2013. "Predicting Citation Counts Using Text and Graph Mining." In, iConference 2013, Workshop on Computational Scientometrics: Theory and Application. https://www.microsoft.com/en-us/research/publication/predicting-citation-counts-using-text-graph-mining/.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

Rossi, Ryan A., Rong Zhou, and Nesreen K. Ahmed. 2017. "Deep Feature Learning for Graphs."

Zhang, Ye, and Byron Wallace. 2015. "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification."