INF554 : Kaggle competition project report

Team Pumpkin

Gaspard Beugnot[1] and Célia Escribe[2]

[1]gaspard.beugnot@polytechnique.edu
[2]celia.escribe@polytechnique.edu

Abstract—**We chose to turn the problem into a classification problem. Given two publications, we use numerous features to predict if one cites the other or not. Our research focused on (i) finding a correct document embedding for the abstracts, (ii) devising features using node attributes or the topology of the graph, (iii) finding a robust classifier to process the data. We got a F1 score of 97.1%. Improvements could be made with the document embedding and the computation costs, which are still expensive.**

I. Word embedding

We first thought that we had to deal with a NLP problem, and that it could be solved using mainly the word embedding of the abstracts. We focused on that in the first place. We searched for the type of word vectorization which worked best, we found the set of parameters which best discriminated our test set, before trying different means to turn corpus of variable lengths into a single vector,

A. Word vectorization

We used **`word2vec`** instead of `TfidVectorizer`. We thought that we had to deal with a fairly **specific** vocabulary: we did not want to use a preexisting word embedding, so we first **build ours using every abstracts**. That task was surprisingly fast to compute. We chose as parameters:

- 300 features
- 5 as the minimum of word count
- 20 for the context size
- A down sampling of 10^{-4}

The text is already preprocessed, we feared the lack of phrases would lower the precision of our model. But a few tests yielded good results. For example, among the list "vector, dimension, algebra, physics", "physics" was said to be the most dissimilar according to the algorithm. We thus hoped it would be able to distinguish whether two articles were similar or not.

B. From list of variable length to a fixed size vector

We encountered more difficulties here. Our first idea was to **average all the vectors and take the L2 distance**. Yet, we thought we were losing much information doing that. We tried using k-NN, comparing only least frequent words, or using convolutional layers.

a) Using k-NN: The idea is to cluster all the words using k-NN, attempting to have 5/6 words per cluster. We are then able to compare two abstracts by looking to their respective partition among the clusters. Unfortunately, even though we managed to have relevant clusters ("matrix, vector, plane"), **we had also completely odd ones which made this measure inefficient**.

b) Comparing least frequent words: Repartition of the number of words per abstract is really wide. Mean is about 100, with a standard deviation of 50. We tried to focus on what we thought contained the more meaning about an abstract, that is the least frequent words, those which characterize it the most. However, taking the n-th ($n \in [10, 50]$) least frequent before taking the L2 distance yielded disappointing results[1].

c) Neural net: Finally, we tried to **build a neural net**. For two abstracts s and t, a word embedding of size w, we tried the following pipeline (fig. 1):

- Choose the n least frequent words in each abstract, thus giving two matrices of size w, n
- Concatenate them, giving a volume of size $w, n, 2$
- Use 2D convolution layers, with a filter of size $1, p$ so as to keep the transformations line-wise, thus focusing on one meaning per line.
- use a 2D convolution layer with **filter of size** $1, n$, to have a vector of size $w, 1$.
- Use dense and pooling layer to have a $0-1$ output.

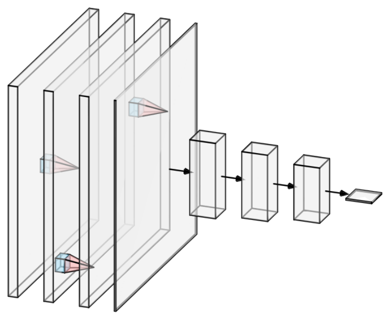

Figure 1: Neural net tried to turn word embedding in document embedding

Unfortunately, this method only yielded a F1-Score of 0.75, which was quite poor compared to **the L2 distance which**

[1]Taking the matching which minimizes the distance might give better result, but is computationnaly expensive.

was able to discriminate about 80% of the set !

C. Conclusion

We found **using word2vec, averaging and using L2 distance** yielded the best results. Yet, we think there's a lot of improvement which could be made there. We considered training our `word2vec` model on external sources, such as physics and maths dictionnaries, or scraping the definition of the least frequent words on wikipedia, but none of us had the skills needed.

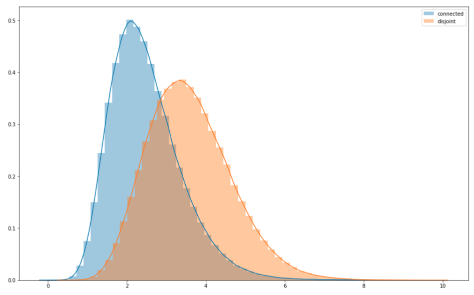

Figure 2: L2 distance between the average of the abstract's words embeddings

II. Feature engineering

We focused our feature engineering on two aspects : **node attributes and topological features**, based on the graph structure.

We first used node attributes (journal, year of publication, title, authors, abstract) to define the following features for a publications pair : **difference between publication years, overlapping words in title, identical journal or not, document embedding for the abstract, number of common authors, affinity between authors.**

We then focused our attention on the graph structure, which led us to define four categories of measures for link prediction : similarity measures, node centrality, community structure and distance measures.

Feature	Category
1. Jaccard coefficient	Similarity
2. Adamic-Adar index	
3. Degree	Centrality
4. Betweenness	
5. Closeness	
6. Eigenvector	
7. PageRank	
8. Clustering coefficient	
9. In same community	Community
10. Shortest path	Distance
11. Shortest path community	

Table I: Samples of features we studied, group by category

In this part, we will present graphs where the features are plotted for connected pairs of publications (blue), and unconnected ones (red), in order to investigate if the features have a high predicting power. We won't specify that in each of the legend. All are not presented for lack of space.

A. Features choice

*1) **Similarity measures**:* A simple approach to link prediction is to rank all pairs of nodes by their score according to a specific similarity index, and take those node pairs with the highest score to be the most likely connected pairs.

a) Jaccard similarity coefficient: The Jaccard similarity coefficient of two vertices is the number of common neighbours divided by the union of the neighbours of both vertices.

b) Adamic-Adar index: This index extends the simple counting of common neighbours to include a term which gives less connected neighbours higher weight.

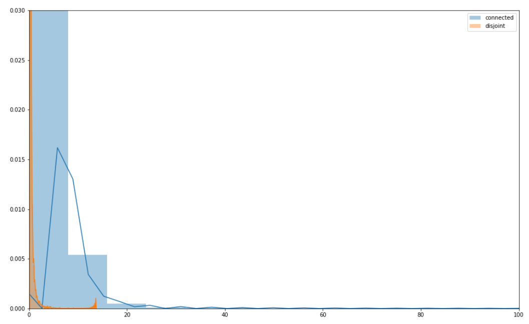

Figure 3: Adamic coefficient

*2) **Centrality measures**:*

a) Degree and eigenvector centrality: These features focus on the number of connections of each node. Eigenvector centrality takes into account the fact that all neighbours are not equal, but should be given a different weight according to the score of their own neighbours.

b) Closeness and betweeness centrality: Both features measure path-based indicators : betweenness measures the extent to which a node lies on the shortest paths between other nodes, while closeness centrality accounts for the mean distance from a node to all other nodes.

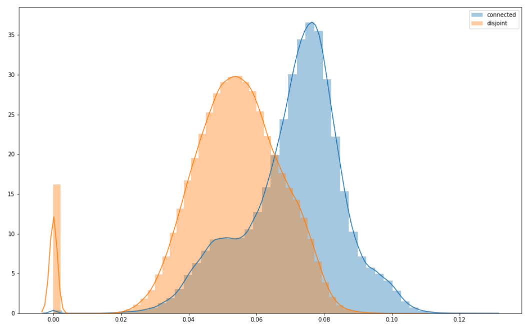

Figure 4: Closeness centrality

*3) **Community structure**:* We then focused our attention on trying to partition the graph into densely connected area. Two nodes belonging to the same community have higher chances of being connected.

*4) **Distance measures**:* Distance measures focus on how many edges separate two nodes. The less edges, the more

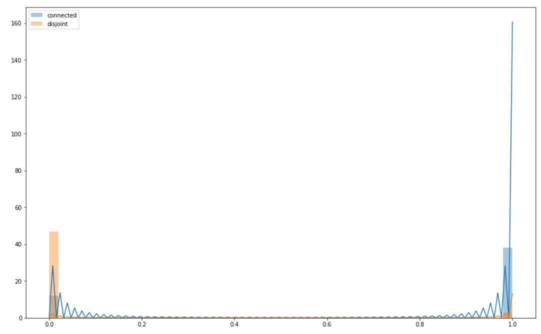

Figure 5: Belonging to the same community

likely two nodes are connected. We used two distance measures : shortest path between two nodes, and the shortest path between the communities to which the nodes belong.

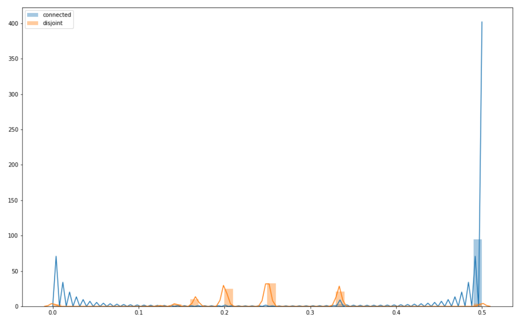

Figure 6: Reciprocal of the shortest path between two nodes

From the different figures presented, we can see that the features chosen seem to have a high separating power.

B. Graph construction

Our first approach was to build the publications graph from all the training set, and to use the same graph to compute the testing set features used to predict labels. However, a problem came from that approach : **we would train the neural net on the training set, for which topological features were calculated using information we did not have for the testing set (the existence or not of an edge between the pair nodes)**. We came to realize this impaired our prediction, when we split the training set in 80/20 and calculated the graph on 80 % of the training set while calculating testing features on the 20 % remaining, for which we ignored the existence of edges. (table II)

Features such as betweenness, closeness or eigenvector centrality had great predicting score on the 80 %, but fared bad when computed on the testing set for which we did not take into account the edges between the nodes. We also noticed that aggregation of centrality features (such as clustering and degree) led to an even worse predictive power loss. On the contrary, features based on the nodes attributes had the same predicting score on the training set and on the testing test, highlighting that the missing graph information on the testing set did not impact the feature engineering.

Feature	training set	testing set
Adamic index	94.3	94.3
Second neighbours	87.6	87.3
Eigenvector	65.1	64
Clustering coefficient	75	73.5
Distances doc embedding	75.5	75.4
Betweenness	76.1	70.2
Closeness	76	70.6
Shortest path	94.4	94.1
PageRank	68.8	67
Is same community	80.0	79.9
Overlapping words in title	61.2	61.3
Number of common authors	61.0	60.8
Clustering + degree	77.0	69.6

Table II: Predictive power of features on the training and testing set

That is why **we changed our approach**, and chose the following one. **We built the graph on 80 % of the training set**, in order to have the most complete graph possible. **We trained our neural network on the 20 % remaining** of the training set (corresponding to 120 000 data, enough to have a good prediction power on the testing set), without taking into account the edges associated with those 20 %. Thanks to this approach, testing set features were similar to those used for the training of the neural network (ignoring the edges between the pair nodes).

C. Feature ranking

In order to evaluate the predicting power of each feature, we used **random forest algorithm with impurity-based ranking**. The principle of this technique is to compute how much each feature decreases the weight impurity in a decision tree.

Feature	Decrease impurity
Adamic index	0.085
Second neighbours	0.0232
Eigenvector	0.0207
Clustering coefficient	0.0165
Distances doc embedding	0.011
Community distance	0.0099
Betweenness	0.0096
Degree	0.0091
Closeness	0.0091
Shortest path	0.0074
PageRank	0.0061
Is same community	0.006
Year difference	0.0034
Jaccard coefficient	0.0032
Overlapping words in title	0.0022
Number of common authors	0.001
Identical journal	0.0005

Table III: Feature ranking based on mean decrease impurity

We should keep in mind that when two features are highly correlated (such as adamic index and jaccard coefficient), this might lead to one of them getting a poor score. That explains the impressive score of adamic index and the poor score of jaccard coefficient. **Adamic index** scores very high, as well as the list of **second neighbours in common**. These features are followed by centrality-based features : **eigenvector, clustering coefficient, betweenness, degree and closeness centrality**. They all score relatively good, indicating the importance of highly influential nodes in the prediction. We can also aknowledge **the importance of the doc embedding distance**, highlighting the information carried in the abstract. However, the rest of node attributes-based features is scoring quite poorly (overlapping words in title, number of common authors and identical journals).

III. Model tuning

We tried different predictive algorithms to predict the existence of an edge between two nodes. We first worked with **support vector machines** to have computationally-efficient methods. However, it soon turned out that **neural networks** had higher predictive score. You will find in table IV comparison between different classifiers, taking into account the following metrics : **F1 score, accuracy, recall score, precision score, squared error, time.**

Feature	**F1**	**Acc**	**Prec.**	**Recall**	**T** (s)
SVM (Linear)	96.0	95.7	97.1	94.9	206
Decision Tree	93.3	95.5	96.7	91.9	5
Naive Bayes	92.9	92.7	**98.3**	88.0	0.1
KNeighbours	95.8	95.5	96.7	94.9	48
Bagging Classifier	95.2	94.9	97.9	92.7	30
Neural network	**97.0**	**96.8**	96.6	94.6	164

Table IV: Comparison between different classifiers

We should note that classifiers fare differently according to the metric measure. The kaggle project took into account the F1 score; even though it took quite a while each time to compute, it yielded the most consistant results. Finally, we should note that Naive Bayes, even if they yield a very poor F1 score, give a really good precision in a very short time.

The neural net we used is fairly simple: we defined a multilayer perceptron with **4 hidden layers of 64 units each**. As we mentioned before, we had problems of overfitting. Thus, we added **dropout** with $r = 0.2$ functions between each layer, to prevent that. We used **classical Adam optimizer**, with **ReLU activation functions** and a **sigmoid activation** at the end to ensure binary classification.

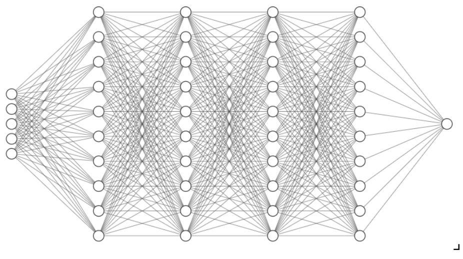

Figure 7: The MLPC we used ; there were 64 units per layer.

Conclusion

All that work led us to a F1 score of 97%. However, we were suprised not to be able to get a few percent more. Indeed, we tried numerous techniques to improve our abstract embedding, going as far as using a promising neural network (fig. 1); we did our best to monitor the behaviour of each feature we added (tables II, III); we changed twice our way to compute the features, to prevent overfitting; we tested numerous classifiers to find the most relevant one (table IV). All that was possible thanks to the rigor we imposed ourselves on the cleanness of the code.

We think the main improvements can come from the document embedding ; despite our best efforts, improvements may also come from a better picking of the features.

Finally, we learnt from this project of the importance of visualizing our features, to have an idea of their separating-power ; we learnt patience, and to overcome disappointment after having prepared for hours a set of features which finally yield poor results ; and most of all, we were harshly remembered of the importance of documentating oneself before commiting to work: indeed, we discovered only very late that existing librairies could have saved us much time.