Link Prediction in the French Web

Project report, December 30, 2019
INF554 – Machine and Deep Learning
MSc&T in Artificial Intelligence and Advanced Visual Computing
École Polytechnique

Martín Cepeda
martin.cepeda@polytechnique.edu

Romain Égele
romain.egele@polytechnique.edu

Cédric Javault
cedric.javault@polytechnique.edu

***Abstract*—Given a subset of the annotated French web graph $G = (V_G, E_G, T_G)$ incomplete on its edges and a set of crawled text for each vertex $T = \{t_v\}_{v \in V_G}$, our goal is to predict whether a pair of known vertices v_i, v_j has an edge connecting them. For this purpose, we first selected a random forest out of many classifiers seen in the course and used it to test many different features and to select efficiently the most relevant ones; the features have to capture both the "structure" of the graph and the "meaning" of the text. Then, we developed a quite sophisticated, home-brewed neural network to get the most of the selected features.**

I. Introduction

A. Working in the right order

Let us define two sub-directed-graph of the French Web $G = (V_G, E_G, T_G)$ and $H = (V_H, E_H, T_H)$ where $V_H \subseteq V_G$ and $T_H \subseteq T_G$ with V_x a set of vertices (representing web pages), E_x a set of directed edges (representing a link from one page to another) and T_x a set of documents corresponding to each node of the graph (representing web pages content). In these settings, the link prediction problem reduce to build a model which for all $e \in E_h$ will output a value in $0, 1$ where 0 means "e does not exist" and 1 means "e exists". We chose to follow a supervised learning approach because our data were labelled. We could have tried other approaches such as unsupervised (e.g. clustering with K-means) but, we did not had time. To validate models we split the original training set in a sub-training set and a validation set. Thus, we were able to identify the so called over-fitting issue and discard bad models. We continued with the following steps:

1) We engineered a features vector for all edges in G and H. We started with low dimensionality (up to 18 dimensions) to efficiently iterate. However, we ended up with a total of 878 dimensions by introducing `Node2Vec` [1] and `Word2Vec` [2] features.
2) We tried different classifiers and selected the best ones: a Random Forest (RF) which was quickly trained and did not need real fine tuning and, a Neural Network (NN) which was more efficient but quite tricky to fine tune.
3) We used the RF to assess the quality of our features and select the best ones.
4) We finally used a sophisticated NN with the selected set of features from the previous step.

B. A methodological bias

We built the features from the original training data set G. Then, we splited in a sub-training set and a validation set. Hence, we introduced a bias in our validation set on features embedding the structure of our graph. We intentionally chose to do so because of the computational resources required to compute some of the features such as: `UMAP` [3] from A the adjacency matrix of G (5 dimensions for each vertex of G, using the Jaccard distance and 17 nearest neighbours which is the mean degree of vertices in G) and `Node2Vec` [1] (128 dimensions for each vertex). Otherwise, it would have been necessary to recompute all these features for our final training before submitting to Kaggle because we assumed it was better to use the whole training set to efficiently solve the link prediction problem which depends a lot on the structure of the graph. Because of this methodological bias we can explain the discrepancy of our local results (0.9678 F1 score) and submitted results (0.9326 F1 score).

II. Comparison of different classifiers

As mentioned in the introduction, we tried different classifiers, always with the same data: a 6 dimension matrix (made of a 3 dimensions `UMAP` embedding for each vertex). This is quite simple but precise enough to see how different classifiers behave.

A. Logistic regression

We used the `scikit-learn` implementation and got a pretty poor accuracy (0.6271), probably because the central assumption of logistic regression (there is a linear separability of the data) which in our case was not met. This is quite logical but it had to be tested.

B. Guassian kernel

In theory, kernel methods could be a good idea because the data are not well linearly separated. However, in practice, using a kernel approach means building a matrix of size n^2 (n being the size of the data set : about half a million). Together with the memory problem, there is a complexity issue as the support vector machine (SVM) algorithm has a complexity between n^2 and n^3. Nevertheless, we tried the Gaussian Kernel of `scikit-learn`, training the model with only 5,000 data and got 0.6148 for the accuracy.

C. k-Nearest Neighbours

The most basic classifier that performs well with non linear data is k-Nearest Neighbours (k-NN). Using the implementation from `scikit-learn`, we had significantly better results. We tried different numbers of neighbours k (Table I) to see how the model behaves :

Nearest Neighbours	Training Accuracy	Nearest Neighbours	Training Accuracy
3	0.8286	13	0.8381
7	0.8368	15	0.8378
9	0.8378	21	0.8363
11	**0.8383**	51	0.8306

Table I
K-NN RESULTS

D. Naive Bayes

We tried the Naive Bayes knowing that it intrinsically raises difficulties for our problem: it requires a probabilistic modelling of the likelihood of features (we had no idea what it could be for `UMAP`... and what it would be later on with other features, so we tried the Gaussian distribution). Additionally, Naive Bayes assumes that features are independent which is of course not the case (especially for the 3 dimensions of `UMAP`). Without any surprise, we got a poor accuracy: 0.631.

E. Random forest

A Random Forest (RF) is a set of binary decision trees. We observed the efficiency of this classifier for our problem and had **our best accuracy so far : 0.87745 with 20 trees** and even 0.8801 with 100 trees. We used the `scikit-learn` implementation as well, (we should have chosen an even number of trees because such as 21 or 101 for a 2 classes classifiers, but never mind).

Apart from having good prediction results, RF had three main advantages: 1. It is very easy to change the number of dimensions (useful later on, when testing many features) 2. The quality of the result do not depend too much on the parameters (no need to fine tune) and 3. The computation is reasonably quick (a few minutes for 100 trees - 550,000 data with 18 dimensions).

F. Deep Neural Network

As it was just a comparative test, we started with a very simple model having one hidden layer (128 units, `relu` activation function) followed by the output layer (1 unit, `sigmoid` activation function). We did not spend time to refine this model but, we got an interesting accuracy of 0.796.

G. *Classifiers : a conclusion*

We tried several classifiers with very different results. We decided to use the random forest to evaluate the quality of features, select the most relevant ones and finally build a sophisticated NN to make the most of the selected features.

III. Building relevant features for the graph

A. Graph, oriented graph, and symmetry

Our graph is an **oriented graph** but we used the **non oriented graph** to build features for each vertex. We reintroduced the orientation by embedding the 2 vertex of each edge in the correct order : for each pair of vertices v_i, v_j, we built a feature vector $f(v_i, v_j) \in \mathbb{R}^d$ which was a concatenation of several descriptors. They are 453,797 lines in the training set ; we expended it to 623,940 by adding a $(v_j, v_i, 0)$ every time we had a $(v_i, v_j, 1)$. Our Random Forest confirmed that it was a good idea.

B. Eigenmap on the Normalized Laplacian matrix

As seen in course and in the litterature [4], we tried to use a matrix-factorization approach but did not succeed to get something relevant.

C. Umap on the Adjacency matrix

In order to capture some of the graph's structure, we worked with the adjacency matrix and compressed it via an `UMAP` dimensionality reduction scheme in order to have a 5-dimensional representation of each vertex. We tried different parameters for `UMAP` and we finally selected: 5 components, 17 nearest neighbours, 0 as minimal distance, 200 epochs.

D. Degree In, Degree Out and Jaccard coefficient

The graph being oriented, it is possible to add the degree in and the degree out for each vertex of the edge (4 dimensions). The Jaccard coefficient [5] between v_i and v_j also captures some relevant information. Indeed, those features improve the accuracy with our random forest: from 0.8750 to 0.9017 with the 4 dimensions of the degree and up to 0.9187 adding the Jaccard coefficient.

E. Node2Vec

The `Node2Vec` [1] embedding algorithm captures a deeper connectivity and structure information of G. We computed the `Node2Vec` in a 128 dimensions embedding vector. Because of its much bigger dimensionality, we kept it for the NN.

IV. Building relevant features from the text

The aim is to capture as much information as possible from the text and to transform it into a vector for each pair of edge (v_i, v_j). It can be information related to v_i, to v_j or to the link between v_i and v_j (e.g. a distance between them). We started with very easy features and progressively built more complicated ones.

A. Size of File

The simplest information available on the texts is probably the size of the file. We used $\log(1 + size)$ which gives information: the accuracy increases from 0.8750 to 0.8866.

B. Languages

While inspecting the texts, we discovered that they were not only in French or even in French and English but they are 42 different languages. More interestingly, the probability of having a connecting edge depends on the languages in $text_1$ and in $text_2$. For example, if the two sites are written in a different language, the probability that they are linked is 0.57 whereas it is 0.68 if they are both written in the same language. However, there is an exception: if they are both in English, it drops to 0.57. We had an improvement when giving those information to the RF: accuracy moves from 0.8750 (only `UMAP`) to 0.8808 (`UMAP` + Languages of Files).

C. Latent semantic indexation

We wanted to compute a meaningful similarity between documents. Hence, we worked on three different Latent Semantic Indexation (LSI) [6] schemes using the `gensim` and `nltk` libraries:

1) We used 5,000 characters per document: accuracy improvement from 0.8750 (`UMAP`) to 0.8878 ((`UMAP`) + LSI).
2) We calculated the Tf-Idf [7] on 50,000 characters per documents (impossible to go further due to computational resources limitations). The accuracy initially was 0.8917. Then, we improved the pre-processing of documents by replacing some character encoding issues which lead us to a 0.8921 accuracy.
3) We built 4 LSI matrices for document similarities among the 4 main languages (English, Spanish, German and French). The documents in the same language are more precisely separated but because 40% of the links are between sites in different languages, we could not improve the accuracy.

For each of these cases we computed the LSI similarity between each pair of documents and observed a much bigger value for the connected nodes (e.g. 0.221 versus 0.098 for LSI in the second scheme).

D. Word2Vec and WDM

At last, we embedded the text-based information contained in T via a trained `Word2Vec` [2] using the `spacy` library. We pre-processed the text and then computed the mean of all word vectors for each t_v. We read about the Word Mover's Distance [8] but understood that we were far from having enough calculating power.

After those from the LSI, the Word2Vec embedding allowed us to compute a new similarity between the edges (e.g. cosine similarity) which improved th RF accuracy from 0.8750 (only UMAP) to 0.8833 (UMAP + Word2Vec distance) and up to 0.8936 by adding both LSI and `Word2Vec` similarities. We tried to compute `Word2Vec` in English for the English sites but it did not give better results than the `Word2Vec` computed in French for the whole dataset.

`Node2Vec` and `Word2Vec` gives high dimension vectors (128 and 300 dimensions respectively for each vertex): we used them to feed the final NN but did not use them directly with RF.

E. Translation issues

We observed a bad similarity between very simple sentences from different languages such as: "Les chiens n'aiment pas les chats." and "Dogs do not like cats.". Furthermore, less than two third of the sites are in French. The other main languages are English (28.5%), German and Spanish (1% each) ; 2.36% of the sites are empty or with so little information that it is not possible to choose the language. As demonstrated before with our simple example, capturing information from texts in different languages is difficult so we wanted to use automatic translation to work with the same language (French) and make them comparable but we did not manage to use this method.

V. Model Tuning & Model Selection

A. Model Tuning

We finally selected the following features: `UMAP`, `Node2Vec`, `Word2Vec`, degree in, degree out, document's language, LSI and `Word2Vec` similarity. The final feature vector had 878 dimensions. We used a $0.77/0.33$ ratio to split our training set $(\mathbf{X}, \mathbf{y})$ in a sub-training set and a sub-validation set. We used an asynchronous model-based search from DeepHyper to search for the hyper-parameters (e.g. units, activation function, dropout rate per layer) of our NN and used only 10% of the sub-training set and 100% the sub-validation, with f1-score as a maximisation goal.

B. Best model

The input data are pre-processed with a `MinMaxScaler` and a `StdScaler`. The classifier is made of several parts. First, each class of feature is processed with a different sub-models.The same sub-model is used to process similar features for both nodes, thus we double the gradient propagation. Then, outputs of these sub-models are concatenated and fed into a final classifier composed of three dense layers. We used the dropout regularisation policy after each fully-connected layer.

VI. Conclusion

During this project, we tried different machine learning approaches which helped us understand their pros and cons. We noticed the impressive performance of RF in addition of its interpretability capabilities by looking at the feature importance which is not possible on NN. As future ideas, we would be interested in improving the pre-processing of our data by looking at effects caused by outliers in the data set. We would like to achieve the translations step and build a more sophisticated latent space to compare the similarity of documents (e.g. recurrent neural network plugged in the NN). Finally we would like to do more structural learning of our data by trying deep graph-convolution neural network and improve the robustness with an adversarial model during the training part.

VII. Annex

In terms of implementation details, we worked in Python with the following learning libraries and frameworks: Tensorflow, scikit-learn, umap-learn, gensim, nltk and spacy

References

[1] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[3] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018.

[4] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," *Advances in Neural Information Processing Systems*, vol. 14, pp. 585–591, 2001.

[5] P. M. Kogge, "Jaccard coefficients as a potential graph benchmark." in *IPDPS Workshops*. IEEE Computer Society, 2016, pp. 921–928.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[7] C. Sammut and G. I. Webb, Eds., *TF–IDF*. Boston, MA: Springer US, 2010, pp. 986–987.

[8] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From word embeddings to document distances," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 957–966.