KAGGLE - LINK PREDICTION CHALLENGE (TM&NLP)

Oblivious NN

16 Mars 2018

Thomas GILLES, Thomas WANG, Marc JOURDAN

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

Table des matières

1 Presentation of the competition

1.1 Objective

A citation network is represented as a graph $\mathscr{G}(V, E)$ where V is the set of nodes and E is the set of edges (links). Each node corresponds to a paper and the existence of an directed edge between two nodes u and v means that one of the papers cites the other. Each node is associated with information such as the title of the paper, publication year, author names and a short abstract.

After having deleted randomly some edges, the *Link Prediction Challenge* had for objective the accurate reconstruction of the initial network using graph-theoretical and textual features, and possibly other information. The score used is the F_1-score.

$$F_1 = \frac{2pr}{r+p} \qquad \text{where} \qquad p = \frac{TP}{TP+FP}, r = \frac{TP}{TP+FN}$$

1.2 Data

The data given where composed of three files :

training_set.txt : 615,512 labeled node pairs (1 if there is an edge between the two nodes, 0 otherwise). One pair and label per row, as : source node ID, target node ID, and 1 or 0.
testing_set.txt : 32,648 node pairs. The file contains one node pair per row, as : source node ID, target node ID.
node_information.csv : for each paper out of 27,770, contains the following information (1) unique ID, (2) publication year (between 1993 and 2003), (3) title, (4) authors, (5) name of journal (not available for all papers), and (6) abstract.

2 Feature Engineering

As for every Machine Learning task, the most important part and the decisive one is the Feature Engineering. We adopt several strategies to capture the most relevant features. The public baselines gives the code to produce three additional features :
Overlap_title : number of overlapping words in title
Date_diff : temporal distance between the papers
Common_authors : number of common authors

We must also pay attention to the fact that we can't use raw text as features since the classifiers don't support it. Therefore we must capture the information encapsulated in the texts in numeric values.

Moreover, as regards the features obtained thanks to the name of the journal, since this information isn't always there, we must compute features accordingly.

2.1 Simple features

Likewise, we add some basic features :
Common_auth_prop : the Jaccard distance (# of words in common divided by total # of words) between the authors
Overlap_journal : number of common words in the Journal
Common_title_prop : the Jaccard distance between the titles

2.2 Text Similarities measures

2.2.1 Word embeddings

Word Embedding is one of the most fundamental component of Text Mining since it allows to have a smaller representation of word for a given vocabulary. Therefore we use Word2Vec to create two separated Embedding, associated with the vocabulary of the title and with the vocabulary of the abstract.

WMD_abstract : Word Mover's Distance (WMD) between the abstracts

WMD_title : WMD between the titles

2.2.2 Similarity measurements

Another way to represent words into vector is to use TfidfVectorizer. We create a dictionary by inserting all the words that are used in a certain category of information of a paper. This thus allowed us to represent the text (abstract or title) into a histogram. Consequently we are able to compare similarity of the histogram to create a new feature.

We applied tfidf multiple ways :

— **words** : Tfidf on words

— **1-3 characters** : Tfidf on list of 1 to 3 consecutive characters

— **1-5 characters** : Tfidf on list of 1 to 5 consecutive characters

2.3 Graph-based features

Many interesting features can be retrieved through the use of graphical representation of the links. As our main source of optimized features, we use the paper *Link Prediction in Social Networks using Computationally Efficient Topological Features* [1] which lists many of them for both directed and undirected graphs. We also added a few features, inspired either from variations of the paper, or from functions available in the NetworkX library we used for the graph implementation.

2.3.1 Vertex features

Each of these features is computed for both the target and the source, independently.

Target_degree, Source_degree : Number of neighbor of the considered node

Target_indegree, Source_indegree : Number of edges directed inwards divided by node degree

Target_outdegree, Source_outdegree : Number of edges directed outwards divided by node degree

Target_nh_subgraph_edges, Source_nh_subgraph_edges : Number of edges in the neighborhood subgraph generated by a considered node.

$$\Gamma(v) = \{u \in V \mid (u,v) \in E \; or \; (v,u) \in E\}$$

$$nh_subgraph(v) = \{(x,y) \in E \mid x,y \in \Gamma(v)\}$$

Target_nh_subgraph_edges_plus, Source_nh_subgraph_edges_plus : Similar to the previous one, except we use $\Gamma^+(v) = \Gamma(v) \cup \{v\}$. We denote *nh_subgraph*$^+(v)$ this value

2.3.2 Edge features

Each of these features is computed once for the pair source-target.

Preferential attachment : Product between the degrees of the source and target

Common_friends : For a given pair of nodes (u,v), we want to know the number of vertices that are at a 1 edge-distance of for u and v. Mathematically :

$$common_friend(u,v) = |\Gamma(u) \cap \Gamma(v)$$

Total_friends : For a given pair of nodes (u,v), total number of nodes in the union of the two neighborhood subgraph.
Friends_measure : For a given pair of nodes (u,v), we want to know the number of vertices that are at a 3 edge-distance of for u and v. Mathematically :

$$friends\_measure(u,v) = \sum_{x\in\Gamma(u)} \sum_{y\in\Gamma(v)} \delta(x,y)$$

$$\delta(x,y) = \begin{cases} 1\, if\ (x,y) \in E\ or\ (y,x) \in E \\ 0\, otherwise \end{cases}$$

Sub_nh_edges : Number of edges in the neighborhood subgraph.
Sub_nh_edges_plus : Number of edges in the neighborhood subgraph$^+$
Len_path : Shortest path length between the source and the target.

2.3.3 Oriented graph features

If we construct the graph using the source-target directions on the edges, we can extract some additional features.

Target_scc, Source_scc : Degree of a node divided by the number of strongly connected components in the neighborhood subgraph.
Target_wcc, Source_wcc : Same as previously, using the number of weakly connected components in the neighborhood subgraph.
Len_path_st : Shortest path length from the source to the target.
Len_path_ts : Shortest path length from the target to the source.

2.3.4 Additional features

Target_clustering, Source_clustering : Compute the cluster coefficient of a node.

$$clustering(v) = \frac{2T(v)}{d(v)(d(v)-1)}$$

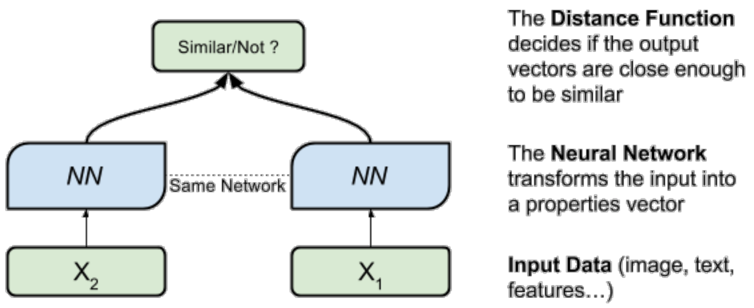
T(v) the number of triangles using v as one the vertices.
Target_core, Source_core : K-core is a subgraph containing only nodes of degree k or more. Ths core number of a node is the highest value of k of a k-core containing that node.
Target_pagerank, Source_pagerank : Use the PageRank algorithm to determine a value for a node. The basis is to imagine an agent randomly choosing edges to go to and outputting the number of times it would go through that node.

2.4 Siamese neural networks

Rather than computing similarities from two vectorization of each text, we can train an artificial neural network to compute such a similarity directly.

To first have the same treatment on both text, and to be able to input them both in the neural network, we use a siamese architecture, such as the one described in *Learning Text Similarity with Siamese Recurrent Networks* [2] : both texts are given as input to layers that share the same weights, we wi. These layers output a vector describing each text. Both vectors are then combined, and the combination is given as input to a few other layers, that finally output the similarity score. The net takes both texts as input, and output a score between 0 and 1 (with a sigmoid activation) ; which is trained with 0 or 1 or the training set, corresponding to the existence of a citation.

This gives us a new feature, a similarity score based on the abstracts, trained to be exactly suited for the problem of link prediction.


Similar/Not ?
NN
Same Network
NN
X_2
X_1
The **Distance Function** decides if the output vectors are close enough to be similar
The **Neural Network** transforms the input into a properties vector
Input Data (image, text, features...)

2.4.1 Siamese layers

We use 3 different kind of siamese layers that will extract the features from the text. These layers have to take the sequentiality of the input into account. With this 3 different approches we have 3 different networks that will give us 3 different features.

Recurrent layers : GRU The most common approach to treat texts with NN is RNN. We chose to use GRU instead os LSTM because it is simpler, therefore it runs faster and overfits less.

Convolutional layers : word level Another approach to treat text are Convolutional nextorks. We inspire ourselves from *Convolutional Neural Networks for Sentence Classification* [3] to design parallel layers with different filter sizes in order to extract different context informations from the text.

Convolutional layers : character level Instead of taking word embeddings as inputs for the convolutional layers, we can use a character embedding, so we can be more tolerant to different orthographs of words. We use an deeper architecter than with the words, inspired from *Character-level Convolutional Networks for Text Classification* [4]

2.4.2 Combination of siamese layers

To simulate a similarity or distance bewteen the two outputs of the siamese layers, we compute both the multiplication term to term and the difference term to term between both vectors, and give their concatenation as input to the next layers.

3 Prediction

In order to predict the existence of an edges given our input feature vector, we use a XGBClassifier available in the XGBoost library which provides scalable and flexible Gradient Boosting method. We use also LGBMClassifier from LightGBM but they didn't score as high as the previous ones. To monitor the training and avoid overfitting we use K-folds method for cross validation. It allows us also to have an average prediction for the testing_set which improves the results since we smooth the result by keeping the interesting trends captured by each training. Curiously, LGBM obtained a similar of even better score on the local CV score, but XGB was significantly better on the leaderboard, so it is the one we chose for our final classifier (blending or stacking both didn't improve either).

One of the huge advantage of this classifier is that it doesn't require much hyperparameters tuning. While keeping the other parameters constant (chosen with manual optimization), we implemented some optimization techniques for the following continuous hyperparameters :

— learning_rate, step size shrinkage used in update to prevents overfitting

Table 1 – Comparison between LGBM and XGB (optimized parameters for both

Score	**LGBM**	**XGB**
5 folds CV	0.98008	0.98007
Public leaderboard	0.97702	0.98192
Private leaderboard	0.97746	0.98052

— subsample, subsample ratio of the training instance
— colsample_bytree, subsample ratio of columns when constructing each tree
— lambda_l1, L1 regularization term on weights
— lambda_l2, L2 regularization term on weights

We use several techniques :
— Pure Random Search (PRS) with the RandomizedSearchCV from sklearn.model_selection, the results are better than the ones of the Exhaustive Search implemented in GridSearchCV
— Bayesian optimization using Gaussian Processes with the gp_minimize from skopt (scikit-optimize)
— Covariance Matrix Adaptation Evolution Strategy (CMA-ES) with the fmin from the cma package
However, the improvements obtained weren't remarkable.

4 Results

Our final submission obtained the public score of 0.98182 and the private score of 0.98057, which gives us the first place of the competition.

The parameters which gave us this result where the following :

```
xgb = XGBClassifier(n_estimators=512, max_depth=6, subsample=0.9,
        colsample_bytree=0.8)
```

Références

[1] Michael Fire, Lena Tenenboim, Ofrit Lesser, Rami Puzis, Lior Rokach, and Yuval Elovici. Link prediction in social networks using computationally efficient topological features. In *SocialCom/PASSAT*, pages 73–80, 2011.

[2] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In *Rep4NLP@ACL*, 2016.

[3] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[4] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.