Team GKP : ALTEGRAD 2019-20 Data Challenge

Pierre ADEIKALAM[1] , **Guangyue CHEN**[1] , **Kevin XU**[1]
[1]Ecole Polytechnique
pierre.adeikalam@polytechnique.edu, guangyue.chen@polytechnique.edu, kevin.xu@polytechnique.edu

Abstract

This report contains our work on the Kaggle Challenge of the course ALTEGRAD 2019-20. The goal of the challenge is to solve a web domain classification problem. Our task is to predict the category to which each domain of the test set is part of. We are provided with the subgraph of the French web graph where nodes correspond to domains and edges correspond to hyperlinks. In addition, we had the text extracted from the pages from the HTML source code for each domain.
We present in this report the different choices made and the experiments performed to solve the prediction task. However, we eventually did not achieve a high rank on the leaderboard for this challenge.

1 Introduction

The objective is to build a model able to predict the type of website a domain belongs to. There are 8 categories :

- business/finance
- entertainment
- tech/science
- education/research
- politics/government/law
- health/medical
- news/press
- sports

We are given a subgraph of the French web graph which have 28 002 vertices and 319 498 weighted directed edges. Each node corresponds to a web host and each edge corresponds to a hyperlink from one domain to another one. Besides, 2 125 nodes from this graph are already labeled and constitute the training set. We aim to predict the 560 domains' class of the given test dataset. The text of the pages of each domain are also provided as .txt files. Furthermore, we begin the challenge with two baseline models, one based only on the graph and the other only using the text files. They reach respectively a score of 1.75 and 1.25 on the public leaderboard.

The main difficulty is to combine these datasets together and build a whole pipeline to predict correctly the categories of the test set.
Firstly, we introduce the different steps undertaken for cleaning the data. Then, we try several methods from simple ones (count number of words in each domain pages) to a bit more complex (Node embeddings) approaches which use efficiently the different datasets. Each method is followed respectively by experiments.

2 Preliminaries : Text Preprocessing

First and foremost, we had at our disposal a .txt text file containing the text of all the pages for each domain. Since the text files are directly extracted from the HTML source code of the pages, the files are quite dirty and contain a lot of useless characters. Indeed, this preliminary stage can help machine learning algorithms to achieve overall better results. Thus, it is crucial to carry out a preprocess step on the text files before any further work. Here is the steps that we operate :

- Lowercase all texts
- Extra white spaces removal
- Tokenization
- Stopword removal
- Punctuation and special character removal
- Stemming using Snowball Stemmer

To perform punctuation and stopwords removal, we used the library NLTK which is a leading platform for building Python programs to work with human language data. We noticed that even though the domains of the graph are French, there are a few number of text files that contains foreign vocabulary. This probably due to the different languages proposed by some websites for foreign visitors. That's why, we use the NLTK's Snowball stemmer [Porter, 2001] for French language to remove morphological affixes from words, which leave only the word stem. This can help to reduce the size of vocabulary. Nevertheless, this stemming task takes about 30 minutes to complete.
In addition, we also considered to carry out part-of-speech tagging on our dataset during the preprocess. But after a few experiments, we concluded this step was unfeasible because it requires a very long computation time and we did not have the

suited hardware. This task would extract for instance nouns or adjectives from the text so we could have reduced even more the number of irrelevant words.

3 Learning Node Representations

Since we are given a graph and the task is to predict the class some nodes, we first address the problem by using primarily methods based directly on the graph. One way was to create a *node embedding* space. Indeed, by embedding networks into a low-dimensional space i.e. learn vector representations for each vertex, with the goal of reconstructing the network in the learned embedding space, we can use them to train a model to classify the vertices in the different categories. Similar nodes would then be close to each other in this low-dimensional space.
For our problem, we explored unsupervised methods and supervised method to capture these embeddings.

3.1 Unsupervised Methods

Unsupervised Methods generalizes language modeling and unsupervised feature learning from sequences of words to graphs. For this competition, we used two well-known Random-Walk-based methods: *DeepWalk* and *Node2vec*, which uses local information obtained from truncated random walks to learn latent representations by treating walks as the equivalent of sentences.

DeepWalk

Deepwalk [Perozzi *et al.*, 2014] is a depth-first traversal algorithm that can repeatedly access visited nodes. It learns representations of a graph's vertices by running short random walks. These representations capture neighborhood similarity and community membership.
Given the current access start node, we randomly sample the nodes from its neighbors as the next access node, and repeat this process until the length of the access sequence meets a preset condition. After obtaining a sufficient number of node access sequences, we use the skip-gram model for vector learning.

Nonetheless, the performance of DeepWalk is the worst among the network embedding methods in most cases. The reasons are twofold. First of all, DeepWalk does not have an explicit objective function to capture the network structure. Secondly, DeepWalk uses a random walk to enrich the neighbors of vertexes, which introduces a lot of noises due to the randomness, especially for vertexes which have high degrees.

Node2vec

Node2vec [Grover and Leskovec, 2016] is a graph embedding method that comprehensively considers DFS neighborhood and BFS neighborhood. It can be seen as an extension of DeepWalk, which is a DeepWalk that combines DFS and BFS random walks. Whereas, DeepWalk uses a rigid search strategy. Conversely, node2vec simulates a family of biased random walks which explore diverse neighborhoods of a given vertex.

Algorithm 1 DeepwWalk

Input: Graph G(V,E)
Parameter: window size w, embedding size d, walk per vertex γ, walk length t
Output: Matrix of vertex representations $\Phi \in \mathbf{R}^{|V| \times d}$

Initialization: Sample Φ from $U^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathbf{O} = Shuffle(V)$
  for each $v_i \in \mathbf{O}$ **do**
    $W_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, W_{v_i}, w))
  end for
end for

Algorithm 2 Node2vec

Input: Graph G(V,E,W),
Parameter: Dimension d, Walk per node r, Walk length l, Context size k, Return p, In-out q

Function: LearnFeature(G, d, r, l, k, p, q)
$\pi = PreprocessModifiedWeights(G, p, q)$
$G^{'} = (V, E, \pi)$
Initialize $walks$ to $Empty$
for $iter = 0$ to r **do**
  for all nodes $u \in \mathbf{V}$ **do**
    $walk = node2vecWalk(G^{'}, u, l)$
    Append $walk$ to $walks$
  end for
end for
$f = StochasticGradientDescent(k, d, walks)$
return f

Function: Node2vecWalk(G(V,E,π), Start node u, l)
Initialize walk to $[u]$
for $walk\_iter = 0$ to l **do**
  $curr = walk[-1]$
  $V_{curr} = GetNeighbors(curr, G^{'})$
  $s = AliasSample(V_{curr}, \pi)$
  Append s to $walk$
end for
return $walk$

Experiment

We implemented in PyTorch the approaches Deep walk and Node2Vec then adapted them to our dataset for comparing with the provided graph baseline model. (See table1)

Compared to baseline model, deepwalk has not improved much, but node2vec has indeed learned some relationships between some nodes. With the help of accuracy rate and embedding visualization(See Figure 1), we find that these representations of graphs vertices are not sufficient but still useful. In the section 4, we tried to use these learned features, and got better scores.

Model	CV-score
Baseline	1.75
Deepwalk	1.75
Node2vec	1.56

Table 1: Node classification log loss score with cross validation

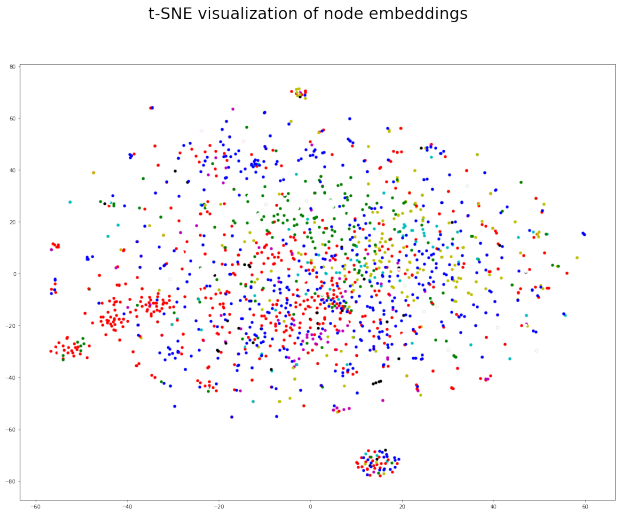

Figure 1: T-SNE Visualisation of the nodes of the training set with Node2vec

3.2 Supervised Method : Graph Neural Networks

The number of labeled nodes divided by the total number of nodes that we can use for training is about 0.07. Our task is commonly know as a semi-supervised learning on graph-structured data problem because we want to use a large amount of unlabeled data during training phase. The supervised methods for this task should in theory outperforms unsupervised methods as experimented previously.

One of the most effective framework for representation learning of graphs is *Graph Neural Networks* (GNN) [Dwivedi *et al.*, 2020]. We can remark that our problem is very similar to the problem raised by the CORA dataset [1]. The task is also given as a graph-based semi-supervised learning, and the goal is to predict the label (seven classes) of a node which represents a scientific publication. And each edge corresponds to a citation link. For this frequently dataset in the field of GNNs, numerous methods have been developed.
The simplest form of GNN is *Graph Convolutional Networks* [Kipf and Welling, 2016]. This approach consists in using neural networks to update weights which capture relevant information. Indeed, by taking in input the adjacency matrix A of the graph and the features of nodes, the model is able to follow a neighborhood aggregation strategy to learn both the graph structure and the features of the nodes.[Xu *et al.*, 2018].

[1] https://relational.fit.cvut.cz/dataset/CORA

Experiment

Since this approach seems to work quite well on the CORA dataset, we adapt the PyTorch implementation of GCN provided by [Kipf and Welling, 2016] to our dataset.

One major difference is the size of the graph. In order to compute the matrix multiplications, we use a sparse-dense format.
In the CORA dataset, the node features are sparse bag-of-words feature vectors for each document. Thus, we use for each text file the TF-IDF weights obtained by exploiting all the text corpus. The node feature matrix contains eventually 28 002 rows and 90 522 columns. The correlation between node features and node labels are essential when it comes to train a GNN model [Duong *et al.*, 2019]. However, artificial features can also be better than real features in some cases.
In order to train our model, we have divided the original train set in **80% for a new train set** and **20% for a validation set**.

The model is made up of three layers : two message passing layers and one layer to compute the probability of belonging to each of the classes. Besides, we apply Batch normalization [Ioffe and Szegedy, 2015] on the two first layers and Dropout [Srivastava *et al.*, 2014] on the first hidden layer to speed up the training process and regularize the model. We use the Adam optimizer [Kingma and Ba, 2014]. On top of that, we also add weight decay [Krogh and Hertz, 1992] to reduce overfitting.

After converting our data in a useable input for the model, we first trained the model for arbitrary hyperparameters. First, we were able to overfit and achieve a high accuracy on the training set by setting the dropout ratio to 0 and removing the weight decay. Figure 2 shows the representation of the nodes present in the training set projected into the two-dimensional space. We observe clearly the eight distinct clusters that constitute our dataset.

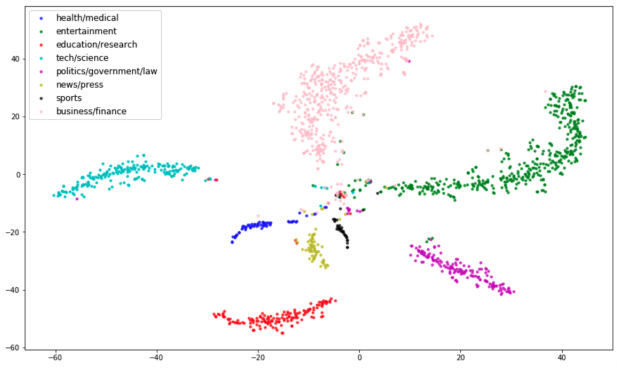

Figure 2: T-SNE Visualisation of the nodes of the new train set with GNN

We subsequently tried to reduce the overfitting by tuning the hyper-parameters in order to produce a correct model
With a grid search on the hyperparameters, we found that the best performances were obtained with 20 epochs, 128 and 64

number of hidden units for the two first hidden layers, 0.01 for learning rate, 0.4 for dropout ratio and 0.01 for weight decay. We obtained an accuracy of 0.3784 and log loss score of 1.6982 on the validation set. These results are unfortunately only just a little better than the provided graph baseline which achieves a score of 1.75
The figure 3 reveals the noticeable inefficiency of the trained model to distinguish accurately the class of the unlabeled nodes.

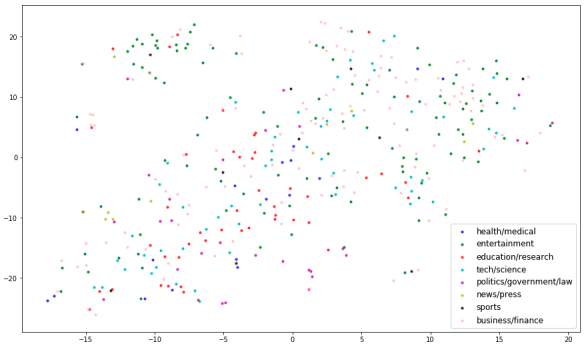

Figure 3: T-SNE Visualisation of the nodes of the validation set with GNN

In short, the model used here did not yield good performance for our problem. The issue is the size of our graph. As elaborated by [Dwivedi *et al.*, 2020], the datasets used to invent these architectures were all quite small. In our dataset, the number of labeled nodes is way too small compared to the total number of nodes. The model is not able to learn an efficient way to classify correctly the nodes.
Moreover, the unsupervised methods yielded better results than the supervised methods in these graph-based approaches.

4 Document classification

For this part, we consider the text files as documents. This task is also called document classification. We experiment a few methods using the hosts of training set to build an accurate model.

4.1 Using TF-IDF

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. Here each document is the text extracted from the web pages of each domain. It is calculated by multiplying two different metrics: TF (the term frequency of a word in a document) and IDF(The inverse document frequency of the word across a set of documents). This method is way more efficient than a simple Bag of Words technique.

In the text-based baseline provided for this challenge, TF-IDF matrix obtained with the original dirty text data of hosts in the train set is fed to a logictic regression classifier to predict the categories of the hosts in the test set. This simple combination yields a score of 1.25 on the leaderboard.

Model	CV-score
Logistic using TF-IDF	1.33
XGBoost using TF-IDF	1.29
XGBoost using TF-IDF with Node2vec	1.25

Table 2: Document classification log loss score using TF-IDF

Baseline : Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

XGBoost

XGBoost is a decision-tree-based ensemble Machine Learning algorithm which is an implementation of gradient boosted decision trees designed for speed and performance. This classifier is powerful for many tasks. However, it has a lot of hyperparameters and it is crucial to tune them in order to get the best performances.

4.2 Final Result

As a result, we combine the features learned from node embedding representation and TF-IDF obtained with the text files of hosts present in the train set only. Then, we optimized by tuning the hyperparameters of XGBoost classifier to achieve our best result. (See table 2) After testing with different graphic embedding methods, we finally selected node2vec which is performing the best, and after tuning the node2vec's hyperparameters: walk length, number of walks, and window size, we obtained the best model. By submitting the results on the test set in Kaggle, we got a feedback score of 1.09280.

5 Conclusion

We have presented all the different approaches we have experimented for this Kaggle Challenge. Graph-based semi-supervised learning is an active area of research and there exists a multitude of methods to solve this kind of problems. First, we elaborated on the preprocessing of the text files which was crucial for the experiments that followed. Afterwards, learning node representations is the first approach we studied. We concluded that node2vec which is a unsupervised method for learning node embeddings worked better than the supervised method using Graph Neural Networks. Furthermore, we achieved much better results using document classification methods. Finally, our highest rank on the public leaderboard was obtained by combining the results of node2vec approach and the results of text-based method using TF-IDF.
Sadly, we did not achieve a high ranking on the public leaderboard after our several attempts.

References

[Duong *et al.*, 2019] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. On node features for graph neural networks, 2019.

[Dwivedi *et al.*, 2020] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. pages 855–864, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[Krogh and Hertz, 1992] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.

[Porter, 2001] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018.