ALTEGRAD Challenge Report

Robin Louiset , **Adriano Del Gallo** and **Geert-Jan Huizing**
robin76lt@gmail.com, adriano.delgallo@telecom-paris.fr, gjhuizing@gmail.com

1 Introduction, inspection of the data

Goal The goal of this Kaggle challenge was to classify website content. To do so, we were given the HTML content of the websites and the links between all of them, each website being represented as a node in a graph.

Baseline methods The implemented baseline methods for each website are:

- a logistic regression after the extraction of features with a TF-IDF method, yielding a log cross entropy loss of 1.25
- a logistic regression based on basic graph features, yielding a 1.75 loss

Challenges At first sight, the challenge looks difficult. Indeed several problems on our data are difficult to overcome, the first one being how unbalanced our data are and how little data there is.

Unlabeled data Our dataset has around 28000 texts. Among these texts only 2120 have labels: this is our train set. Other texts have to be predicted: this is our test set. The remaining are a big resource that we could use for semi-supervised learning.

Unbalanced dataset The training dataset is very unbalanced. Indeed, among the 8 classes we have to predict:

Target label	Proportion on dataset
business/finance	29.46%
entertainment	27.25%
tech/science	13.65%
education/research	9.84%
politics/government/law	9.41%
health/medical	4.33%
news/press	3.91%
sports	2.16%

Table 1: Repartition of target labels in dataset

2 Pre-processing of the data

One of the most important issues of the challenge was the cleaning of our data. Indeed, all the texts have been scrapped from html web pages, but they are not all encoded the same way, they are not all from the same language. We indeed noticed several english texts and few spanish texts.

Encoding error For our pre-processing we implemented several functions. One was aiming at getting rid of the encoding error, we hand-designed a function to replace those by the right character, ie : 'ã§' becomes 'ç'. We then lowered, and got rid of accents, punctuations and numbers, because they are yielding too few information in our opinon.

Images and links processing We also implemented several other tools, such as functions to get rid of images and gif (.jpg, .png, .gif) and links ('www', 'https') with the library 're', which is particularly adapted for strings. Considering our challenge, we thought that getting rid of html residuals woud be a good idea. For that we used the 'Beautiful Soup' library.

Stopwords preprocessing Several other more specific nlp functions have been implemented by our team, such as stopwords processing, to do so we used the library nltk to get rid of english and french stopwords. We also processed too short words processing (1 or 2 letters is not enough to carry information)

3 Text Based Approaches

3.1 TF-IDF

Motivation TF-IDF was used in the text baseline, so we tried to get the most out of it before we switched to more complicated methods.

Tuning hyper-parameters The first parameter to tune is the minimal and maximal frequency of a word in the dataset needed to take it into account. Our best results were achieved when choosing words occurring in 5 to 50% of the dataset. Then we need to tune the logistic regression's parameters. Our best results were acheived with $C = 6.66$, balanced class weight and One-Vs-Rest strategy.

Trying other classifiers We tried using other classifiers, like Support Vector Machines, XGBoost and LightGBM. But we found no improvement on the logistic regression.

Using the unlabeled data We improved our score drastically by using the unlabeled data. Initially, tried using our model to generate pseudo-labels for all the unlabeled data

and then using those as part of the training set. But we found that using too many bad predictions as ground truth made our model worse. That observation led us to the following protocol:

- Train the model with the training set (X_{train}, y_{train})
- Predict the target label probabilities for each of the unlabeled hosts
- Keep only the n hosts with the highest probabilities for any one of the classes : these can best interpreted as the predictions we are the most confident about. We found n close to the training set size to give the best results.
- Assign the predicted label to the selected unlabeled hosts, and add those to the training set.
- Retrain the model

3.2 Doc2Vec

Embeddings We applied Doc2Vec on the aforementioned preprocessing. Our best results were achieved with an embedding size of 60.

Classifier We tried several classifiers : logistic regression, support vector machines, XGBoost and LightGBM. We achieved best performance with an SVM classifier ($C = 4$, RBF kernel), although the results were pretty close to those achieved with the logistic regression.

Unlabeled data In this context we did not reach noticeable performance improvements when using a semi-supervised approach.

3.3 Process documents with Neural Network

Our attentional model

Several models could be emphasized, for example an LSTM approach to catch words temporal-dependency or a 1 dimensional Convolutional Neural Network in order to have a speed-up training and different sort of positional words-dependency caught by our Network. Yet, the major issue here was that the texts are quite long and are divided into several different parts (menu, description, comment, etc).

Attentional approach First we preprocessed and tokenized our documents, kept the 500 first tokens. We then turned them into indices, with a vocabulary built on the test and train documents of size 50000. Bert original paper is here : [2].

Having tested all these previous methods, we ended up with a model being able to assess word-importances based on the overall context of the documents. For that, we implemented an attentional network, a quite famous one : Bert model, with an embedding size of 300 just like in the paper word2vec, 10 number of heads and 10 layers.

Model description We decided not to take the pretrained one, as our problem has a very particular form : our text are composed of different part : menus / comments / description. Also, the information that we can extract from these websites is quite sparse, we decided to take the first words of the documents (up to 500 words we emphasized that it would be enough to carry classification information). We then divided these cropped and padded sequences into 5 sequences of size 100, computed their Bert encoded features map of size (100, 300) for each sequence. Then, we used an LSTM layer with hidden size of 25, to combine our 5 features map into a wise embedding of size (5, 25) which we flattened. The goal being that these embedding would carry different information : the information of the menu (beginning of the text), of the comments (end of the text) and of the website description (middle). At the end of our Network we add a Linear classifier with an output size of 8, the number of classes.

So our final model is :

- Cropping/tokenizing/padding to keep the 500 first words of each documents.
- Word to indices (vocabulary of size 50000).
- division into 5 equal sequences of length 100
- Bert encoding model with embedding size : 300, 10 multi-attention heads and 10 layers.
- LSTM with hidden size 25 on the concatenation of the 5 features-map of size 300.
- Flatten the lstm-output in a 125 length features vector.
- Dense classifier with ReLU, 0.1 dropout, dense layer and then softmax layer for an 8-length output.

You can see a scheme of the model in the appendix.

Our semi-supervised training

We tried several Neural Network models to get good predictions and ended up with a quite complicated model. However, the main issue was actually the data we had at the beginning (only 2100 for training whereas we had around 25000 unlabelled texts on our hands).

Pre-training of our model Because of that we decided to address our problem with a semi-supervised approach. We decided to pre-train our model on the 25000 texts where the label were unknown. To do this, we predicted their TF-IDF probability predictions and train our neural network on these labels. This approach enabled us to get a rather good embedding pretrained without risking overfitting. Also, it enabled us to teach our Network several information such as the importance of words and of their occurences (thanks to TF-IDF), which a normal training would not have caught.

Training After the pretraining, we train our Network on 2/3 of the training data and 1/3 of the testing data. To do this, we set a lower learning rate on our model : 5e-6, and an even lower learning rate : 5e-7 on the words and positional embeddings layers of the Bert because these are very prone to overfitting, especially when there are very few training samples. We took an Adam optimizer with a cross-entropy loss. We succeed to decrease our loss until 1.09.

The time of processing was very long, we used our Google Colab Pro account to access a Tesla GPU in order to speed up our training.

Here are the t-SNE embeddings out of our LSTM :

As we wanted to also extract graph information, we took the LSTM/flatten features vector and injected them into a Graph Neural Network.

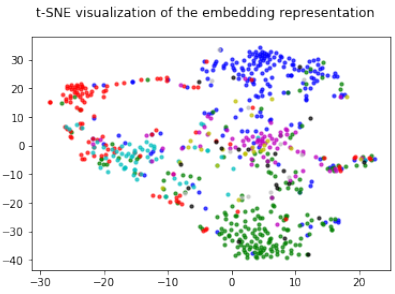

Figure 1: t-SNE visualization of our LSTM output embeddings

4 Graph Based approaches

Motivation Our intuition regarding the graph data is that groups of nodes belonging to the same class should be groups of nodes belonging to the same communities. This intuition is justified because websites about similar topics will tend to share links from one page to another and thus will be strongly connected.

Our first approach consists of doing community detection on the graph and try to regroup members of the same community into members of the same class.

4.1 Clustering

A typical method used for community detection is spectral clustering. [4]. We applied the spectral clustering algorithm followed by KMeans to form 8 clusters (for 8 classes). The classes were then supposed to be decided by majority rule on the train data. The results were disappointing as the resulting clusters were close to randomly sampled ones, most likely because of class imbalance and low number of train data

Consequently this approach was not explored further but it enabled us to understand the graph data a bit better as we learned that the classes were not explicitly represented by communities in the graph. Maybe this approach could have been pursued by detecting sub-communities of the same class, however we prefered to try more elaborate methods such as random walk approaches

4.2 Random Walk Approaches

We now consider another approach to node classification. In this section we try to represent each node as a vector in a d-dimensional space, from which we can perform classification given that the embeddings correctly capture community information.

We first try Deepwalk, [5], an an algorithm that uses random walks as documents and learns representation in a skip-gram inspired fashion. Deepwalk was shown to perform well in community detection tasks since it will be able to find strong relations between nodes that often appear simultaneously in a random walk.

Inspired by deepwalk, we try another random walk approach named Node2Vec [1]. Node2Vec uses the same principle as deepwalk but enables us to control how the random walks are conducted thanks to the parameters p which will control the probability to revisit the previous node and q which will control how much new nodes should be explored. These parameters will enable us to find an optimal balance between BFS and DFS search. The authors of the paper found that emphasising DFS will usually lead to better community detection and that BFS will uncover structural roles of the nodes. We found that among the graph based approaches, Node2Vec was the one who gave the best results. Which is why it was investigated extensively.

The criterion used to determine the quality of the node2vec embeddings was an SVM classifier as well a t-SNE visualization of the embeddings. With optimal parameters we were able to improve the graph baseline score by about 0.5. We hopped to find a better improvement, given the complexity of the method. Figure 4 shows that unsupervised node embeddings do not separate the embedding space into distinct regions for each classes. From this we deduce that the graph itself will not produce adequate results. One must take into account document information to perform classification, or simply refine document information with the graph.

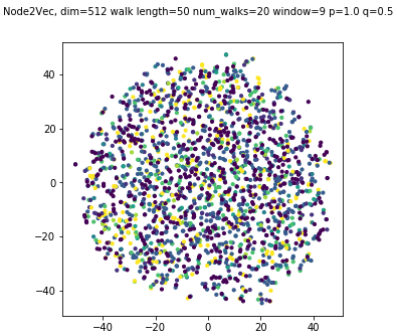

Figure 2: t-SNE visualization of Node2Vec embeddings

5 Combining Graph and Text Data

5.1 Node2Vec and Doc2Vec

One of the first things that came to mind is simply to concatenate the embeddings produced by Node2Vec and Doc2Vec. However the results did not exceed those obtained using Doc2Vec alone.

5.2 Graph Neural Networks

The problem of the previous approach is that graph and text information are learnt independently and then combined. By using graph neural networks we aim to incorporate the extra information contained in the graph to the document embeddings in a semi-supervised manner

GCN

GCN [3] is an architecture for semi-supervised node classification. It takes as an input unsupervised or semi-supervised document embeddings and propagates the information through the network using a message passing scheme. Information of a node will propagate through its K-hop neighborhood. If K is chosen correctly, nodes will share attributes with members of the same community, which is what we wish to achieve for classification.

We use a 2 layer GCN architecture followed by two fully connected layers to perform classification. Numerous parameters have been tried as well as different document representations such as TFIDF, Doc2Vec and semi-supervised embeddings.

The advantage of this approach is that it does not require a lot of labelled data to achieve good results. In the context of our problem this a good thing. However with this approach, it becomes impossible to suppress or add nodes from the graph, which is something one would usually try to balance the classes as would modify the flow of information in the message passing layers

Graph Attention Network

Witnessing the improvement of the score after incorporating graph information, we wanted to further investigate these approaches. One architecture which seemed promising was the GAT (Graph Attention Network) described in [6]. The principle is that node features will be computed by an attention mechanism on the neighborhood nodes. Thus strong links will be given to nodes with a high similarity (via the attention coefficient). This is interesting for us because the attention mechanism will be able to discriminate which members of the neighborhood are from the same class class and ignore the ones that aren't, provided that the initial node embeddings are encode properly class membership.

This is why our best results are achieved on the semi-supervised embeddings obtained from BERT and LSTM.

The final architecture optimised for our problem is as follows

- 1 Multihead Attention Layer with $N_{heads} = 2$ and a hidden dimension $H_{hidden} = 16$.
- The output Multihead-Attention layer with $N_{heads} = 10$.
- A fully connected layer of size $F_c = 30$
- Softmax layer
- the attention dropout ratio is set to 0.15
- the FC layer dropout is set to 0.2
- L2 regularization is done through a weight decay of 10^{-4}.

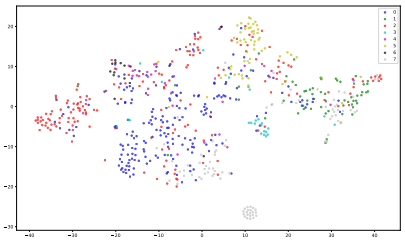

Figure 3: t-SNE visualization of GAT logits

5.3 Averaging several classifiers

One trick that worked very well for us was averaging the probabilities given by different classifiers. For classifiers that are good enough, averaging them will give a better performance than using only the best of them. Indeed, if different methods make prediction errors in different contexts, we should get a reduced variance by combining them.

Our top score on the leaderboard was achieved using the following formula :

$$\frac{4}{5}y_{graphNN} + \frac{1}{25}y_{tfidf} + \frac{3}{25}y_{doc2vec} + \frac{1}{25}y_{node2vec+doc2vec}$$

6 Conclusion and further exploration

We succeeded in using as much information as possible to address this challenge. The major issue was, for us, the use of the unlabelled data and the pre-processing.

We probably could have done more with more time, for example we could have extracted the words of the menu of the website very specifically, we could have translated the non-french websites or translate some documents to another language and back to add more training data. We actually tried to tackle this problem using the Google Translate API but the maximum rate was too low.

The attentional way to process the data was the right way to do it in our opinion but we could have added more information to our graph features, e.g. TF-IDF, topic modelling or menu keywords information.

References

[1] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[2] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[4] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[5] 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.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

7 Appendix

7.1 Attention based model diagram

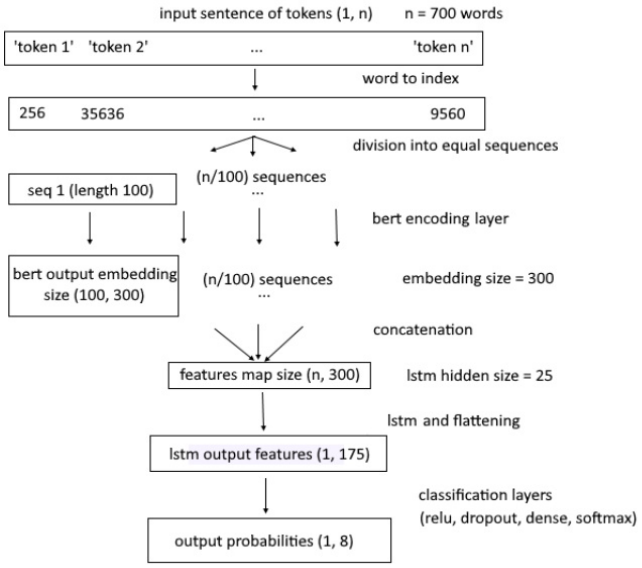

Figure 4: Our model's basic diagram