French web classification : ALTEGRAD challenge 2020

Mohammed Amine ABBOUDI[1] , **Ahmed Taha EDDAHBI**[1] , **Sofiane ENNADIR**[1]
[1]Ecole Polytechnique, IP-Paris
{mohammed-amine.abboudi, ahmed-taha.eddahbi, sofiane.ennadir}@polytechnique.edu,

Abstract

We present in this report our proposed solution to the french web classification data challenge. Our approach is two-fold, the first is mainly graph-based, while the second is purely driven by textual features. As far as the data utilized and our implementation are concerned, the highest performing models has been the one based on text data only.

1 Introduction

The goal of the french web classification challenge is to categorize web domains across 8 classes: ['business/finance', 'education/research', 'entertainment', 'health/medical', 'news / press', 'politics/government/law', 'sports', 'tech/science']. The dataset is made up of a sub-graph of the French web graph where nodes correspond to domains ($\sim$ 28K domains). A directed edge between two nodes indicates that there is a hyperlink from at least one page of the source domain to at least one page of the target domain. Furthermore, your are provided with the text extracted from all the pages of each domain. The train set and the test set contain 1,994 nodes and 560 nodes, respectively. During the challenge, we have tried to implement and test state of the art methods for node classification and/or text classification, we lay out the relevant literature in the next section. Broadly speaking, we explored graph deep learning methods as well as attention mechanisms applied to HTML crawled texts.

2 Relevant Literature

The main problem that we aim to solve falls under the category of node classification tasks. In our case, all graph nodes are coupled with text data that has been crawled from the respective websites making an possible alternative approach, one based on text classification methods. That being stated, a dive into the relevant literature helped us appreciate the intricate similarities between the two tasks. Firstly, a general concept used in both tasks is that of creating representative vectors (embeddings) that give a multi-dimensional quantification of otherwise hardly quantifiable data. In the graph domain, there is **DeepWalk** [6] which uses short random walks to generate a corpus that would help encode the graph structure for each vertex of the graph. It makes the use of Skipgram (*Mikolov et al.*) [5] to achieve the latter step. In the case of text, Word2Vec using the same technique is still the most prevalent way to perform word embeddings. Low-dimensional embeddings of nodes in large graphs have proved extremely useful in a variety of prediction tasks, however most existing approaches require that all nodes in the graph are present during training of the embeddings and so poorly generalizes to unseen nodes. The semi-supervised classification with convolutional networks approach proposed by [4], aims to bridge this gap. A more generalized solution, presented in [3], makes leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Instead of training individual embeddings for each node, we learn a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. In the graph-based part of our report, we implement [4] and supply it with a multitude of features, from a bag-of-words representation to transfer learning from state of the art NLP methods, namely a pre-trained BERT [2] model. As previously mentioned, we divide our work into a graph solution section and a text solution section. Each section is further divided into a succinct theoretical introduction to the approach and the experimental set-up and results.

3 Text-Based Approach

As outlined previously, in this first part, we aim to formulate a solution to our problem by using the text to extract features that could be fed to a classifier to label each nodes.

3.1 Pre-processing

Before using the text of the different documents, we have to accomplish a pre-processing task in order to clean it. Due to the nature of the documents, which are scrapped websites, the first task that we performed was to remove any special characters, punctuation and stopwords in the text, as we thought they were insignificant to our classification task. Afterwards, we have used a French Tokenizer and Lemmatizer in order to extract the different tokens from our text and identify the lemma of each token, enabling us to identify words that were used in different forms. For this task, we used the **`spacy`** package because of its known efficiency in this task. Finally, we decided to only keep the french words since during training, we might be using pre-trained french embedding vectors. In order to facilitate the use of our vocabulary, we have chosen to code the previous task as a class containing the different cleaning steps. (Refer to the `pre-processing.py` code file)

3.2 Term Frequency Approach

We started by applying the classical **TF-IDF** or Term Frequency-Inverse document frequency approach, which is based on the idea of retrieving information from a text by the importance (In term of frequency) of the words that constitute relative to their presence in the rest of the documents. This method require the count of the total occurrences of each word in each document and in all the documents, in order to adjust to the fact that some words appear more frequently, making it a pretty computationally expensive task. After the processing, each document was presented as a vector, where each element represents the number of occurrences of a word in the document divided by the total occurrences of the word in all the documents.

The downside of this method is that it doesn't take into account the context of a word and the different similarities between the words, as it only consider a word as an independent fixed unit.

3.3 Document Embedding Approach

Due to the limitation of the **TF-IDF** method, we have chosen to represent our words as embedding vectors, keeping therefore the similarities between different words. We have used a pre-trained embedding matrix that has been trained using the Wikipedia corpus and is available in the FastText repository, one could refer to [7] in order to have more details on the method of extraction.

A first approach to our classification problem would be to take a mean of the vectors of the words that are contained in each document, resulting in a vector of length embedding for each document. Even though, this method is easy to be implemented, it does not take into consideration a number of important elements, for example the number of occurrences of a word in the sentence. Therefore, we will be using an adapted version of the method that was presented by [1]. This method aims to create a sentence embedding from the embedding vectors of each word and we will be adapting it to create a document embedding using the same analogy.

We will be using the previously computed frequencies that were extracted during the **TF-IDF** approach as a weight to the embedding vectors, and then consider the sum of all the vectors.

In the previously presented algorithm :

- We only consider the words for which we have an embedding vector from the FastText dataset.
- C_w is a normalization constant that aims to keep the vectors in the same range and therefore making its more precise during the training part.

```
Data: Vector Embeddings $V_w$, Frequency of words$f_w$
Result: A 300-Embedding Vector of the document $d$ :
        $E_d$
for all words $w$ in $d$ do
    $W_w = C_w * f_w$ ;
    if The embedding of the word exists then
        $V_w = W_w * V_w$;
    else
        $V_w$ = 0;
    end
end
$E_d = \sum_{w \in d} V_w$;
```

Algorithm 1: Algorithm for computing document embedding vectors.

- We choose to keep the same size of the embedding as the one that was used during the FastText implementation in order to avoid losing information during the process of reducing size, for example using PCA.

The method from which we have adapted our solution is pretty simple and easy to implement. It has also proven to be a very good one, especially in classification tasks.

3.4 BERT Embedding

Another way of generating Embedding vectors is the Bidirectional Encoder Representations from Transformers, otherwise known as BERT. As presented in [2], BERT is applying bidirectional attention of transformers, which is a popular attention model, to language modeling. The bi-directionality of the model enables it to be retreive deeper underlying information from the textual data than a uni-directional language model (which has been used in the previous section).

As presented in [2], BERT uses transformers which is an attention mechanism that has the ability to extract contextual relations between words in a text. In fact, and as opposed to sequential models, which read the text input sequentially, the transformer encoder reads the entire sequence of words at once, and is therefore more accurate and allows to take into consideration the context of the word and its surroundings (left and right).

To be brief, the model consists of feeding word sequences which a percentage (usually 10%) is replaced with a MASK token, and the goal of the model is to predict the original value of the masked words based on the context provided by the remaining words in the sequence.

The implementation of this approach was pretty simple thanks to its use of transformers and the available python package which can be directly used. We have directly fed to the model our pre-processed texts and got as an output an embedding vector for each document, which we will be using afterwards with different models to perform the classification.

3.5 Classification task

After extracting the different features using the previously mentioned methods, we can use them to accomplish our classification task. We used the following approaches:

- A Neural Network that takes as input the document embedding vectors and has as an output a **SoftMax**. We used as an activation function the **ReLU** and the **Tanh**.
- A Logistic Regression model with the "multi-class" parameter turned on in order to allow a multi-label classification. We combined it with a Cross-Validation Grid Search to find the best model parameters.
- Two ensembling methods, namely a Random Forest Classifier and an XGBoostClassifier with a Cross-Validation Grid Search once again in order to fine-tune the classifiers. Because of the number of possibilities, estimators and depth of search, the running time of these methods was significantly longer than the previous one.

Our final submission to the Kaggle challenge consisted of using the BERT features with the Logistic Regression classifier.

3.6 Model improvements

1. Some websites, as mentioned during the questions in the Kaggle Competition Forum, only contain pictures. Therefore and after the parsing, its corresponding documents don't contain any text. In this case, our approach can't succeed to classify its corresponding class since the feature extraction methods return an empty vector. We have chosen to affect to this documents a null vector, hoping that the classifier would find a logical link to deal with these cases. Another approach would be to affect to these documents a probability of belonging to a class that corresponds to the total of documents that don't have a text in that class divided by the total documents that don't have text.
2. A Bagging approach is very beneficial, especially to reduce over-fitting. We have therefore tried to average the predictions that were given by each method in order to have a better score, but this hasn't been the case.

4 Graph-Based Approach

As previously revealed in section 2, the graph-based approach we implemented is graph neural networks (Kipf et al., [4]) specifically targeted at text-aided semi-supervised node classification. This choice was based on two main reasons: firstly because of the seeming simplicity of the core algorithm to be implemented, which entails rapid deployment, run-time complexity and modularity, indeed the model scales linearly in the number of graph edges. Secondly, based on the almost exact match between our two respective tasks, given that part of the conducted experiments by the paper had been the **Cora** citation graph dataset[1].

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

4.1 Basic principle

Let us consider, once again, the problem of classifying nodes, corresponding to french web domains, each accompanied with text documents containing their respective crawled `HTML` content. We especially take interest in the semi-supervised aspect of the method since our dataset has a labeling rate of around 9%, a rate higher than all datasets used in [4]. Given an adjacency matrix $\mathbf{A}$, a GCN layer first computes the following:

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}.\tilde{\mathbf{A}}.\tilde{\mathbf{D}}^{-1/2} \tag{1}$$

where:

- $\tilde{\mathbf{A}} = A + I_n$; **n** being the number of nodes in the graph. The result is the same adjacency matrix with added self-connections so as to avoid divisions by zero.
- $\tilde{\mathbf{D}}_i i = \sum_{j=1}^{n} \tilde{\mathbf{A}}_{i,j}$ is a diagonal matrix.

The previous calculation is a safe row-wise normalisation since we divide each element of the transformed adjacency matrix by the sum of its respective row elements, the output of said layer is therefore:

$$\mathbf{H}^{(l+1)} = \textbf{activation}^{(l)}(\hat{\mathbf{A}}.\mathbf{H}^{(l)}.\mathbf{W}^{(l)} + \mathbf{b}^{(l)}) \tag{2}$$

$\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are trainable weight matrices and biases, respectively. With $0 \leq l \leq L$ where $\mathbf{H}^{(0)} = \mathbf{X}$ is the design matrix and $\mathbf{H}^{(L)}$ is the output of the neural network. The activation function for the hidden layers is **ReLU** while regarding the final output layer we opted for a **LogSoftMax** activation. As a result our loss function is a negative log likelihood loss: **NLLLoss**. This is more advantageous than a conventional **SoftMax**, **MultiClassCrossEntropy** combination. The model consists of two hidden **GCN** layers and takes as input the normalised adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ and the design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, **d** being the number of features per sample.

4.2 Implementations

The basic architecture of the model, as well as the adjacency matrix do not change throughout the different following experiments, only the input design matrix $\mathbf{X}$ does.
We first discuss the constant implementation decisions we have taken, then move on to the details of each incarnation of the feature matrix.
Firstly, as per [4], we do not include a bias in any of the graph convolutional layers. Moreover, we add to the architecture introduced in the paper a third hidden fully connected layer with the same number of 32 neurons.
To sum up, the graph convolutional network contains three hidden layers, the first two are of the type described in equation 2, with $\mathbf{W}^{(0)}$ of size (`d`, 32), $\mathbf{W}^{(1)}$ of size (32, 32) and $\mathbf{W}^{(2)}$ the weight matrix leading to the dense layer is of size (32, 8) along with a trainable bias $\mathbf{b}^{(2)}$.
Regarding weight initialization, we refer to [8] in section 2.3.
Apart from the first feature engineering attempts, all the following approaches are either based completely or partially on section 3.

Bag-of-words Approach

In the same spirit of the original paper, we create a feature matrix based on a simple bag-of words approach. In order to do that, we must first create a vocabulary. After cleaning the crawled `HTML` and removing stopwords, filtering out non-french tokens and lemmatizing them we were left with a vocabulary of size 6950 unique words. All words with an overall recurrence of less than 20 have been removed. It is important to note that the vocabulary was created from processing only the training and testing files and not the entirety of the documents. It has taken just over **7 hours** to generate the final vocabulary, so in order to avoid such long processing time, we include in the data folder a pickle file containing said dictionary.

For each observation in either the training or the testing sets, we constitute a vocabulary-sized vector containing a series of binary values indicating whether each word in the vocabulary is present (indicated by 1) or absent (indicated by 0) in the cleaned corresponding text file.

The main drawbacks of this methods are the same of using a bag-of-words in general: it creates really large sparse vectors, row normalization only makes the situation worse in this case and there is no way to take semantic distance into account.

Self-trained text embedding Approach

Leveraging the outputted vocabulary of the previous method, we constitute a new dictionary which given a word return its index in the embedding matrix. We then generate the final sequence of words for each document, each element being said index. A deeper look at the generated sequences shows a significant disparity in their respective lengths. More than 23% (595 out of 2554) have a zero sequence length, meaning that they either contain no text, the contained text has been completely dismissed post pre-processing or that no words of the retained vocabulary have been found.

Regarding the classifiers, we devise two distinct model architectures. First is a simple Multi-Layer Perceptron, containing two layers. The first is an embedding layer, each row of its weight matrix are the model's vectorial representation of the corresponding word in the vocabulary, we choose an embedding size of 50. Second is a fully connected layer with an output dimension of 8, the number of classes to choose from. It is evident to notice that the link between the two layers is mismatched, the former outputting a matrix of size (sequence length, embedding size) and the latter have a vector the size of the embedding space, we therefore opt for a simple row-wise averaging to represent the embedding of the entire document. The second, more complex, architecture proposed has the same input and output layers as before but injects the following layers in order from input to output: 2 stacked 1D convolutional layers of kernel size 5 and LSTM layer of input and output dimension: embedding dimension.

After training said models we extract the last layer's input (dense layer) and use it as the input of the GCN model in section 4.2, this transfer learning approach allows us to circumvent the non-negligible number of empty input sequences by tapping into the underlying information extracted from the graph structure.

The main drawbacks of this model is that the amount of textual data we have is insufficient to arrive at a beneficial representation of the words present in the different documents.

Pre-trained text embedding Approach

To overcome the "cold start" issue of the previous approach, we make the use of section 3.4 and use it as an input to our graph-based model. Indeed, a version of the BERT model called **CamemBERT** that has been pre-trained on french Wikipedia. The output of the model is vector of size 768 which represents the bi-directional attentive embedding of the document. The tokenization is performed by the model as well.

5 Results and Conclusion

5.1 Results

In order to compare the previously explained and implemented methods, we evaluate the different methods in term of accuracy using a 20 % Test Subset. The following table summarizes the results :

Vectors	LR	RF	XGBoost	MLP
Doc Embeddings	55%	70%	-	54%
BERT Embeddings	60%	96%	97%	70%

Table 1: Results comparison of the text-based approaches.

Feature vectors	accuracy	loss	time
Bag-of-words	43.36%	1.63	21.4s
TF-IDF	42.86%	1.86	46.61s
Self-trained (MLP)	**46.71%**	**1.47**	**9.81s**
Self-trained (ConvLSTM)	39.44%	1.70	8.76s
BERT Embeddings	41.60%	1.53	12.11s

Table 2: Results comparison of the graph-based approaches.

A quick comparison between the two tables shows a significant disparity between the graph-based approaches and the text-based approaches. It seems that GCN does not seem to make use of the graph structure and textual data as shown in [4]. Moreover, as outlined in the included notebook: **`GraphConvNets.ipynb`**, the features extracted from the bag-of-words model as well as the ConvLSTM models overfit heavily. They seem to become almost irrelevant, especially in the case of the former, since the variance of the prediction is very small and the model predicts, after a few dozen epoch, almost always the same probability distribution, that distribution being the empirical ditribution of the train labels. This puts into question the pertinence of not only the features engineered but also the pre-processing techniques used and hypotheses adopted.

5.2 Conclusion

The following project enabled us to learn and apply a number of concepts previously seen in the course in both the graph and text approaches and searched for new ones to complete them. We also had the chance to explored state-of-the-art articles that deal with similar problems which is the key step for every machine learning approach, and that enabled us to have a decent score on the Kaggle leaderboard, which is by the way a very educative way to evaluate this type of courses.

To improve our performance, we could think of better techniques like directly using a BERT inspired multi-class classification model, or looking for other feature extraction approaches that could be more relevant and help us get closer to a more correct prediction.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2019. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.

[2] Devlin et al. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. 2019.

[3] Hamilton et al. **Inductive Representation Learning on Large Graphs**. 2017.

[4] Kipf et al. **Semi-Supervised Classification With Graph Convolutional Networks**. 2017.

[5] Mikolov et al. **Efficient Estimation of Word Representations in Vector Space**. 2013.

[6] Perozzi et al. **DeepWalk: Online Learning of Social Representations**. 2014.

[7] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. **Learning Word Vectors for 157 Languages**. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[8] Glorot X. & Bengio X. **Understanding the difficulty of training deep feedforward neural networks**. 2010.