ALTEGRAD2020 French Web Classification Challenge - BERTExpress Team

Antoine Yang[1] , **Florent Rambaud**[1] and **Arnaud Massenet**[1]
[1]ENS Paris-Saclay
{antoine.yang, florent.rambaud, arnaud.massenet}@ens-paris-saclay.fr

Abstract

In this report, we present the approaches we used for the challenge of classifying the topic of French websites. The information we are given is composed of the text content of websites and a graph representation of the relations between the websites. As in a real world example, we will deal with issues such as having unclean text data and having only a few labeled examples. We first analyze the dataset and particularly the labeled train set. Then we present the different approaches we have tried to exploit the textual data: our preprocessing and tokenization pipeline, TF-IDF coupled with PCA, FastText averaged words or keywords embeddings, or CamemBERT average sentence embedding, different classifiers (Logistic regression, Multilayer Perceptron, Random Forest and XGBoost). We also present the graph pipeline we have experienced: its preprocessing via the creation of subgraphs, the use of embeddings obtained from the Graph Neural Network and Graph Wave coupled with PCA and classifiers (Logistic Regression and Random Forest). The code is availbale at https://github.com/antoyang/Web-Classification

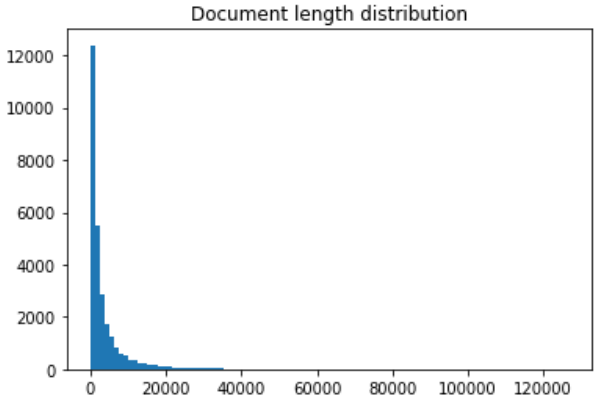
1 Data Analysis

The data of this competition is made of 28,003 documents from the French web. We are given both the content (raw text) and the graph of the documents. Only 2,125 domains are labeled, forming the train set. The test set is made of 560 domains for which we have to predict the most likely category. The distribution of the categories in the train set is described in Table 1.

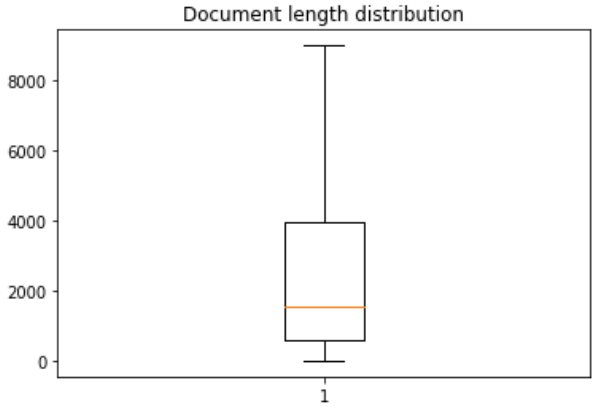
Web domains are a very particular type of textual data. First their length varies a lot: its distribution has a very long tail as shown in Figure 1a. This issue has an impact when building TF-IDF, as it is harder to compare term document frequency with very different lengths between the texts. Figure 1b shows that most of the documents have a length between 0 and 8000. Also the websites can contain words from other languages, such as English or German, which makes difficult the pre-processing task (stopping words, stemming).

Category	Occurrence in train set
business/finance	626
entertainment	579
tech/science	290
education/research	209
politics/government/law	200
health/medical	92
news/press	83
sports	46

Table 1: Categories distribution.



(a) Length distribution



(b) Length boxplot

Figure 1: Documents length

2 Text Pipeline

2.1 Data Preprocessing

After reading the texts of the different websites, we specify French stopwords, usual punctuation, proceed POS-tagging and use Snowball Stemmer (French) to derive French tokens.

2.2 TF-IDF

A first approach is to describe the documents with TF-IDF, compress the information with PCA and train a classifier on top of it. This method is well suited to our problem as the inputs are not well organized and very dissimilar. It focuses only on the vocabulary and its occurrence in order to distinguish the categories.

Here we chose to merge the train and test sets before building the TF-IDF. As we applied a specific preprocessing for French vocabulary, we fed already tokenized inputs to the model. In order to face the wide range of document lengths we used sub-linear term frequency, that is taking $1 + \log(tf)$ instead of tf. We also discarded terms with too low frequency (to have a less noisy vocabulary) and terms with too large frequency (as they do not contain much information).

The resulting matrix has around 18,000 variables. Then, using PCA with 2,048 principal components, we compressed the information and still retained more than 90% of variance explained. In the end we can feed those embeddings to a classifier and compute the probabilities.

2.3 FastText Averaged Word Embeddings

The goal is to obtain website text embeddings that exploit the semantic properties of words in the text and that are hopefully better than the TF-IDF + PCA approach. As previously, these text embeddings will be fed into a classifier.

Therefore we tried to use non-contextualized word embeddings. Word2Vec (Mikolov, 2013) French version was not easy to find, which is why we used FastText (Joulin *et al.*, 2016) (the weights of the model trained for French is provided at https://fasttext.cc/docs/en/crawl-vectors.html).

The first thing we tried was, for each website, averaging the word embeddings (of size 300 with fastText) of all tokens of its text content to derive a text embedding.

However, this naive approach might be sensible to noise (for some documents, dozen of thousands words are averaged all in a 300 dimensional vector). In a finer approach, we tried to use either TF-IDF or PageRank to derive keywords (we tried 20 or 50 keywords per website) just like in the first practice session of the course, and average word embeddings only on these keywords.

2.4 CamemBERT Averaged Sentence Embeddings

To improve it, we thought to build on the models that have established themselves at state-of-the-art in so many tasks in Natural Language Processing, exploiting contextualized information, the transformers. We used a BERT (Devlin *et al.*, 2018) version for French called CamemBERT (Martin *et al.*, 2019), of which the implementation by Hugging Face is open-source at https://github.com/huggingface/transformers.

However, we thought it would not be wise to use contextualized word embeddings directly as the texts corresponding to each text is rather jerked (many short sentences, or parts of text delimited by dots that are not even sentences), or even sequence classification that is more adapted to sentences. Instead, for each text, we derive for each of its sentences its embedding (via the embedding of its CLS token), which is of dimension 768, and we average them to obtain website text embeddings.

Beforehand, we derive the sentences of each text by cleaning each of the documents' text and by using NLTK's sentence tokenizer. Essentially, we remove any characters that will not convey any meaning to the sentences (eg. *, & etc...). Then we recover the sentences by applying NLTK's tokenizer. Although the tokenizer usually performs well, it fails to detect "ill-formed" sentences. For example, if we have a list of buttons with text in a webpage, the tokenizer will tokenize the text on the buttons as a single sentence. As such we use a second set of filters to remove these "ill-formed" sentences. We remove any sentence that has less than 5 words and we remove any sentence that spans too many lines. By doing so, we remove sentences that are likely to carry no information about the website as they are too short and we also avoid "ill-formed" sentences that carry no meaning.

2.5 Classifiers

Once we have embeddings, we train classifiers to solve the multi-classification task. We implemented 4 different classifiers:

- Logistic Regression, using Grid-Search for hyperparameters tuning (searching for penalty L1, L2, ElasticNet or None and the inverse of regularization strength C = .0001, .001, .01, .1, 1, 10, 100 or 1000)
- Multi Layer Perceptron, using Grid-Search for hyperparameters tuning (using as hidden layer size 100, searching for number of layers 1 or 2, learning rate .005, .001 or .0005)
- Random Forest, using Random-Search for hyperparameters tuning (searching for the proportion of maximum features .4, .5, .6, .7, .75, .8, .85, .9 or .95, maximum depth of trees 20, 30, 40, 50, 60, 70, 80, 90 or 100, minimum number of samples per leaf 3, 8, 10, 15, 20 or 50 and number of estimators 50, 100, 150 or 200)
- XGB classifier, using Random-Search for hyperparameters tuning (searching for number of estimators from 100 to 10000, learning rates .5, .2, .1, .07, .05, .04, .03, .02 or .01, minimum split loss or gamma 0, .01, .05, .1, .5, 1, 1.5 or 2, maximum depth 3, 4, 5, 6, 7, 8, 10, 13, 15, 20, 30 or 40, minimum child weight 0 to 10, subsample ratio of columns .8, .85, .9, .95 or 1, subsample ratio of columns for each level .8, .85, .9, .95 or 1 and subsample ratio of columns for each node .8, .85, .9, .95, 1.)

Note that we evaluated our embeddings and classifiers using log loss metric over 5-fold cross validation. We obtained our best score with the Logistic Regression. The Random Forest and XGB classifiers would tend to overfit the train set, as we have many features (embeddings dimension still large even after the PCA) and only few labeled data, despite the random searches on the various regularization parameters.

Our best public leaderboard performances computed by a single model were reached with the CamemBERT embeddings plus Logistic Regression with C = 10 and L2 penalization (1.08128 log loss) and TF-IDF + PCA embeddings with C = 100 and L2 penalization (1.07780 log loss).

2.6 Test Time Augmentation

After getting the score of the CamemBERT embeddings and the TF-IDF + PCA embeddings plus Logistic Regression, we decided to combine both result by averaging their probability estimates. This gave us a public leaderboard performance of 1.01643 log loss.

Following on this idea, we trained a classifier based on the concatenation of the CamemBERT and the TF-IDF embedding using a Logistic Regression. We generated the results and averaged them again with the CamemBERT and the TF-IDF embedding results. We obtained a score of 1.03400 log loss. Note that this result is not very suprising since the performance of the concatenated classifier was not as good as the ones based only on CamemBERT or TF-IDF.

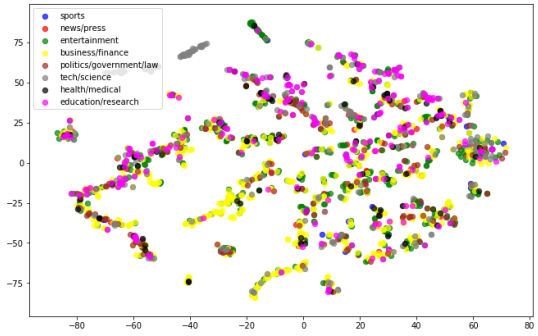
2.7 Concatenated Embeddings

Instead of averaging probabilities, we also tried to concatenate our two best embeddings (CamemBERT and TF-IDF) and train a classifier on top of that. However, this did not result in better cross-validation results. As the modelisation already suffered from high dimensionality, concatenating the input is too ambitious with only few data.

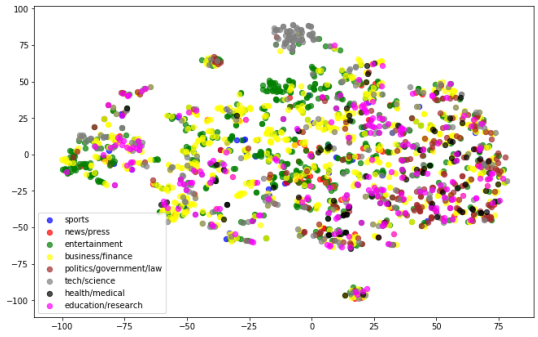
2.8 Embeddings analysis via tSNE visualization

We propose a method to analyse the quality of our different embeddings, based on PCA + t-SNE representation as in a previous homework, on the labeled websites of the training set. Heuristically, the more separated the clusters corresponding to each label in the representation, the more discriminative are the embeddings. One should also note that the different embeddings are of different size (2048 for TFIDF, 300 for FastText, 768 for CamemBERT), and in a significantly higher dimension space that 2, which means we might not be able to observe clear separations even in ideal case.

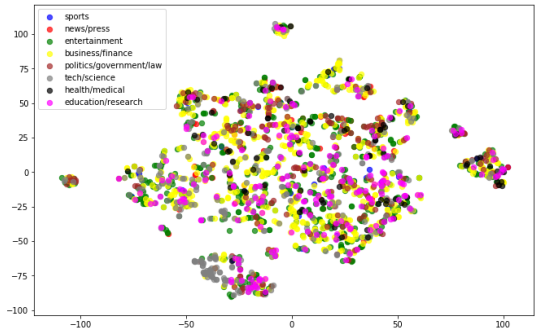
From Figure 2, we can see that TF-IDF embeddings are more discriminative than FastText and CamemBERT, which we could explain by the fact that websites texts are not organized, noisy and jerked. However, using both CamemBERT and TF-IDF embeddings proved to be our top solution as discussed previously.



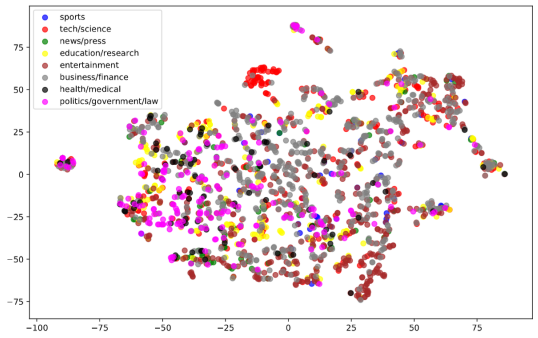
(a) TF-IDF



(b) FastText



(c) FastText on 50 TF-IDF derived keywords (maximum) per website



(d) CamemBERT

Figure 2: t-SNE visualization of different website text embeddings on the labeled train set

3 Graph Pipeline

In this section we will present our approach when dealing with graph data. First, we will briefly discuss the preprocessing tricks that we used to ease the training of our models. We will then discuss the models that we used to find graph embeddings.

3.1 Data Preprocessing

Regarding the graphical data, we have to deal with 27941 nodes among which only 2125 are labeled. This poses two problems:

- We have very few labeled data
- The Graph size might prevent the algorithm to run on our machines

In order to deal with both problems at the same time, we create a subgraph only made of labeled node. This allowed us to have a rough idea of how each of the algorithms could perform on the whole data while saving great amounts of computing power and time.

In order to create the subgraph we started by looking at all labeled nodes n_l and added the nodes connected to these via an edge. By doing so, we lose no information about the close-neighbourhood structure of the labeled nodes. In order to evaluate the performance of our models we cut our subgraph in two and remove any edges that exist between the two resulting subgraphs.

3.2 Graph Neural Network

We start with the Graph Neural Network, which uses the adjacency matrix of the graph to compute the embeddings of the nodes.

Given the weighted adjacency matrix A of the graph G we normalize A:

$$\hat{A} = \bar{D}^{-\frac{1}{2}} \bar{A} \bar{D}^{-\frac{1}{2}}$$

With $\bar{A} = A + I$ and $\bar{D}_{ii} = \sum_j \bar{A}_{ij}$. Note that this process helps with numerical instabilities that can arise when we have nodes with large number of edges with large weights. After this normalization process, we use a three layers network that takes a feature matrix and the normalized adjacency matrix as input. The feature matrix's i^{th} row is the feature vector of node i

Each of the layers for $i = 1, 2$ is of the form:

$$Z_i = f(\bar{A} Z_{i-1} W_i)$$

with Z_0 is the feature matrix, f is the ReLu activation function and the last layer is of the form $Y = softmax(Z_2 W_3)$.

During our experiments we used 16 nodes on the first layer and 32 on the second one. We will discuss the results in the result section.

3.3 Graph Wave

Since the Graph Neural net is limited in its ability to capture structural information, we decided to use the Graph Wave Net (Donnat *et al.* (2018)) for creating our embeddings.

This methods allows to capture spatial information about the nodes and thus, it could help us uncover information about node to node "distances" that we cannot capture with a simple Graph Neural Network. This approach relies on the use of spectral graph wavelets. Essentially, for each node, we propagate a unit of energy over the neighbourhood of said node to map the topology of the neighbourhood and get a structural representation of the node based on that topology.

This approach, in contrast with the Graph Neural Net, will not only at how close two nodes are in the graph, it will also capture how close the neighbourhood's structures of two nodes are.

For the implementation, we relied on the library provided by the Stanford Network Analysis Platform Leskovec and Sosič (2016) which can be found at https://github.com/snap-stanford/graphwave.

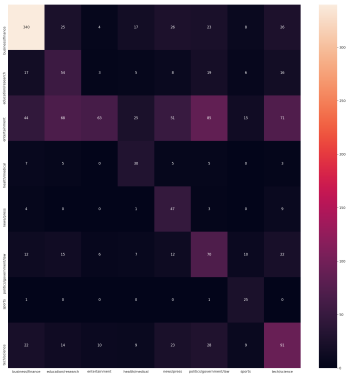
After applying the GraphWave model to get embeddings, we reduced the size of the embeddings using a PCA and then passed the embeddings into various classifiers taken from sklearn. These include RandomTree, SVM and Logistic Regression. We will briefly discuss the result of the approaches.

3.4 Results

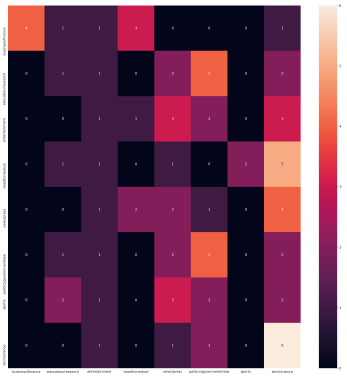
We ran the Graph Neural Net for 100 epochs with 70% of the labeled node as train data and 30% for validation. Despite our attempts to use regularization, by using dropout and changing the number of nodes per layers, we did not go above 3.89% of accuracy on the validation set while the validation accuracy was at 42.96%.

Although the GraphWave algorithm performed better, we only reached 21.6% of accuracy on the validation set while had 50% of accuracy on the train set. The best classifier was the Logistic Regression, as both the Random Forest and the SVM tended to overfit even more. The overfit issue can be better observed on Fig. 3. We can see in the train confusion matrix that the model is still able to give use good results but the classification are all over the place when we deal with the validation set.

Overall, the approaches we used for the graph based approach performed poorly. It seems that the models that we use were not able to fully grasp the relation between the node structure and its class.



(a) Train



(b) Validation

Figure 3: Confusion Matrices with Graph Wave

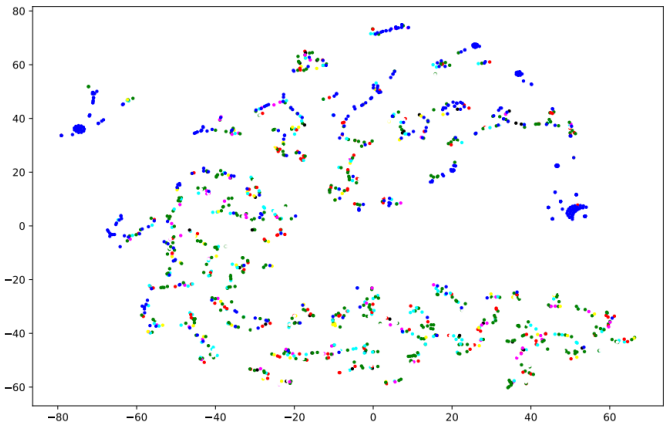


Figure 4: Node Embedding representation with GraphWave

4 Conclusion

To sum up our approach, we first analyzed the task and its inherent difficulties, mainly due to the lack of organization of the text of websites, but also to the fact that most of them were in French (and not in the most common language studied, English) and some were even in another language. Additionally, only a small part of the provided train subset was labeled. We tried various approaches to derive powerful text embeddings, first by combining TF-IDF and PCA, then by trying FastText averaged words or keywords embeddings over each text, then by using CamemBERT averaged contextualized sentences embeddings. In terms of classifiers, we tried Logistic Regression, Multilayer Perceptron, Random Forest and XGBoost Classifiers, of which we tuned the hyperparameters either via Random Search, either via Grid Search. We achieved our best result on the public leaderboard of 1.01643 log loss by combining predictions of a Logistic Regression trained on TF-IDF + PCA embeddings and a Logistic Regression trained on CamemBERT embeddings. We also tried to exploit graph data deriving embeddings obtained from Graph Neural Network and Graph Wave coupled with PCA and using Logistic Regression and Random Forest classifiers. Ideally, we could have used these to propagate labels in the unlabeled websites of the train set.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *International ACM Conference on Knowledge Discovery and Data Mining (KDD)*, volume 24, 2018.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models, 2016.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert: a tasty french language model, 2019.

Sutskever I. Chen K. Corrado G. S. & Dean J. Mikolov, T. Distributed representations of words and phrases and their compositionality, 2013.