French web domain classification

Khaoula Belahsen[1,2] , **Aicha Boujandar**[1,2] , **Emma Demarecaux**[2,3] ,
[1]ENS Paris Saclay, [2]ENSTA Paris, [3]Ecole Polytechnique
khaoula.belahsen@yahoo.fr, aicha.boujandar@gmail.com,
emma.demarecaux@ensta-paris.fr

Abstract

The Kaggle competition *French web classification* comes as 2020's data challenge which concludes the Advanced Learning for Text and Graph Data (ALTEGRAD) class taught by Michaelis Vazirgiannis. This year's challenge consists in building the highest performing model that classifies the French web domains. We have to keep in mind that data was manually annotated and is therefore prone to some noise.

1 Team Members

Our team name for this challenge is **Belahsen_Boujandar_Demarecaux**. The team is made up by **Khaoula Belahsen**, **Aicha Boujandar** and **Emma Demarecaux**. Our full code has been uploaded on GitHub.

2 Introduction

During this competition we have tried to reach several goals:

- Applying what we have seen in class and during the labs;
- Trying to apply some state-of-the-art techniques to include text and graph data in the prediction;
- Reaching a good performance on the Kaggle leaderboard;
- Organizing the code, providing explanations of each step.

In this report, we summarize the research efforts and the different experimental ideas that we have tried, before going into detail about the final pipeline we concluded to for this challenge.

3 Presentation and Evaluation Metric

The web domain classification is performed on a subgraph of the French web graph where nodes correspond to domains, and 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. In this subgraph, we have 28002 nodes and 319498 weighted directed edges.

For each web domain, the text extracted from all its pages is also provided. A subset of these web domains were manually classified into 8 categories and split to train and test sets (with respectively 1994 and 560 elements). The objective is to predict to which categories belongs each domain of the test set. The performance of the model will be evaluated using the multi-class loss function:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} log(p_{ij})$$

where: $y_{ij} = \mathbb{1}_{i \text{ belongs to class } j}$.

Category	Number of train domains
business/finance	587
entertainment	550
tech/science	257
education/research	203
politics/government/law	189
health/medical	83
news/press	79
sports	46

Table 1: Labels of the 8 categories.

4 Data Analysis

From the train data without duplicates, we have the domain distribution in Table 1. Let us look at the subgraphs of some domains:

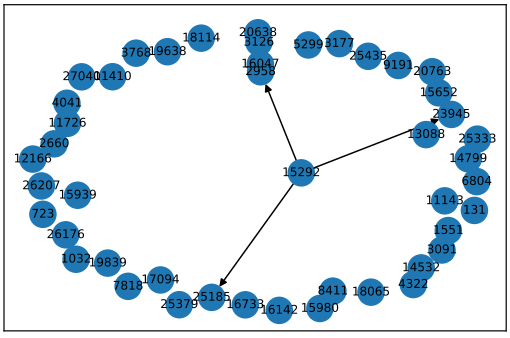

Figure 1: Subgraph of the domain sports.

Among the 46 nodes belonging to the sports domain, there are only 3 internal connections among them (Figure 1). The adjacency matrix for this graph is very sparse and it might be difficult to use the structure of this graph to gain an uplift on the classification task for this particular domain.

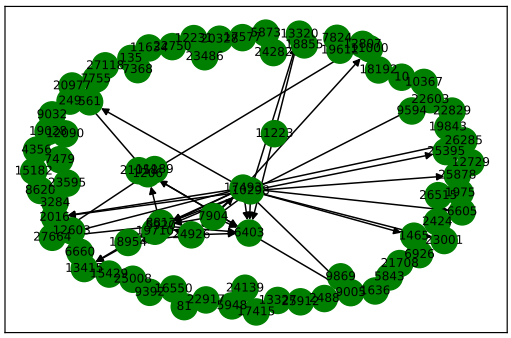

Figure 2: Subgraph of the domain health/medical.

We can make the same remark for the domain heath/medical (Figure 2). There seems to be too few connections between the nodes to make a good prediction. We have noticed the same pattern for the other six domains except for politics/government/law which has 189 nodes and a lot of connections between the web pages (Figure 3). Therefore, the use of graph data might benefit the predictions of this specific domain.

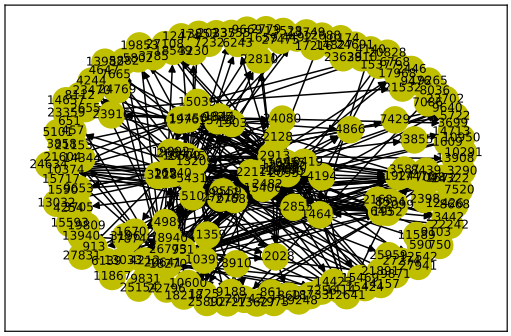

Figure 3: Subgraph of the domain politics/government/law.

However, the graph structure on its own might not be sufficient in order to have good performances for the classification task.

5 Text Preprocessing

Before defining any model, we need to clean and preprocess the raw data. When dealing with textual data it is important to implement various preprocessing steps in order to normalize the data as much as possible, so as to reduce noise. Normalization aims at putting all the texts on a level playing field, for instance, converting all characters to lowercase. Noise removal cleans up the text, for instance, remove extra white spaces. We performed the following steps implemented in the `clean_host_texts` function, found within the `preprocess.py` file.

- Replacing the ”latin1” encoding by the ”utf8” ignoring errors (in order to keep the accentuation);
- Converting all texts to lowercase;
- Removing HTML tags;
- Removing any url;
- Removing any e-mail address;
- Removing any comment or image between square brackets;
- Removing any string of the form ”string_with_special_characters>”;
- Removing punctuation;
- Removing numbers;
- Removing extra white spaces;
- Tokenization: splitting strings of text into smaller pieces, in our case, into words based on white spaces;
- Removing stop words: there are very common words, for instance ”je”, ”suis”, ”le”, ”la”, ”les”, and they probably do not help at all in the text classifications. Therefore, we can remove them to save computing time and efforts as some texts are pretty large. The content of the web pages is in French, but some web pages themselves are in English. Therefore, we have chosen to remove both French and English stop words;
- Stemming, which is the process of converting a word to its root form, for instance: ”winner”, ”winning”, and ”win” would be stemmed into ”win”.

As an example, Figures 4 and 5 show the same extract of a web page respectively before and after performing the preprocessing steps.

```
le gretsi et le gdr isis organisent depuis 2006 une
école d'été   annuelle en traitement du signal et des
images. ouverte à toute   personne intéressée
(académique ou industrielle), elle s'adresse
prioritairement à des doctorants ou chercheurs en
début de carrière, et   a pour but de présenter une
synthèse ainsi que les avancées les plus   récentes
dans un thème de recherche d'actualité.
```

Figure 4: Lowercased web page extract before the preprocessing.

```
grets gdr isis organisent depuis écol annuel trait
signal imag ouvert tout person intéress academ
industriel adress prioritair doctor chercheur début
carri présent synthes ains avanc plus récent them
recherch actual
```

Figure 5: Lowercased web page extract after the preprocessing.

Finally, we noticed that among the data, a lot of nodes present some text errors as in Figures 6, 7 and 8.

Figure 6: Full text of node 906.

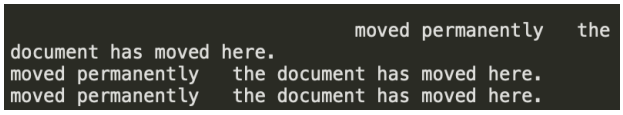

Figure 7: Full text of node 3145.

Figure 8: Full text of node 14841.

Those nodes are very noisy for the classification task as all the words present in those texts have nothing to do with the node labelled domain. Therefore, we decided that texts containing sentences such as:

- "the document has moved here",
- "the requested url was rejected",
- "301 moved permanently",
- "403 forbidden",
- "the requested url was not found on this server",

and having a length lower than 2000 characters, would be replaced by an empty string in order to avoid giving importance to noisy words which are not relevant for the classification task.

6 Work and Ideas

We tried to implement different methods for text classification, using texts only, graphs only, and by creating embedding features for both texts and graphs.

With the text data, we considered traditional machine learning approaches for word embeddings such as `TF-IDF`. We also used deep learning low-dimensional continuous space embeddings which can capture the similarity between words and escape the curse of dimensionality.

With the graph data, we tried several approaches such as `Deepwalk` and Graph Neural Network (GNN). We also combined the power of both sources of data, and we tested different classifiers.

Later on, we focused our work on parameters tuning, preprocessing and feature engineering for a `Logistic Regression` model with only the texts, as graph-based models did not perform as well as we expected. In fact, graph models usually involve deep neural networks and due to the lack of graph data, the networks that we tested overfitted largely even with regularization.

7 Models

7.1 Text Embeddings

In order to learn texts features, we have tried the following methods:

`TF-IDF` Vectorizer

`TF-IDF` is shorthand for term frequency-inverse document frequency. It is a common tool in Natural Language Processing (NLP). Term frequency (TF) determines how important a word is by looking at how frequently it appears in the document. It measures the importance of each word. If a word appears a lot of times, then the word must be important. Inverse document frequency (IDF) is used to calculate the weight of rare words across all documents. The words which occur rarely in the corpus have a high IDF score. During fitting, the `TF-IDF` function discovers the most common words in the corpus and saves them to the vocabulary. A document is transformed by counting the number of times each word in the vocabulary appears in the document. The weight of each word is normalized by the number of times it appears in the corpus. So a word that appears in only 10% of all documents will be assigned a higher value (and thus treated more importantly) then one which appears in say 90% of documents. `TF-IDF` are sparse vectors where the number of non-zero values in the vector is equal to the number of unique words in the document.

For a term t, a document d belonging to a set of documents D, the term frequency–inverse document frequency is computed as follows:

$$\text{TF}(t,d) = 0.5 + 0.5\frac{f_{t,d}}{\max\{f_{t',d}, t' \in d\}}$$

$$\text{IDF}(t,D) = \log\frac{|\{d \in D\}}{|\{d \in D : t \in d\}|}$$

with $f_{t,d}$ the number of time t appears in d. To perform this embedding in our project, we have used the `scikit-learn` implementation `TfidfVectorizer`.

`Word2vec` Model

As presented in [Tomas Mikolov and Dean,], the `Word2vec` model aims at learning continuous vector representations of words from a text dataset. The main advantages of this method are its capacity to capture similarity between words that goes beyond syntactic similarities (Example cited in [Tomas Mikolov and Dean,]: vector("King") - vector("Man") + vector("Woman") = vector("Queen")), and its capacity to escape the curse of dimensionality and to be trained on large datasets (Example cited in [Tomas Mikolov and Dean,]: the model has succeeded at producing words vectors of dimension between 50 and 100 by being trained on a dataset with more than a few hundred of millions of words). The `Word2vec` model uses a neural network architecture (one hidden projection layer) to learn the words vectors, and the minimisation of the loss depends on the model's

variant that has been chosen. There exists two variants (presented in section New Log-linear Models of [Tomas Mikolov and Dean,]):

- Continuous bag-of-words model (`CBOW`): the `CBOW` architecture aims at predicting a word representation based on its context (previous and next words).
- `Skip-gram` model: the `Skip-gram` architecture aims at predicting the representation of the surrounding words given the current word.

In order to use the obtained words representations in a document classification task, we create each document representation by summing all its words vectors.

7.2 Text Models

`TF-IDF` and `Logistic Regression` Model

As per the file `text_baseline.py` this model is the only one that does not use the graph structure of the data. However, it achieves great performances when performing an effective preprocessing and when carefully choosing the parameters of the methods `TfidfVectorizer` and `LogisticRegression` (from `scikit-learn`).

Parameters Optimization

In order to choose the best parameters, we used the function `gp_minimize` from the library `skopt`. It performs a bayesian optimization using gaussian processes. Instead of feeding the optimizer with lists of hyperparameters to test, the function's values are assumed to follow a multivariate gaussian. According to that, a smart choice to choose the next parameter to evaluate can be made by the acquisition function over the gaussian prior which is much quicker to evaluate. Therefore, we fed the optimizer with space dimensions that can be defined as:

- a (lower_bound, upper_bound) tuple (for Real or Integer dimensions);
- a list of categories (for Categorical dimensions);
- an instance of a Dimension object (Real, Integer or Categorical).

We forced some parameters prior to the optimization for `TfidfVectorizer`:

- decode_error = "ignore"
- sublinear_tf = True

The best score achieved was 1.232 with the following parameters:

- `TfidfVectorizer`:
  - max_df $= 0.9$
  - min_df $= 0.0149$
  - ngram_range $= (1, 1)$
  - binary = False
  - smooth_idf = True
- `LogisticRegression`:
  - tol $= 10e - 5$
  - C $= 4.6$

The `TF-IDF + Logistic Regression` model with those parameters constitutes our best performing model on the Kaggle public leaderboard. The multi-class loss achieved is 1.03771. It comes from the file `tfidf_text.py` in the `models` folder of our code. To run this model, open a terminal from the root directory of our code and run:

- `cd models`
- `python3 tfidf_text.py`

Data Augmentation Through Texts Separation

In this method, we wanted to try splitting each text into different small texts in order to gain train size. The idea was to get inspired from bagging methods in order to aggregate the classification over the subtexts onto the class for the complete text.

The texts were split according to the occurrence of a large horizontal line and very small resulting subtexts were removed to reduce noise. To keep the information about the original texts, we created a dictionary mapping each text to its subtexts.

We then managed to run a classification algorithm over the new data. The results were very bad with a loss of nearly 9.567. This can be explained by the fact that the labelling was done manually and not on the whole web domain. When enlarging our dataset, we would have had to manually label each subtext. Instead, we decided to transpose the parent's label to the subtexts, hoping that most of them would be from the same context of their parent text.

Despite that, and looking at the score obtained, it seems that the subtexts contained a lot of noisy data, which was decorrelated from the category of the parent text. Thus, we decided to focus our study on the original texts only, looking at improvements in the preprocessing and parameters optimization.

7.3 Text and Graph Models

`TF-IDF` and `Deepwalk`

In this model, we have tried to combine text and graph features. For texts, we have applied a `TF-IDF` vectorizer on the cleaned text data. For the graph, we have used a weighted version of the `Deepwalk` algorithm seen on Lab6 of the course and initially presented in [Bryan Perozzi and Skiena,]. In this weighted version, we take into consideration the fact that we work with a weighted directed graph by using the normalized weights of the edges related to a node as transition probabilities from this node to its neighbours. We then pass the generated walks to a `Skip-gram` model in order to generate vectors. We combine the text features extracted with `TF-IDF` and the graph features extracted with `Deepwalk` algorithm. Then we split the train data into 80% of train and 20% of evaluation data in order to evaluate our model.

For the following parameters values, we got the results shown in the results section on the evaluation set:

- `TfidfVectorizer`: we used the same parameters as the best performing model on text data and all other parameters were set to default values.
- `Deepwalk`:
  - n_dim $= 128$

  - n_walks = 500
  - walk_length = 100
- `LogisticRegression`: we used the same parameters as the best performing model on text data and all other parameters were set to default values.

`Word2vec` and `Deepwalk`

In this model, we have also tried to combine text and graph features. For the graph, we have applied the same algorithm as in the previous model (a weighted version of `Deepwalk`). For texts, we have applied a `Word2vec` based model on the cleaned text data. In this model, we have used the `gensim.models` implementation of `Word2vec` and the `Skip-gram` architecture. The training was performed on all the texts data. Then, the word embeddings were used to build the document features as follows: each document feature corresponds to the sum of the embeddings of all its words. We combined the texts features extracted with this `Word2vec` based model and the graph features extracted with `Deepwalk` algorithm. We then split the train data into 80% of train and 20% of evaluation data in order to evaluate our model. For the following parameters values we were able to get the results shown in the results section on the evaluation data:

- `Word2vec`:
  - size = 128
  - window = 8
  - min_count = 0
  - sg = 1
  - workers = 8
- `Deepwalk`:
  - n_dim = 128
  - n_walks = 100
  - walk_length = 200
- `LogisticRegression`: we used the same parameters as the best performing model on text data and all other parameters were set to default values.

The `Logistic Regression` model failed to converge even when we increased the number of iterations. We can explain the high performance of `TF-IDF` based models in comparison with the `Word2vec` based model by the fact that `Word2vec` models are more adapted for tasks where we need to capture words' contexts. While in domain classification tasks, capturing the occurrence of some words is sufficient to determine in which domain a text belongs.

Graph Neural Networks (GNN)

In this model, we have tried to combine texts and graph features. For texts, we have applied a `TF-IDF` vectorizer on the cleaned text data. For the graph, we have used a GNN for learning node representations and performing node classification. The goal was to predict the class labels of the nodes of the test set using information from both the graph structure (given by the adjacency matrix) and the attributes of the nodes (using the `TF-IDF` embedding of the texts). Here, we do not consider the full graph with the 28002 nodes but we use the subgraph of annotated nodes (train and test hosts). We implemented a GNN model that consists of 3 layers. The two first layers are message passing layers in which we pass the normalized adjacency matrix in order to capture the structure of the graph and to mix it with the node embeddings:

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

where $\tilde{A} = A + I$ and $\tilde{D}$ is a diagonal matrix such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. This normalization trick addresses numerical instabilities which may lead to exploding/vanishing gradients when used in a deep neural network model. The two message passing layers are followed by a fully-connected layer which makes use of the *softmax* function to produce a probability distribution over the class labels. For the `TfidfVectorizer` we used the same parameters as the best performing model on text data. We used the following parameters for the GNN:

- epochs = 200
- n_hidden_1 = 100
- n_hidden_2 = 200
- learning_rate = 0.05
- weight_decay = $1e-2$
- dropout_rate = 0.4

The results are shown in the results section on the evaluation set.

8 Results

Table 2 shows the results on our evaluation set of the models described above.

Model	Multi-class loss	Accuracy
TF-IDF + LGR	1.22	0.53
TF-IDF + DeepWalk	1.37	0.50
TF-IDF + GNN	1.70	0.35
Word2vec + DeepWalk	4.25	0.49

Table 2: Web domain classification results.

References

[Bryan Perozzi and Skiena,] Rami Al-Rfou Bryan Perozzi and Steven Skiena. Deepwalk: Online learning of social representations. *arXiv:1403.6652v2 [cs.SI] 27 Jun 2014*.

[Tomas Mikolov and Dean,] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781v3 [cs.CL] 7 Sep 2013*.