French web domain classification

Team GCM: Camille Cochet - Mithuran Gajendran - Hugo Mallet
Master 2 Data Science - Ecole Polytechnique

Abstract

This report is written in the context of the course Advanced Learning For Text and Graph Data (ALTEGRAD), under the supervision of Michalis Vazirgiannis. It concludes by a data challenge that allows to apply our lessons in a real environment. By reading this report, one will understand how we proceeded.

1 Introduction

The aim of this challenge is to soft-classify into domains (non-mutually exclusive categories) few websites from the test set by using the subgraph of the French web graph and/or the scrapped text files from each websites. In the following, we will describe the different components of our methods, how we used the available data to build algorithms. We will then explain how we tuned the different parameters of our model, and compare the results obtained with our different approaches.

2 Feature Engineering

2.1 Data Description

The dataset was generated by scrapping French websites' domain. This work was performed by the DaSciM team. The dataset consists of 2125 domains. We were provided 2 different types of data:

- A subgraph of the French web graph. It has 28 002 vertices and 319 498 weighted, directed edges. Nodes correspond to domain ids while edges represents the existence of at least one hyperlink from one page to another.
- A text file for each domain containing the scrapped text of all the pages of the domain. The text was extracted from the HTML source code of the pages. The provided texts contain a mix of different languages.

The challenge consists of classifying the domains in 8 categories:

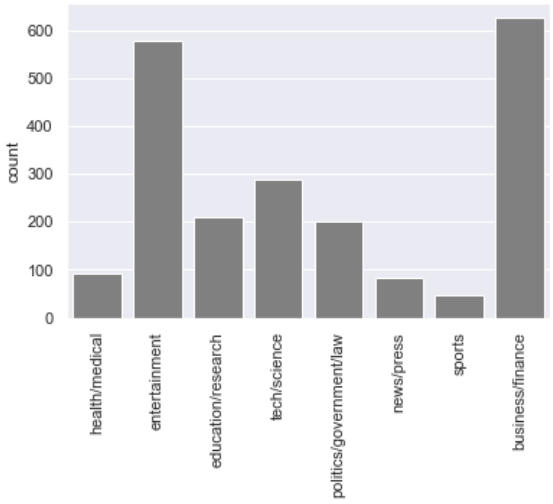

Figure 1: Labels of the 8 categories

2.2 Data Cleaning

After analysing the text files, we found out it was too noisy: white-spaces, special characters and so on. We cleaned them before applying our models with the following process:

- **Lowering**: Transform all characters to lowercase in order to make the training consistent
- **Tokenization**: Extract words from text while getting rid of white-spaces and punctuations
- **StopWords removing**: Remove French and English stopwords that have no semantic meaning
- **Translating**: Translating English words to French to make the training consistent
- **Stemming**: Stemming words to reduce tokens to their root form.

Further analysis of our dataset permitted to find new insights.

- **Closed domains**: some websites have been closed for some reasons. They contain unrelated words to the domain: "301 moved permanently, nginx". We looked at the text's lenght distributions and decided to put a threshold at 2500 characters. However, this method can lead to overfitting.

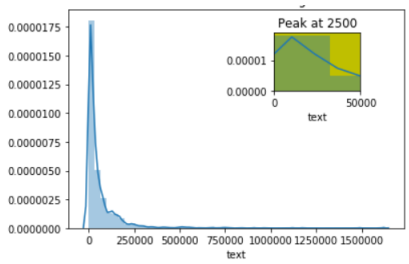

Figure 2: Distribution of text's length

- **Header and footer**: we realized that each website has a footer and header. They add some useless vocabulary that can be misleading for our algorithms. As a rule of thumb, we arbitrarily decided to delete the first quarter and last quarter of each corpus. This issue could have been avoided, if the HTML tags were in the corpus.

Note: This process needs to be adapted to the algorithms. Indeed, all the cleaning steps where not used in our experimental approach as algorithms needed a trade-off to tackle the over-fit/under-fit issue.

The preprocessing task gives the following results:

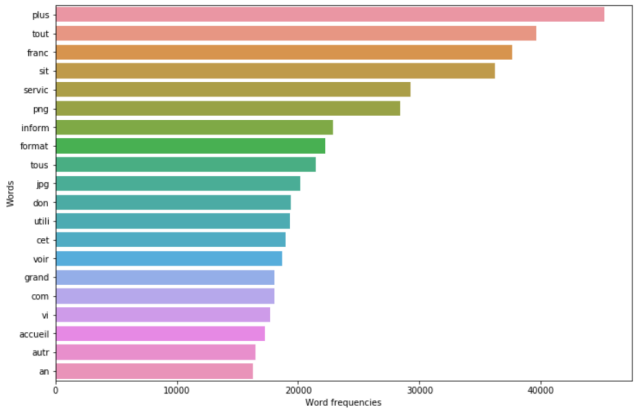

Figure 3: Common words after the preprocessing task

3 Features Extraction

While doing Natural Language Processing (NLP), the goal is to represent words by vectors as it contains numeric values that can be handled by computers.In the following, we will explain how we created our features. We used different methods.

3.1 Method 1: FastText algorithm

Word2Vec

The second method is based on Word Embedding which represents words by a dense vector and can capture semantic similarity between words. Word2Vec is a method to construct such an embedding. It can be obtained using two methods:

- **Continuous Bag-of-Words:** CBOW encoding follows this rule: given a context, which word is most likely to appear as a target.

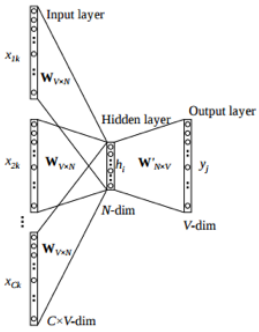

Figure 4: CBOW training mechanism

- **Skip-Gram :** Skip-gram's idea is to create encodings by finding a context from a target word.

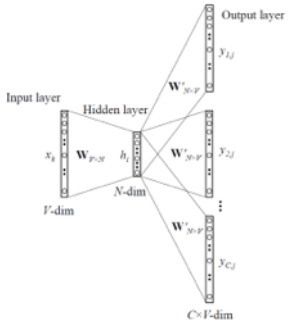

Figure 5: Skip-Gram training mechanism

However, after some trials with *Word2Vec*, we decided not to carry on our experimentation. *Word2Vec* faces a huge limitation: it cannot deal with unseen words. The mapping of an unseen word won't be correct.

FastText

We turned ourselves toward Facebook's solution: *FastText* (2016). It allows training word embeddings from a training corpus with the additional ability to obtain word vectors for out-of-vocabulary words. We enriched word vectors with subword called n-grams information. By adding up the vector representation of those decompositions, the model provides us the embedding of the initial word.

In other words, while being similar to the embeddings computed in the *Word2Vec*, *FastText* feeds the model with the words n-grams too. So the vector for a word is made of the sum of this character n grams. For example the word vector "apple" is a sum of the vectors of the n-grams "ap", "app", "appl", "apple", "ppl", "pple", "pple", "ple", "le".

In our case, this modelling is interesting because the structure of French language is specific. A word can be made out of suffixes, prefixes and its core (stem).

The consequences are the following:

- **Rare words**: are better represented in our model as they have more weight (more vectors describing them)
- **Unseen words**: do not raise errors, and it allows the model to compute their embeddings as the aggregate of the n-grams included in the word
- **Heavier and slower** training as the n-grams must be computed and aggregated

```
Entrée [381]: 1 model_ted.wv.similar_by_word("rhinopharyngyte")

Out[381]: [('larynx', 0.771564781665802),
           ('rhino', 0.7685766220092773),
           ('rhinos', 0.7628204822540283),
           ('hémophiles', 0.7276030778884888),
           ('rhinoceros', 0.7178475856781006),
           ('laryngé', 0.7020445466041565),
           ('lymphoma', 0.7009330987930298),
           ('lymphocytes', 0.6928580403327942),
           ('laryngectomie', 0.6905566453933716),
           ('rhinocéros', 0.6900511980056763)]
```

Figure 6: Computing the most similar words to an unseen word

How did we proceed ?

- **First approach** was slow and naïve. We trained our *FastText* model over all the words of the corpus before iterating over a text and calculating the similarities between a target word and the 8 distinct labels. Once the best one would have been chosen, we would replace the corpus's words by the label. Then, by applying TF-IDF we would have a frequency calculation of our corpus. This method did not work, giving us a score of 9.6 on the leaderboard. By replacing a word with its closest class, we omit to take into consideration the context of the word while a word has a different meaning depending of its context. By **hard-classifying** those words, it led to **big errors**.
- **Second approach** was more sophisticated. After having trained our *FastText* model, we replaced all the words of a corpus by their vectors and then calculated the corpus mean. Why? We intuitively thought that **the global meaning of a text is the average meaning of its words**. Then for classification we let a classifier decide the outcome of the corpus vector. Indeed, we considered each corpus vector as a "feature" and let the classifier infer the answer for us.

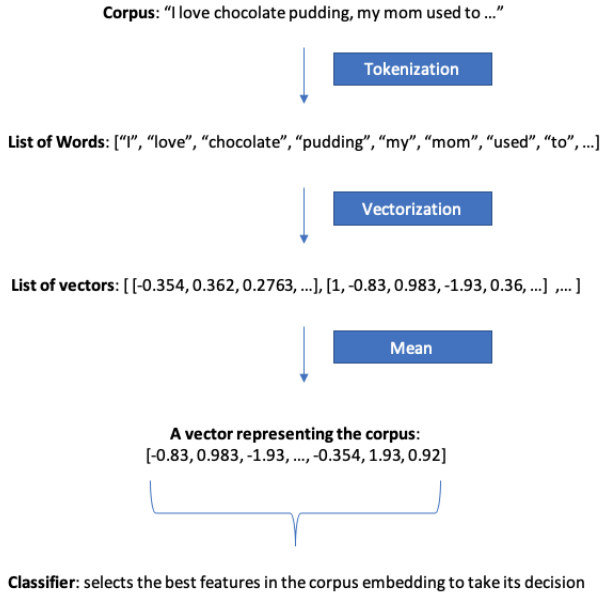

Figure 7: Corpus vectors mean approach

This approach led us to a score of **1.06035**.

Note: *FastText* is a *Word2Vec* extension. Therefore, data must be "properly cleaned". Syntax should be grammatically correct and one should not stem words, if so, n-gramm decomposition will be inefficient.

t-SNE Visualization

An easy way to verify that our model is effectively learning is to check whether its embedding representation makes sense. We used T-distributed Stochastic Neighbor Embedding (t-SNE). It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space.

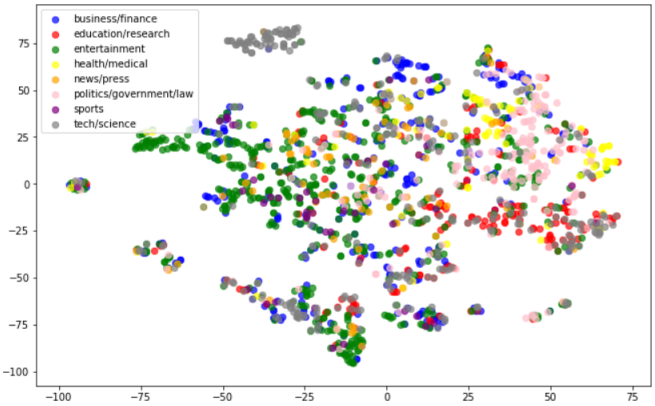

Figure 8: t-SNE visualization document embeddings

3.2 Method 2: TF-IDF

Our second model is based on *Term Frequency-Inverse Document Frequency* (TF-IDF) representation. It measures how important a word is to a document. This is done by multiplying two metrics:

- **Term frequency**: First step is to measure the frequency of a word in a document. This value depends on the document's length. However, the longer a document, the more frequent words should be. Hence, in order to avoid bias introduced by the document's length, we divide the frequency of the word by the number of words in the document. Thus, shorter documents have the same weight in decision than longer documents.

$$\text{TF(word, document)} = \frac{\text{count of word in document}}{\text{number of words in document}}$$

  From this operation, we obtain a vectorized form of our document.

- **Inverse document frequency**: Second step aims to measure how much information the word provides. In order to do so, we divide the count of document set by the the number of occurrences of a word in the document set. Finally, applying a logarithm allows us to smoothen the results for really large corpus.

$$\text{IDF(word, document)} = \log(\frac{\text{count of documents in a corpus}}{\text{number of documents where word appears+1}})$$

  Hence, a word appearing in a lot of documents in the corpus (such as stopwords) will give a low value of idf whereas less common words will bare more meaning for the document and give a high IDF value.

Combining those two metrics allows us to get the *TF-IDF* score. This method was a good start for our challenge and we could achieve good results thanks to it. However, some important parameters of the *TF-IDF* needed to be tuned in order to improve our score. In fact, the minimum and the maximum document frequency were decisive in building a meaningful vocabulary. Those parameters allow us to decide what are the minimum and maximum number of word occurrences for which we take the word into consideration for our vocabulary. Thus, it is common to get rid of words that have a really high frequency as they bare less meaning (like stopwords) and setting the maximum document frequency to 95%. Hence, words having a frequency higher than this threshold will be skipped. The same reasoning occurs while considering low occurring words. In fact, in most cases, very rare words (occuring once or twice) are not likely to be valuable for our understanding of the document.
However, during our experimentation, we've seen that our score increased a lot while setting the bounding thresholds to get a large vocabulary. Hence, setting the minimum document frequency to 1 word and the maximum document frequency at 99% showed good results. This means two things :

- In our corpus, low occurring words bare a lot of meaning and need to be taken into consideration.

- As we removed stopwords, many words that occurs at a high frequency are more likely to be meaningful. This can be especially true in the case of text coming from the web, where it is common to repeat "buzzwords" in different parts of the document (such as titles, head of paragraphs, lists...) in order to get a good SEO.

In an attempt to improve the relevancy of our *TF-IDF*, we decided to consider *bi-grams* in addition to simple words for our count. For example, in the sentence *"My cat doesn't like potatoes"*, bi-grams are "My cat", "cat doesn't", "doesn't like", "like potatoes". It is a way of introducing a context in our analysis which can be useful to discriminate the meaning of a sentence. However, this method wasn't really efficient and didn't improve our model. Moreover, considering each words and bi-grams for our *TF-IDF* multiplies the dimensionality of our resulting vector, which becomes much more difficult to process. Our hypothesis about the inefficiency of this technique is that a majority of web pages doesn't bare enough complex text and ambiguous sentences for the use of context to be truly useful.

Finally, we decided to use sublinear term-frequency scaling, thus replacing the original formula by : $1 + log(\text{tf})$. It aims to overcome the initial term frequency count "default" which considers that several occurrences of a term in a document truly carry several times the significance of a single occurrence. Applying the sublinearity improved our score, which can be due to the use of many irrelevant buzzwords in web pages.

Thus, this representation gives sparse vectors in high dimensionality and is unable to capture semantic similarity. This approach led us to a score of **1.02017**.

4 Conclusion

Surprisingly, although *FastText* is an advanced model, *TF-IDF* provides better score. It seems that tokens are more impactful than context vectors. Our hypothesis is simple: *FastText* works better with grammatically better written texts, yet in our case the text has been scrapped and is quite dirty. In short, one may ask: **Is it relevant to consider scrapped data as contextual data ?**

References

[1] A. Balsubramani and A. Ramdas. "sequential nonparametric testing with the law of the iterated logarithm". In *"Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence"*, page . 42–51, 2016.

[2] Yoav Goldberg and Omer Levy. "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method.". 2014.

[3] Quoc Le and Tomas Mikolov. "distributed representations of sentences and documents. in international conference on machine learning". page 1188–1196, 2014.

[4] Javed Shaikh. "https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a".

[5] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. "efficient estimation of word representations in vector space.". In *Statistics Journal Club,*, page arXiv preprint arXiv:1301.3781, 2013.

[6] Kai Chen Greg S Corrado Tomas Mikolov, Ilya Sutskever and Jeff Dean. "distributed representations of words and phrases and their compositionality. in advances in neural information processing systems". page 3111–3119, 2013.