PROJECT AlteGrad2019
French web domain classification

Samuel Guilluy[1] , **NGUYEN Tu Anh**[1] and **VU Thi Hai Yen**[2]
[1]Master MVA
[2]Master Data Science
Team name: Samuel - TuAnh - HaiYen

Abstract

This report introduces and studies the methods that we used in the AlteGrad 2019 Data Challenge on *French web domain classification*. We finally achieved the 3rd place and obtained a final score of **0.96314** on the public test set.

1 Introduction

In this project, we'll study the problem of web domain classification. Given a dataset of French websites/domains with the containing texts for each website along with a hyperlink graph, the objective is to well predict the category to which the domains belong. During this report, we'll first investigate the given dataset, then we'll further study the methods used for our submission model, finally we'll discuss some observations of the challenge and conclude.

2 Investigation on the data

The dataset contains of 28,002 domains of the web pages in France, each containing the texts extracted from all the pages of that domain. A web graph is also given where each node correspond to a domain and where each edge represents a hyperlink connection between two sites.

2.1 Overview of the dataset

The training data only consists of 2,125 labeled domain ids. Each domain is classified into one of the 8 categories: *education/research*, *business/finance*, *sports*, *tech/science*, *news/press*, *politics/government/law*, *health/medical*, *entertainment*. As seen in the figure 1, the distribution of the label is not equally distributed. As shown in the section 4.1, this has an impact on the implications that the evaluation with the Accuracy metric will not give the same results as with the Multi-class Loss.

2.2 Study of the text dataset

As our data are extracted directly from the html pages of the websites from French and English sites, the data are very noisy and a pre-process of the data is required.

The figure 3 represents the distribution of the words per class, we can notice that it is highly correlated to the distribution of the labels per class. Thus, we can assume that the number of word is equally distributed per label.

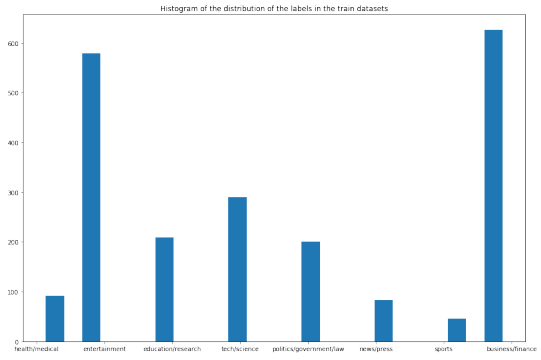

Figure 1: Histogram of the distribution of the labels in the train data set

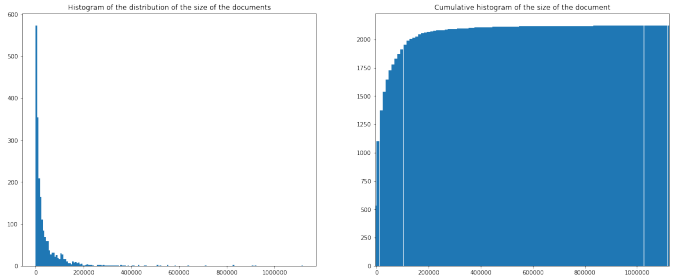

Figure 2: Histogram of the size of the text document (left) and cumsum (right) of the size of the text document

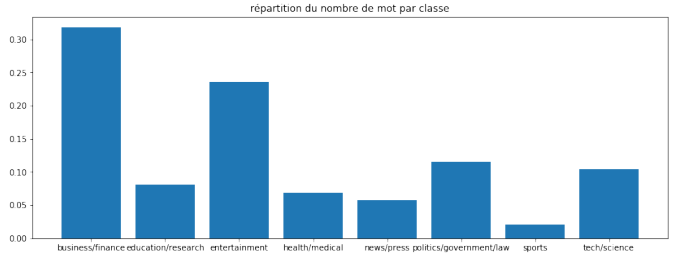

Figure 3: histogram of the number of word per class

	1	2	3	4	5
business/finance	repassage	bloctel	housse	vestes	détachées
education/research	inria	researchers	doctoral	anr	doctorales
entertainment	cupcakes	brunch	biscuit	saladier	macarons
health/medical	vasculaire	opératoire	cancérologie	dépistage	douleurs
news/press	trump	avalanche	commentés	commandant	gendarmes
politics/government/law	legifrance	relèvent	modernisation	pacs	recouvrement
sports	maillots	entraîneur	battu	wear	championnat
tech/science	geogebra	eula	soft32	utorrent	bittorrent

Figure 4: presentation of the top 5 words per class

We then tried to identify the most relevant word per labels. To do so, we introduce a metrics based on the tf-idf but for labels classification which measure the separability of the

$$\sum_{i \in Labels} \left(\frac{p_i * log\left(\frac{1}{g_i}\right)}{\sum_{j \in Labels} p_j * log\left(\frac{1}{g_j}\right)} - \frac{1}{8} \right)^2 \quad (1)$$

Where : p_i is normalized number of time we found the word p in the class i and g_i is the percentage of the total words which has the label i. Using this, we can extract the top 5 five words for each class in the figure 4.

Finally, we can conclude that the tf-idf methods is a good lead to pre-process the data. However our personal metrics is too selective as it keeps only 3.7% of the words present in the data set and thus don't achieve our best performance.

2.3 Study of the graph

The graph consists of 28,002 nodes and 319,498 weighted, directed edges connecting the domains. As there are only 1,994 different domains in the training dataset, one simple observation could be using only a sub graph extracted from the training set which contains 1,994 nodes and 5,067 edges. However, as we'll see in an instant, the surrounding nodes in the graph can be used to predict the label of a training node.

Study of the neighbors relation inside the graph

One intuition we have is that the websites from the same category are most likely to be neighbors. We check this hypotheses on the training set by counting the number of edges from a node of one class to a node of another class.

The figure 5 represents the normalized neighbors matrix for the 8 different classes. We can see that, except for the press class and the business/finance class, the other classes are mostly linked to other members of the same class. Thus we deduce that the labels of the surrounding nodes could contribute to the true label of a node and therefore can be used for prediction.

3 Model Improvement Process

We study in this section the methods we applied that actually improve the performance of the model, some other approaches (that did not work for this challenge) will be discussed in section 4.

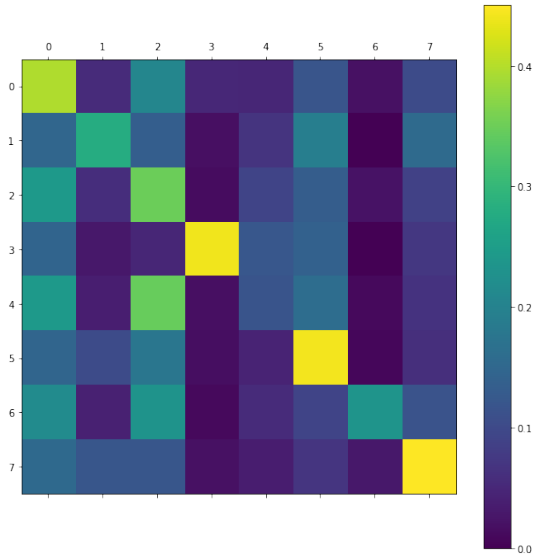

Figure 5: representation of the neighbors matrix normalize by the total number of edges from each class

3.1 Improve Text Baseline

We start by improving the text baseline as we found it contributes a crucial part to the final performance of the model.

Preprocessing Text Data

As we said earlier, the texts from the websites contain a lot of noise and therefore need to be cleaned.

We first remove all the special characters in the texts including **ˆ$@%&*#** /\ ><][ )({}**"-_'+=∼ |!?;:.,** and replace them by a `None` string ". We found it is more beneficial to replace them by " instead of a space ' ', this may come from the fact that many french words are composed of some of these characters, such as *ajourd'hui* or *grand-mère*, and therefore replace these characters with a space can lose the meaning of these words. We also found that it is beneficial not to remove numbers from the texts despite the fact that there are a lot of noisy numbers such as number of telephone in the corpus.

Then we remove among the remaining words all the stop words, ie. the common words that makes no sense to the sentence such as *to, the,...* or *les, de,...* in French. As the corpus contains both English and French text, we remove all the stop words in these two languages. In addition, we also remove words composed of less than 3 characters to get rid other noisy words in the corpus.

Fine-tuning the doc frequency range of tf-idf

We found that the range of the doc-frequency of the words to be taken can affect greatly to the results. We observe that the range $(10, 1000)$ in the text baseline is far from optimal. We find indeed that increasing the minimal doc frequency term is very efficient to improve the performance of the model, this indicates that words with low document frequency tend to be noisy words and therefore are harmful for the model. After fine-tuning the range, we found that the range $(50, 2000)$ works reasonably well.

Normalize the training data
Although the tf-idf vectorizer already normalizes the vectors with the ℓ_2 norm. We found that it is much better to normalize the tf-idf vectors with the `max` norm. This helps a lot in the training process.

Combining all together
We now give the experiment results when combining the methods above together. The results are reported on a train/validation split on the given training data (*with duplicates*) with a validation ratio of 0.1. This is not a normal way to deal with train/validation data as the validation also contains some duplicates in the training data. However, we'll explain why this is a good choice in section 4.2. We'll use a Logistic regression Model (with ℓ_2 penalization, *this is important*) as it tends to give very reasonable results for this little training data scenario (we'll discuss more about this in the section 4.3).

	train loss	train acc (%)	val loss	val acc (%)
text_baseline	0.946	75.41	1.329	49.72
+ preprocess	0.939	75.6	1.321	51.11
+ change df range	0.951	72.9	1.267	55.39
+ normalization	0.515	88.85	**1.228**	60.09

Table 1: Text model performance when adding some techniques

Table 1 shows the results obtained when combining the above techniques. We can see that there is just a slight improvement after preprocessing, however this step is very important and also helps to boost the results with other improvements afterwards. We see that the normalization helps a lot in the training process, while changing df range helps the model to better generalize. Finally, this model allows us to achieve a score of **1.125** for the test set on Kaggle.

3.2 Combination of graph and text

After predicting the class of all the element from the test sets by a only text method, we would like to combine these information with the one contained in the graph.

Combine the text predictions and the graph structure
As discussed in section 2.3, the labels of the neighbors can have a great influence on the final result. For this reason, we'll make use of the model obtained for the text. We first generate the prediction scores of all the nodes in the graph. Then, for each training node, we compute the weighted average prediction scores of the incoming and outgoing nodes. This results in 16 additional features for each node.

We'll combine these features with the tf-idf features received previously to pass through the classifier. Note that we'll not make use of the graph features as it is harmful to the final performance of the model (cf. section 4.4).

Dimensionality reduction for tf-idf vectors
The tf-idf vectors has 5,512 features, which is very huge comparing with the 16 features of the neighbors' scores obtained above. For this reason, we do a dimensionality reduction so that the size of the text features is comparable with the scores features. We use the latent semantic analysis (LSA) method ([2]) (which is known as `TruncatedSVD` in Sklearn) to reduce the dimension of the tf-idf vectors. We found that the best results are obtained for *n_components* around 120.

After reducing the dimension of the text vectors, we concatenate them with the 16 features computed above to give final prediction.

Combine what've done so far!
We now report the results obtained after the two methods above. Our train/valid split as well as the classifier is the same as described above. Results are shown in table 2.

	train loss	train acc (%)	val loss	val acc (%)
text_model	0.515	88.85	1.228	60.09
+ neighbor scores	0.468	89.74	1.17	58.68
+ dim reduction	0.905	66.37	**1.136**	55.86

Table 2: Combination of Text and Graph model

We can see that by adding only the scores of the neighborhoods, the loss decreases considerably. Using dimensionality reduction further improves this score a little bit, however the most important thing is that it makes the model less overfitting, which allows us to try with other complex models such as a feed-forward network in order to increase the performance of the model. We achieved a score of **1.039** on Kaggle using these techniques.

3.3 Other techniques to improve the result

Score-iterations for better result
As in the last section, we used the prediction score obtained from the text model, which may be less well than the model obtain when combining the graph and text models. We used this model again to reproduce a new prediction score for the neighborhood nodes. We finally obtained a new score of **1.003** on Kaggle. We found that further iteration does not improve the test score.

Using a simple MLP
As we have suggested, the use of Logistic Regression model is beneficial as it avoids overfitting too much. However, as we see in table 2 we have generalized the problem without increasing the validation loss, we can use a more complex model such as a Feed-Forward Neural Network. It permitted us to increase the train loss as well as the validation loss of the model.

However, we note that as we are so little data, using neural network can be very risky as the validation loss can be highly varied. After many tries, we achieved the best score of **0.963**, which was our best score for the challenge.

4 Other Discussions

This section aims to discuss some of our findings during this challenge and give some approaches that don't work well for the challenge (but might be very good for others). Our claims, however, may not be always true but we would like to give some ideas about the challenge.

4.1 Is the Loss always coherent with Accuracy?

The Multi-class Loss is a measurement of accuracy that incorporates the idea of probabilistic confidence

$$p_{ij} \longleftarrow min(max(p_{ij}, \epsilon)1 - \epsilon)$$
$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} log(p_{ij}) \quad (2)$$

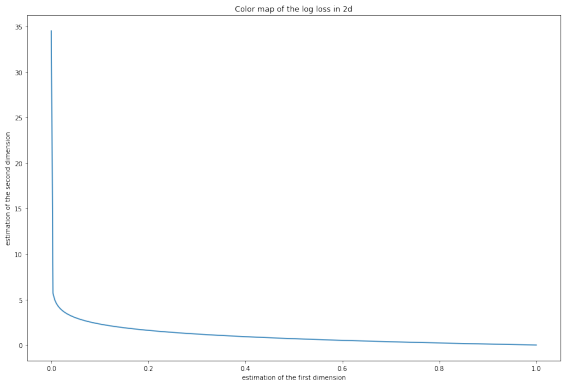

Figure 6: 2d representation of the log loss

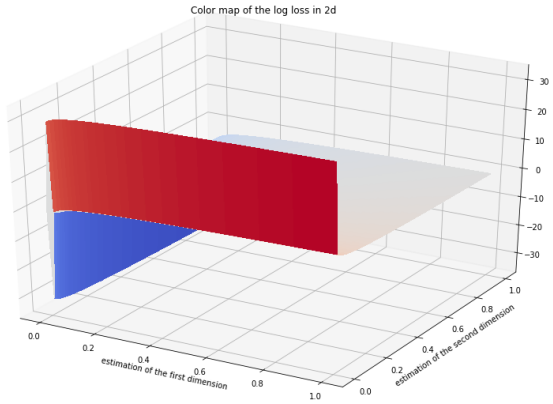

Figure 7: 3d representation of the log loss

where N is the number of samples, C is the number of classes, y_{ij} is 1 if sample i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability that sample i belongs to class j. And ϵ a constant in order to bound the divergence of the log.

Multi-class Loss heavily penalises classifiers that are confident about an incorrect classification. For example, if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large indeed.

As a consequence, if the model succeeds to correctly predict many labels but in contrast gives some confident wrong predictions. The loss would equally be very high too. In the case of the challenge, as the label is not equally distributed, the classifier might give very confident predictions for the common labels, resulting in high loss while still have a high accuracy. This also happens for models that over-fit a lot, and this is why over-fitting should be avoided.

There are at some approaches to dealing with this poor classifications (that we did not have time to try or tried but not working):

1. Examine the problematic observations relative to the full data set. Remove them from the data and re-train the classifier.
2. Smoothing the predicted probabilities using, for example, Laplace Smoothing. This will result in a less "certain" classifier and might improve the overall Log Loss.
3. Give some weights to the labels in order to make the classifier treat the label equally. In fact we have tried this with a weighted Logistic Regression classifier but apparently it does not seem to work well.

4.2 Why should we use duplicate data for validation?

We have reported earlier that the use of train/validation split for the data that contains duplicates results in some common data between validation and training data. This, normally, is not recommended as it favorites model over-fitting. However, we have found that for this challenge, it is better to do this way as the test set on Kaggle gives somewhat coherent results with the results on duplicated validation data.

There are some possible explanations for this behavior:

1. The duplicated train dataset contains for some domains several possible labels. As a result, if it performs well on the validation extracted from this dataset, the Multi-class Loss will tend to be better (see previous section).
2. The test set on Kaggle may be distributed as a combination of train and valid data. As a result, it's better that the validation data contains some training data as well.
3. The number of validation (and training) data is so little that the reported results are not always true and coherent with other test set (on Kaggle for example).

4.3 Is Deep always better?

As Deep learning achieves good results in many NLP tasks [4], we have tried to develop two networks :

1. Conv 1D network using a pre-trained embedding
2. LSTM network

Conv 1D network using a pre-trained embedding

Convolutional Neural Network works well for identifying simple patterns within data which will then be used to form more complex patterns within higher layers. [1]

A 1D CNN derives features from fixed-length segments of the overall data set and where the location of the feature within the segment is not of high relevance.

In addition to the CNN model, we use a pre-trained embedding. We select the linear structured embedding GloVe. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. [5]

The figure 8 presents the results of this methods. We can notice that the model over-fits after two epochs as the loss on the test set is not decreasing.

LSTM network

The LSTM network processes the sequence of word embeddings to capture long and short term distance dependencies within the sentence. [3]

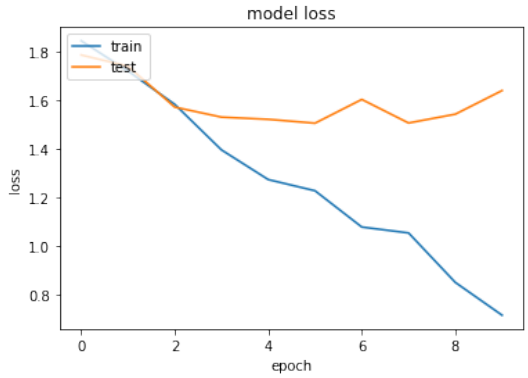

Figure 8: loss of the CONV 1D model through the epochs

The figure 9 presents the loss obtained for our LSTM model through the epochs. Here, we also notice the fast overfitting of the model after a few epochs.

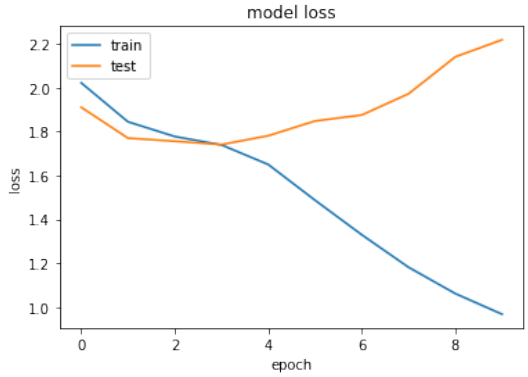

Figure 9: loss of the bidirectionnal LSTM model through the epochs

We can see that both the two methods overfit too early compared to the results obtained by the Logistic Regression reported in tables 1 and 2. We deduce that for this little data scenario, using deep methods is not always the best choice.

4.4 Does the graph features help to improve the model?

We tried to add the graph features to the final model obtained in table 2 and obtained the results as in table 3. We see that adding graph features are indeed harmful to both the training and validation data.

	train loss	train acc (%)	val loss	val acc (%)
text_graph	0.905	66.37	1.136	55.86
+ graph_features	1.573	43.98	1.649	42.25

Table 3: Model performance when adding graph node features

Thus, the use of graph features of the nodes is not favorable (or that we have not found the good features of the graph).

5 Conclusion

In this report, we have presented our model submitted to the Altegrad 2019 Data Challenge on *French web domain classification* as well as our findings on the dataset and the challenge itself. We hope that this helps to provide a better view on this data challenge.

References

[1] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 160–167, Helsinki, Finland, 2008. ACM Press.

[2] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, pages 391–407, 1990.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[4] Marc Moreno Lopez and Jugal Kalita. Deep Learning applied to NLP. *arXiv:1703.03091 [cs]*, March 2017. arXiv: 1703.03091.

[5] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.