Guessing the domain of a job from its description
Machine Learning for Natural Language Processing 2021

Kim Antunez
kim.antunez@ensae.fr

Alain Quartier-la-Tente
alain.quartierlatente@ensae.fr

Abstract

Using different NLP models (mainly LSTM and CamemBERT) we predicted jobs' domains using job descriptions provided by French governmental agencies.

1 Introduction: problem framing and datasets

Pôle Emploi is the French governmental agency which registers unemployed people and helps them find jobs and provides them with financial aid. To offer suitable jobs, they use a classification list of jobs called the "ROME" nomenclature[1].

For this project, we tried to answer to the following research question: Using Natural Language Processing modeling, is it possible to properly guess the domain of a job simply by reading its description?

To answer this question, we used two datasets provided by **Pôle Emploi** and web scraped additional information from **Onisep** (another governmental agency which provides students, parents and teachers information about jobs and training courses)[2]. The final dataset is composed of **1 247 descriptions** belonging to **14 different classes** (major domains)[3].

To predict the different major domains, we split the datasets into 2 sets (training and test sets) and performed **9 models of Sequence Classification** on the training set using `Pytorch` (see section 2). Once fit, we evaluated the model on the test set (unseen jobs and denominations) using F1 scores per class.

[1]A ROME code corresponds to a job and the 532 jobs of the database are classified in 110 professional domains ("domaine professionnel") and 14 major domains ("grands domaines"). Each job also contains many denominations (which are many names which refers to the same ROME code).

[2]More details about the datasets and the methods are provided in the .ipynb file on our github repository http://github.com/ARKEnsae/JobDomainPrediction_NLP.

[3]Given the limited size of the database used, we only tried here to guess the major domain classification.

2 Experiments Protocol : Embedding and Modeling

To perform Sequence Classification, we used different models, briefly described below.

2.1 Model 1 = Baseline: SVM on word2vec sentence-embeddings

For Model 1, we used pre-trained word-embeddings obtained from a French corpus[4] trained with a skip-gram word2vec approach and an embedding of size 200. It has the drawback to probably be a bit less performant than if we trained it ourselves in the specific domain of jobs, but we made this choice due to our few number of observations. In this simple model, each sentence-embedding is obtained by performing the average of word-embeddings of all words in the sentence. Finally, we use the supervised learning model *SVM* to perform data classification.

2.2 Models 2 to 7 = LSTM (with and without previous word-embedding)

For this second set of models, we used LSTMs, artificial RNN architectures with feedback connections allowing a contextual representation of tokens. A LSTM uses an embedding layer. For Models 2 and 3, this layer is initialized with random weights trained simultaneously with back-propagation as any other weights. Model 1 uses fixed-length inputs whereas Model 2 accepts variable-length descriptions of jobs and is more performant (but increases the training time).

Models 4 and 5 are respectively the same as models 2 and 3 but we initialized the embedding layer with the pretrained word2vec embedding used for Model 1. For Models 6 and 7, we do the same, but we freeze the updating of the word-embedding weights during the training.

We tried to improve the performance of Model 7 (best LSTM model) by tuning (Probst Philipp

[4]Available at https://fauconnier.github.io. The model was trained in the FrWac corpus of 1.6 billion of words.

and Boulesteix (2018)) learning rate which is known to be one of its most crucial hyperparameters (Greff Klaus and Jürgen (2016)). We observed that the performance varies a lot, but our initial hyperparameter were one of the best.

2.3 Models 8 and 9 = Fine-tuning the pre-trained CamemBERT transformer

For our last models, we use the famous bidirectional transformer BERT. The pre-trained CamemBERT (French equivalent) model can be easily fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, including sequence classification. For that, we use the `BertForSequenceClassification` function which uses the model transformer with a linear layer on top of the pooled output.

For Model 8, we used the same training and tests sets as previous models. For Model 9, we trained the model on the Onisep dataset and tested it on the Pôle Emploi dataset to see if there is a domain shift between the 2 datasets.

3 Results : Quantitative and Qualitative Evaluation

To evaluate the different models **quantitatively**, we used **F1-scores** per class and their global weighted and macro values (table 1). Indeed, it is more suitable than the accuracy when the dataset is not balanced (like for us) and to account for the weakness of our models in the classification of some specific (and rare) classes.

Model 8 (CamemBERT) is the best model followed by Model 7 (LSTM with frozen word2vec word-embeddings). Surprisingly, the baseline performs better than all the other LSTM models which means that using the words without their contexts in the description can show pretty good results in predicting the job domain. Furthermore, Model 9 shows also good results (figure 1) which means that the model does not seem to be very sensible to domain shift (descriptions from Onisep versus from Pôle Emploi).

In the notebook, we also evaluated our models **qualitatively**, giving some examples of good and bad predictions of our models.

4 Conclusion and discussion

To conclude, both LSTM (Model 7) and CamemBERT (Model 8) seem to be efficient in guessing the domain of a job from its description. Both methods are sensible to the context of the words in the descriptions, in contrast to our baseline, and it seems to improve the results even if the baseline was not that bad.

For the dataset we used, CamemBERT is the most efficient in term of weighted F1 score and Model 9 shows that it is still efficient even in presence of domain shift. Indeed, CamemBERT has the advantages of working at the sub-word units using byte-pair-encoding and thus has no Out-Of-Vocabulary problem. Thanks to the use of attention mechanisms, CamemBERT can also make some inputs more important than others; and whereas vanilla LSTM only uses left context, CamemBERT is bidirectional. However, it is also important to look at the **trade-off between performance and complexity**: our LSTM models are much faster to run and sometimes the use of a simpler model can be a good engineering option, even if it is a bit less efficient.

In the future, it could be interesting to **improve the LSTM implementation** we used, making it bidirectional and adding attention mechanisms, as raised in our NLP class. However, it could slow down the process and we must keep in mind that the recurrent nature of LSTM limits the possibility to **scale the training process to more data**: we cannot parallelize LSTM easily whereas it is easier with BERT's transformers.

In addition to job descriptions, we could use **additional information** to do some further experimentations and check if necessary diplomas or conditions for exercising the activities are useful to label the job's domain. Using **additional datasets** could also be interesting to improve the training process and to be able to predict job's domain at a less aggregated level. We can think of online CVs or LinkedIn job offers. However, collecting such data without the consent of their authors is more prone to **ethical concerns** than using job descriptions produced by specialized agencies and voluntarily put in open data.

Annexe

	Macro avg	**Weighted avg**
Model 1: Baseline	0.50	0.55
Model 2: LSTM (fixedL+RWE)	0.06	0.10
Model 3: LSTM (varL+RWE)	0.27	0.33
Model 4: LSTM (fixedL+W2V)	0.11	0.17
Model 5: LSTM (varL+ W2V)	0.36	0.40
Model 6: LSTM (fixedL+W2V+frozen)	0.11	0.14
Model 7: LSTM (varL+ W2V+frozen)	0.69	0.69
Model 8: CamemBERT	0.74	0.76
Model 9: CamemBERT with domain shift	0.57	0.62

Table 1: Macro and weighted F1 scores for our 9 models.
Note: The macro and weighted averages respectively correspond to the averages of F1 scores per class without and with considering the proportion for each label in the dataset. fixedL = fixed-length inputs / varL = variable-length inputs / RWE = random initialization of word-embeddings / W2V = word2vec word-embeddings / frozen = frozen word-embedding weights.

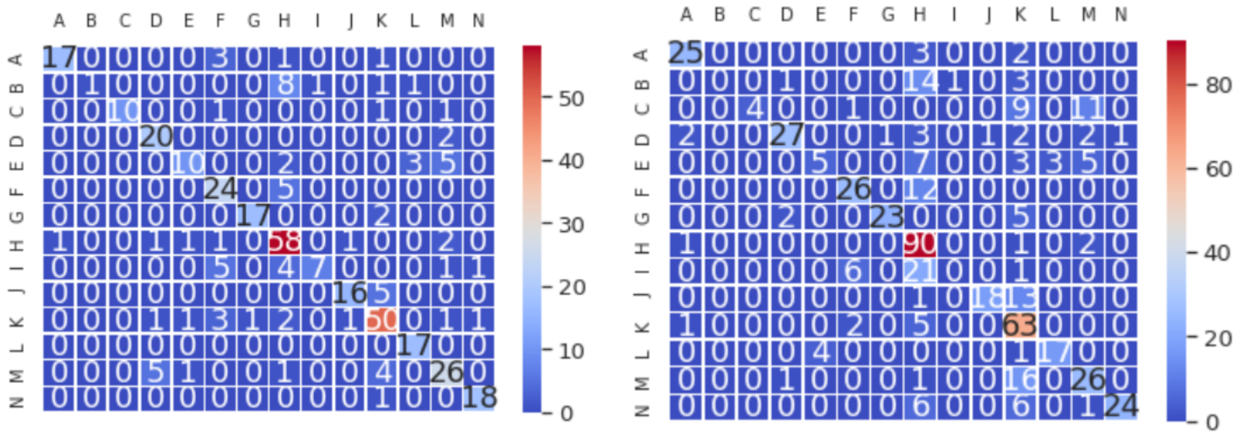

Figure 1: Confusion matrices for Models 8 (left) and 9 (right).

References

Koutnik Jan Steunebrink Bas R. Greff Klaus, Srivastava Rupesh K. and Schmidhuber Jürgen. 2016. Lstm: A search space odyssey. ieee transactions on neural networks and learning systems. *IEEE transactions on neural networks and learning systems*.

Bernd Bischl Probst Philipp and Anne-Laure Boulesteix. 2018. Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*.

https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df.

https://jovian.ai/aakanksha-ns/lstm-multiclass-text-classification.

https://towardsdatascience.com/multi-class-text-classification-with-deep-learning-using-bert-b59ca2f5c613.