Détection d'entités nommées géographiques par réseau de neurones récurrents

Lucas Anki, Léo Gaillard, Justine Revol

Détection d'entités nommées géographiques par réseau de neurones récurrents

Lucas Anki*, Léo Gaillard*, Justine Revol*

*Inist-CNRS (UAR 76), 2 rue Jean Zay, 54500 Vandoeuvre-lès-Nancy, France
[prénom].[nom]@inist.fr

Résumé. Cet article présente notre méthodologie pour répondre au défi TextMine 2024 (voir Guille (2023)). L'objectif est la **Reconnaissance d'entités géographiques dans un corpus des instructions nautiques**. Nous utilisons pour cela un réseau de neurones récurrents que nous construisons et entraînons en utilisant le framework **Flair** (Akbik et al. (2019)).

1 Introduction

Le groupe de travail TextMine a pour but de réunir des chercheurs et chercheuses sur la thématique large de la fouille de texte. Le défi TextMine naît d'une problématique spécifique à la fouille de texte (ici la détection d'entités nommées géographiques), la solution est apportée sous forme d'un défi, en mettant à disposition des jeux de données inédits à la communauté scientifique. C'est dans ce cadre que nous participons à l'édition 2024 : le problème est posé par l'Institut national de l'information géographique et forestière (IGN) et le Service hydrographique et océanographique de la Marine (Shom). L'objectif est la **reconnaissance d'entités géographiques dans un corpus des instructions nautiques**. Pour répondre à ce défi, les participants et participantes ont à leur disposition un corpus constitué d'extraits de 15 volumes des instructions nautiques annotés selon trois labels distincts : **name**, pour les noms propres purs ; **geogFeat**, pour les noms communs qui identifient une caractéristique géographique ainsi que **geogName**, pour les noms associés à une caractéristique géographique. Le jeu de donnée est constitué de 66030 tokens et de 18537 labels et chaque token peut avoir jusqu'à deux labels. L'ensemble de ces données est disponibles sur la plateforme Kaggle à ce lien [1]. Pour évaluer les scores, le modèle doit prédire les entités nommées sur le fichier *test.csv*, qui est non annoté, et les participants et participantes sont départagés en fonction de la micro f-mesure résultante.

2 Etat de l'art

La Reconnaissance d'entités nommées, une composante importante de la recherche d'information, a connu une évolution significative au fil des années. Depuis la sortie de *BERT* en 2019, les transformers fine-tunés pour de la détection d'entités nommées ont des performances plus que compétitives.

1. https://www.kaggle.com/competitions/defi-textmine-2024/data

Dans le cadre du défi TextMine, nous avons choisi d'utiliser un réseau de neurones récurrents (RNN) grâce à la bibliothèque **Flair**, ceci pour plusieurs raisons : encore aujourd'hui, les RNN sont toujours beaucoup utilisés pour faire de la détection d'entités nommées comme par exemple dans l'article de Smirnova et Mayr (2023) pour de la reconnaissance d'entités nommées sur des remerciements scientifiques ou encore dans l'article de Suignard et al. (2023) en réponse au défi TextMine 2023 (pour ne citer qu'eux). Dans ces deux cas, le framework **Flair** est utilisé et donne de bons résultats. De plus, en comparaison des transformers les RNN sont moins lourd pour une performance du même ordre sur cette tâche. On peut également noter que le framework **Flair** est encore maintenu : la dernière contribution apportée date de décembre 2023 contre décembre 2019 pour le projet **BERT-NER** [2].

En comparant avec d'autres framework fréquemment utilisés pour faire de la détection d'entités nommées, cet article de Vychegzhanin et Kotelnikov (2019) montre que **Flair** obtient souvent des meilleurs f-mesure.

L'ensemble de ces raisons ainsi que nos expériences passées nous ont montré que **Flair** est un framework adapté pour répondre à ce défi, pour ses performances ainsi que pour sa fiabilité. Le framework **Flair** est entièrement disponible sur GitHub, à ce lien [3].

3 Les données d'entraînement

3.1 Métriques sur les labels

Le jeu de données d'entrainement compte 39857 tokens, et chaque token peut avoir entre 0 et 2 étiquettes. Le tableau 1 la répartition des différents labels possibles. Nous pouvons observer que les classes sont déséquilibrées : il y a 41 fois plus de tokens dont le label est **aucun** que de token dont le label est **geogName**. Nous avons choisi de ne pas considérer la classe **aucun** à l'entraînement et d'attribuer ce label aux tokens n'en portant pas d'autres. En plus de régler le problème de déséquilibre entre les classes, ce procédé est naturel au vu de la signification logique du label.

Labels	nombres d'occurrences	fréquence d'occurence
aucun	32663	0.82
geogFeat	2704	0.07
geogName name	2117	0.05
geogFeat geogName	1463	0.04
geogName	910	0.02
geogFeat name	0	0
name	0	0

TAB. 1 – *Répartition des différentes étiquettes possibles dans le jeu d'entrainement train_2.csv.*

Après analyse des données, seules quatre étiquettes se retrouvent dans les données d'entraînement sur les six possibles au départ. Le label **geogFeat name** n'est pas possible à trouver,

2. https://github.com/kamalkraj/BERT-NER
3. https://github.com/flairNLP/flair

étant donné que **geogFeat** fait référence à des noms communs et **name** à des noms propres. Il est possible que le label **name** se retrouve seul dans les données d'évaluation, mais nous avons choisi de ne pas le détecter pour éviter le bruit et se concentrer uniquement sur des entités nommées géographiques : seul les **name geogName** seront détectés.

3.2 Formatage des données

Pour une tâche de détection d'entités nommées et sous version 0.10 (dernière version documentée en date de *juin 2023*), l'utilisation du framework **Flair** nécessite une structure particulière (voir Akbik et al. (2019))

Les règles de formatage sont les suivantes :

1. Trois fichiers textes sont constitués : *train.txt test.txt* et *dev.txt*.
2. Une ligne est constitué d'un unique token, et de son label, séparés par un espace ou une tabulation.
3. Une ligne vide sépare chaque phrase.

Le fichier *train.txt* contient les données d'entraînement. Après chaque époque, nous calculons la f-mesure sur un ensemble de données *dev.txt* et enfin à la fin de l'entraînement nous calculons la f-mesure ou toute autre métrique de qualité sur le corpus *test.txt*. Nous avons choisi une répartition assez usuelle des données à notre disposition : 0.7 / 0.15 / 0.15. La répartition entre les différents fichiers se fait aléatoirement : chaque phrase a une probabilité de 0.7 d'être dans le fichier *train.txt*, de 0.15 d'être dans le fichier *dev.txt* et le reste dans les données constituent le fichier *test.txt*.

4 Entraînement du modèle et utilisation de Flair

En adéquation avec la feuille de route du **CNRS** pour la science ouverte, nous mettons à disposition le code utilisé pour l'entraînement à ce lien [4].

4.1 L'embedding

Nous avons choisi d'utiliser un embedding contextuel de dimension 300, suggéré dans cet article Akbik et al. (2018) par **Flair** pour la majorité des cas d'usages : on utilise la fonction *StackedEmbeddings* de **Flair** pour combiner [5] un embedding **FastText** (Bojanowski et al. (2016)), un embedding contextuel prenant comme contexte le token juste après et un embedding contextuel prenant comme contexte le token juste avant. Chacun de ces embedding a été entraîné sur des données issues du Wikipédia français.

Dans notre optique de modèle léger, nous avons privilégié cet embedding contextuel plutôt qu'un embedding type transformers. En effet les embedding contextuels suggérés par **Flair** ne sont pas significativement différents pour des cas d'usages similaires d'après l'article de Brunner et al. (2020).

4. `https://github.com/Luc-Ank/textmine-2023`
5. Ces trois Embedding sont nommés dans le code respectivement *FlairEmbeddings('fr')*, *FlairEmbeddings('fr-forward')* et *FlairEmbeddings('fr-backward')*

4.2 Structure du modèle

Le réseau de neurone récurrent utilisé par notre modèle comporte trois couche :

— Une couche de reprojection de l'embedding. Dans notre cas la dimension n'est pas réduite et permet un simple ajustement des poids par un modèle linéaire lors de l'apprentissage.

— Deux couches type LSTM, chacune ayant un *word dropout* de 0.05.

4.3 Paramètres de l'entrainement

Pour information, les paramètres utilisés lors de l'entraînement du modèle sont écrits dans le tableau 2. Les paramètres non précisés sont les paramètres par défaut de **Flair** en version **0.10**. Avec le framework **Flair**, si une époque n'est pas une amélioration significative (sur un jeu distinct de l'entrainement), les poids du réseau sont inchangés. Nous souhaitions un grand *learning rate*, un grand nombre d'*époque* et une grande *patience* pour permettre le plus d'occurences possibles et éviter au maximum toute convergence de la loss vers un minimum local.

Paramètre	Valeur	Explicitation
learning_rate	0.2	qu'on abrègera *lr*
patience	10	réduction du *lr* après 10 mauvaises époques
mini_batch_size	12	petit batch size avec des LSTM
min_learning_rate	0.001	valeur par défaut
anneal_factor	0.8	facteur de réduction du *lr*

TAB. 2 – *Exemple de données d'entrainements pour utiliser **Flair**.*

5 Résultats

Afin d'optimiser les premiers résultats, nous avons utilisé la possibilité de faire de nombreuses soumissions à **Kaggle** pour augmenter artificiellement les données d'entraînement : nous n'avons alors plus besoin de fichier *test.txt*, la f-mesure obtenue lors de la soumission faisant office de résultat. Ainsi, les données présentes dans ce fichier peuvent être injectées dans les données d'entraînement : 0.85 pour le fichier *train.txt* et 0.15 pour le fichier *dev.txt*

La figure 1 montre l'évolution [6] de la loss calculée après chaque époque, calculée sur les données issues du fichier *dev.txt*. L'allure de la courbe ne montre ni un sur-apprentissage ni un sous-apprentissage.

La meilleure micro f-mesure obtenue sur le fichier *test.tsv* est de **0.97925**. Nous détaillons les métriques pour chaque label dans le tableau 3. Cependant, ces métriques sont calculées sur le jeu de donnée *dev.txt* étant donné que nous avons pris le parti d'utiliser le jeu de donnée d'évaluation du concours TextMine 2024, *test.tsv* comme unique évaluation de la qualité du modèle..

6. Pour modéliser des données discrètes par une courbe, nous avons utilisé une interpolation ce qui explique que la loss semble remonter légèrement lors de certaines époques, ce qui n'est pas possible avec la méthode employée.

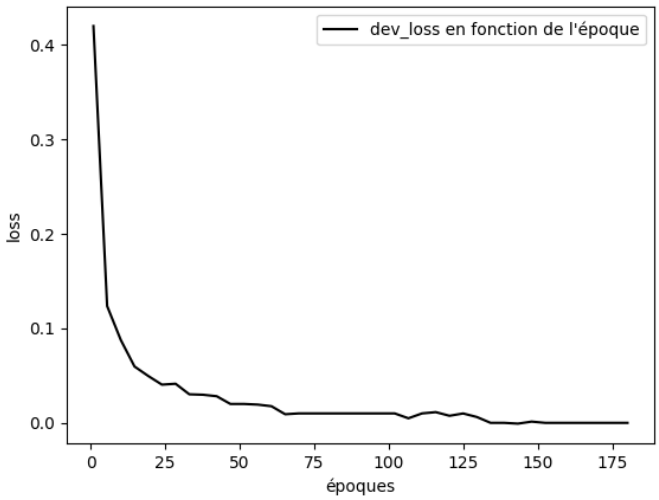

FIG. 1 – *Evolution de la loss lors de l'entrainement du modèle pour le fichier 'dev.txt' en fonction des époques.*

Labels	precision	recall	f1-score
geogFeat	0.8986	0.9538	0.9254
geogName name	0.9737	1.0000	0.9867
geogFeat geogName	1.0000	0.8333	0.9091
geogName	1.0000	0.8667	0.9286

TAB. 3 – *Métriques obtenues sur le fichier "dev.txt" pour chacun des labels à la fin de l'entrainement*

6 Conclusion et perspective

En conclusion, les résultats de notre modèle sont plus que satisfaisant et ils peuvent être aiséments reproduits ou modifiés en utilisant le code à disposition sur **Github** et les données sur **Kaggle**, même sur un ordinateur basique.

Nous n'avons pas beaucoup exploré la question de la dimension de l'embedding : en partant du principe qu'il y a moins de vocabulaire dans des corpus d'instructions nautique, ou tout autre corpus spécialisé, que dans Wikipédia il pourrait être intéressant de réduire la dimension sur laquelle l'embedding est projeté. Il peut être également intéressant de comparer ces résultats avec ceux que nous aurions pu obtenir en utilisant un LLM pour faire l'embedding, voir en utilisant la structure totale d'un modèle type transformers pour réaliser ce défi.

Références

Akbik, A., T. Bergmann, D. Blythe, K. Rasul, S. Schweter, et R. Vollgraf (2019). FLAIR : An easy-to-use framework for state-of-the-art NLP. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 54–59.

Akbik, A., D. Blythe, et R. Vollgraf (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pp. 1638–1649.

Bojanowski, P., E. Grave, A. Joulin, et T. Mikolov (2016). Enriching word vectors with subword information. *arXiv preprint arXiv :1607.04606*.

Brunner, A., N. D. T. Tu, L. Weimer, et F. Jannidis (2020). To bert or not to bert - comparing contextual embeddings in a deep learning architecture for the automatic recognition of four types of speech, thought and writing representation. In *SwissText/KONVENS*.

Guille, A. (2023). Défi textmine 2024.

Smirnova, N. et P. Mayr (2023). Embedding models for supervised automatic extraction and classification of named entities in scientific acknowledgements. *Scientometrics*, 1–25.

Suignard, P., L. Hassani, et M. Bothua (EasyChair, 2023). Participation d'edf rd au défi textmine 2023 : Reconnaissance d'entités d'intérêts dans les signatures d'e-mails. EasyChair Preprint no. 10098.

Vychegzhanin, S. et E. Kotelnikov (2019). Comparison of named entity recognition tools applied to news articles. pp. 72–77.

Summary

This article presents the methodology we used for the TextMine 2024 challenge. The objective is the **Recognition of geographical entities in a corpus of nautical instructions**. We built and trained a RNN with the **Flair** framework.