Transformers and GNNs for Fake News Detection

Stefano Bocconi[1,*], Alessandro Patruno[1] and Andrey Malakhov[1]

[1]Zephyros Solution, Netherlands

Abstract

This paper describes Zephyros team's submission for MediaEval 22 Fake News Detection. Our approach is based on transformers for text analysis and Graph Neural Network for structure analysis. The latter proved to be challenging and more research is required to achieve good classification results. This reflects also on task 3, where the structure analysis did not succeed to contribute positively to the outcomes. Nevertheless, we expect that eventually research will find a method of combining different sources of knowledge that results in better performances.

1. Introduction

In this paper we describe the approach followed by the Zephyros team in tackling each of the proposed tasks in the MediaEval 22 Fake News Detection task (described in [1]). The rationale of our approach follows the sequence of the tasks: we first tried to perform classification only using non-relational text information contained in the tweets (task 1), we then attempted to extract information from the graph of relations between the authors of the tweets (task 2), and finally we tried to combine these two approaches to tackle classification using both these sources of knowledge. Given task 1 seemed like the foundation to proceed further to the other tasks, considerable time has gone in experimenting with task 1. Our approach for this task has been based on transformers to obtain embeddings for the tweet texts, and on a feed-forward Neural Network to train for the classification task. We discuss this in Section 3. Once results for this task were deemed satisfactory, we focused on task 2. This task had two main challenges: how to deal with the considerable size of the directed graph made of relations between users, and how to translate the information provided for each user in a form that could be efficiently used in the computation. Most of the effort went into the former, as we describe in Section 4. Finally, we combined both approaches for task 3, with the assumption that providing more knowledge (two sources of information, structure-based and text-based) would result in better performance for the classification. This seems not to be the case, as we discuss in Section 5, although this also might be due to the fact that we had limited time left to try different approaches. Nonetheless, we think that the challenge represented by task 3, namely combining different sources of knowledge, holds a big potential and we plan to concentrate on that in the future. We first start with introducing the tools and methods we have used in the following section.

MediaEval'22: Multimedia Evaluation Workshop, January 13–15, 2023, Bergen, Norway and Online

*Corresponding author.
stefano@zephyros.solutions (S. Bocconi); alex@zephyros.solutions (A. Patruno); andrey@zephyros.solutions (A. Malakhov)

2. Related Work

In classifying texts our approach relies on the NLP capabilities recently exhibited by Transformers in general, and in particular by BERT [2]. The latter and many other models have been made widely available by the Hugging Face library[1].

Other related approaches in text analysis would have been worth trying, had we had the time. Firstly, the classical term frequency–inverse document frequency (TF/IDF). Furthermore, simple language models such as Unigrams and N-grams[2] could have provided a baseline to compare with the performances of transformers (which are also language models). On the other hand, using transformers such as BERT one needs to transform the single-word embeddings into sentence embedding for classification (as it will be described further). There are also approaches such as Sentence-BERT [3] that provide sentence embeddings as the output of the transformer.

Regarding the analysis of graph-structured data (such as in task 2) our work is strongly related to recent research and efforts in the domain of Graph Neural Networks, or GNNs ([4] and [5] provide good overviews of the subject). GNNs have been used in node, link and graph classification tasks, and have proven able to capture the influence of neighbouring nodes on a given node. Further, increasing the number of levels in the networks increases the diameter of the neighbourhood that has influence. We use one of the two popular libraries for GNNs, DGL[3], the other being PyG[4].

Finally, the work of other teams in previous MediaEval Fake News challenges is very related to our work, as described in [6] and [7].

3. Text-Based Misinformation and Conspiracies Detection

As mentioned in the previous sections, our work on task 1 has been based on transformers. We chose a model architecture composed by the encoder of the transformer, feeding into a feed-forward neural network in order to perform the classification. For the former we have tried several transformers that have been trained on datasets we deemed to be similar to the challenge's dataset. These were:

- BERTweet [8], which was trained on 850M English Tweets, containing 845M Tweets from 01/2012 to 08/2019 and 5M Tweets related to the COVID-19 pandemic, with its variants 'vinai/bertweet-base', 'vinai/bertweet-covid19-base-cased', 'vinai/bertweet-covid19-base-uncased', and 'vinai/bertweet-large'[5].
- COVID-Twitter-BERT v2 [9], which was trained on a large corpus of Twitter messages on the topic of COVID-19. The v2 model[6] is trained on 97M tweets (1.2B training examples).
- As a reference we also used a transformer (Roberta [10]) not specifically trained on tweets. This 'roberta-base'[7] transformer performed consistently worse than the previous ones.

Given transformers provide an embedding for each word (of the tweet in our case), we had to "merge" these embeddings into a single one to be used for the classification of the whole

[1]https://huggingface.co/
[2]https://en.wikipedia.org/wiki/Language_model
[3]https://www.dgl.ai/
[4]https://github.com/pyg-team/pytorch_geometric
[5]Respectively, https://huggingface.co/vinai/bertweet-base, https://huggingface.co/vinai/bertweet-covid19-base-cased, https://huggingface.co/vinai/bertweet-covid19-base-uncased and https://huggingface.co/vinai/bertweet-large
[6]https://huggingface.co/digitalepidemiologylab/covid-twitter-bert-v2
[7]https://huggingface.co/roberta-base

tweet. In the original BERT-paper [2] it is suggested to use the "pooler" embedding: "The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.". We also experimented with: 1) taking the mean of the last layer, 2) the mean of the second last layer, 3) summing the last 4 layers and taking their mean, 4) summing the last 4 layers and taking only the [CLS] embedding, 5) concatenating the last 4 layers and taking their mean, and 6) concatenating the last 4 layers and taking only the [CLS] embedding.

The feed-forward classification NN was composed by two linear layers with Tanh in between, 2 dropout levels before the linear levels and a non-linear function, which was a Sigmoid for binary output and a Softmax for multi-class outputs. Ideally, when using a pre-trained transformer, one should fine-tune its performances by retraining with the task's dataset. This would have implied in our case to propagate the gradient from the classification NN to the transformer. As this was computationally very time-consuming, we chose to use the embeddings from the transformer as input to the classification NN and only train the latter.

We tested 2 classification approaches: the first (called one-step) was to classify each tweet separately for each of the conspiracy classes, yielding 9 outcomes per tweet, each of value in {1,2,3} (values as described in [1]). This is equivalent to 9 independent multi-class classification problems. The second (called two-steps) was based on the assumption that each conspiracy class could be correlated to other conspiracy classes in the same tweet. We therefore tried first to detect what classes were present in a tweet (thus class present {2,3} versus class not present {1}), and then decide whether each tweet was discussing {2} or promoting {3} the classes present in it. The first problem was a multi-label problem, while the second one was a binary classification. Naturally this approach disregards the fact that there are tweets that promote one or more conspiracy classes and discuss others, but given there are only 16 tweets out of 1913 that do so in the training dataset we chose to try anyway.

In this and the other tasks we used 5 different seeds and early stopping in case of no improvement in the test loss for 7 consecutive epochs. We used 5 stratified splits doing cross-validation of the different approaches. In case of the multi-label problem in the two-steps approach, stratification is not trivial and we chose the Iterative Stratification method described in [11].

Finally, we also tried to introduce a "Cannot Determine" {0} class to see whether the performances in cross-validation would improve. We did so by choosing an optimal threshold (with respect to the MCC metrics) in the training split and evaluating it on the test split.

The results from the cross-validation on the training set are as follows: for both one-step and two-steps approaches, the best transformer is "digitalepidemiologylab_covid-twitter-bert-v2", and the best output layer is the mean of the concatenation of the last 4 layers. The one-step approach had consistently better cross-validation results than the two-steps approach, therefore we only submitted results from that. Our best submission had an optimal threshold only for classes "Intentional Pandemic" and "Population reduction Control", and fixed for all the others. It achieved a train MCC = 0.9505 and test MCC = 0.5958.

4. Graph-Based Conspiracy Source Detection

As mentioned before, we used GNNs to tackle task 2. In order to do so, the first step was to create a graph where the nodes were represented by the users, and the edges the relation between them. This graph was directed as the links in the dataset were assumed to be directed. The obvious choice here was to include all links between users (['complete']), but given the considerable size of the resulting graph, we also experimented with some methods to reduce the edges. We chose the following variants: ['direct'] includes only edges between users that belong to training or test sets, ’2hops’ includes all direct or max-one-node-in-between edges between users that belong to training or test set, ’1radius’ includes all in-edges and out-edges for users that belong to training or test set. Due to filtering edges and the fact that the graph is directed, the end result might have zero-in-degree nodes, i.e. nodes that do not have incoming edges, which is a problem for GNN as these nodes have undefined features (they cannot be the target of the feature propagation from other nodes). To solve this we adopted 2 strategies, make the graph undirected and add self-loops.

The second problem was to translate the provided user information into embeddings to include as node features in the graph. Most of the user info was already in numeric format, with the exception of ‘creation_date‘ that we converted to a timestamp (ms from January 1, 1970), and ’location_country’, ’location_state’ and ’location_city’ that were hashed. Since we planned to use these embeddings together with the results of task 1, we scaled each value of the above mentioned node feature in the same range as the embeddings provided by the transformers in task 1. Beside this method, given the fact that we were not too confident that these features were meaningful, we also adopted a random features approach.

We then experimented with several Convolutional GNNs, and different numbers ([3-10]) of convolution layers . The best results were given by SAGEConv[8] and GraphConv[9].

The submission that achieved the highest score was for a ’direct’ graph with self-loops, initial random features, 10 layers with SAGEConv using pool as aggregator function, and no ’Cannot Determine’ class. MCC in training was 0.0758 and in test 0.1111.

5. Graph and Text-Based Conspiracy Detection

For task 3 we decided to combine the two previous approaches by concatenating the embedding for each tweet (task 1) with the embedding of the user that authored it (task 2). Initially we tried to balance the dimension of the embedding, given that dim(emb_task1) = 4096 was much bigger than dim(emb_task2), by increasing the size of emb_task2. Unfortunately this was computationally too expensive and we were forced to have dim(emb_task2) = 10. This might be the reason that with concatenation the results were worse than with just the tweet embeddings as in task 1. MCC in training was 0.6894 and in test 0.2460. A better approach might have been to feed the two different embeddings to two different NN and combine their result with a third NN, but we had no time to experiment with this.

Using concatenation we noticed an improvement when increasing epochs, and we thought that it may be due to the fact that the network learns to ignore the user embeddings.

6. Discussion and Outlook

Several future directions would be worth investigating. For task 1 a multi-class {1,2,3} and multi-label (multiple conspiracy classes at the same time) approach would be worth investigating. Alternatively, it might be worth investigating whether each tweet can be decomposed in sentences, each getting their label, so that the ground-truth becomes more specific. According to the task rules, this cannot be achieved using external knowledge, though. For task 2 there should be a better way to translate user information in meaningful embeddings, where the performances increase with respect to random features. As already mentioned, task 3 is the most interesting one as it uses different sources of information, but this needs a method to effectively translate this potentially more complete knowledge into better classification performances.

[8]https://docs.dgl.ai/en/latest/generated/dgl.nn.pytorch.conv.SAGEConv.html
[9]https://docs.dgl.ai/en/latest/generated/dgl.nn.pytorch.conv.GraphConv.html

References

[1] K. Pogorelov, D. T. Schroeder, S. Brenner, A. Maulana, J. Langguth, Combining tweets and connections graph for fakenews detection at mediaeval 2022, in: MediaEval'22: Multimedia Evaluation Workshop, 2023.

[2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL: https://arxiv.org/abs/1810.04805. doi:10.48550/ARXIV.1810.04805.

[3] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL: https://arxiv.org/abs/1908.10084. doi:10.48550/ARXIV.1908.10084.

[4] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph Neural Networks: A Review of Methods and Applications (2018) 1–22. URL: http://arxiv.org/abs/1812.08434. arXiv:1812.08434.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, arXiv (2019) 1–22. URL: https://arxiv.org/abs/1901.00596v4. arXiv:arXiv:1901.00596v4.

[6] M. G. Constantin, S. Hicks, M. Larson (Eds.), Working Notes Proceedings of the MediaEval 2021 Workshop, Online, 2021. URL: https://2021.multimediaeval.com/.

[7] S. Hicks, D. Jha, K. Pogorelov, A. G. S. D. Herrera, D. Bogdanov, P.-E. Martin, S. Andreadis, M.-S. Dao, Z. Liu, J. Vargas-Quirós, B. Kille, M. Larson (Eds.), Working Notes Proceedings of the MediaEval 2020 Workshop, volume Vol-2882, CEUR, Online, 2020. URL: https://2021.multimediaeval.com/.

[8] D. Q. Nguyen, T. Vu, A. T. Nguyen, BERTweet: A pre-trained language model for English Tweets, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, pp. 9–14.

[9] M. Müller, M. Salathé, P. E. Kummervold, Covid-twitter-bert: A natural language processing model to analyse covid-19 content on twitter, arXiv preprint arXiv:2005.07503 (2020).

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, 2019. URL: https://arxiv.org/abs/1907.11692. doi:10.48550/ARXIV.1907.11692.

[11] K. Sechidis, G. Tsoumakas, I. Vlahavas, On the stratification of multi-label data, in: D. Gunopulos, T. Hofmann, D. Malerba, M. Vazirgiannis (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 145–158.