UAICS at CheckThat! 2020: Fact-checking claim prioritization

Ciprian-Gabriel Cusmuliuc, Lucia-Georgiana Coca, Adrian Iftene

"Alexandru Ioan Cuza" University, Faculty of Computer Science, Iasi, Romania

{cusmuliuc.ciprian.gabriel, coca.lucia.georgiana, adiftene}@info.uaic.ro

Abstract. Claim proving can be an incredibly challenging task considering the amount of information in the world increases day by day. Journalists and people alike spend a lot of time investigating claims and fact-checking different statements. In order to address this problem CLEF 2020 CheckThat! proposes 5 tasks that each present a different side of the problem. Our team participated in Task 1 and Task 5 which aim to rank statements by check-worthiness. For Task 5, we proposed 3 methods, each based on a different machine learning algorithms, Naïve Bayes, Logistic Regression, and Decision Tree. For Task 1, we created a system based on BERT. For Task 5, the best result we achieved using the official measure MAP was with the Naive Bayes. This paper presents the details and results of our approaches.

Keywords: Naive Bayes, BERT, Logistic Regression, Decision Tree

1 Introduction

Increase in social network popularity has led users to conduct multiple activities on them, such as message exchanging, posting, news reading, commenting, and so on. Instant sharing and broadcasting have enabled users fast and a vast access to information, but all with a cost. The problem arises that news propagation in these platforms is becoming uncontrollable, users frequently read and share information without checking the veracity of a certain claim, leading to misinformation.

The problem of needing to check the news and claims in these networks is slowly becoming of extremely high interest to major players, such as Facebook and Twitter, as recently they are having talks of integrating such tools in their systems, one such example is Twitter fact checking Donald Trump and labeling his tweets as 'manipulated media'[1] sparking outrage amongst its supporters. This problem is not entirely present

[1] https://www.bbc.com/news/technology-53106029

in social media, we can see an effort to spread misinformation and propaganda on the entire internet.

CLEF 2020 CheckThat! [1][19] is an evaluation campaign that is being organized as part of CLEF 2020 [2] and contains 5 tasks, each related to fact-checking. Our team participated in two tasks, Tasks 1 and 5.

Task 1 requires the development of a system capable of ranking a stream of potentially-related tweets according to their check-worthiness; this task ran in English and Arabic, we participated only in the English version by developing a model based on BERT [3], a bidirectional transformer developed by Google with exceptional performance.

Task 5 has the objective of identifying which sentences from a political debate should be prioritized for fact-checking. In this task, we submitted 3 models based on Naive Bayes, Logistic Regression and Decision Tree.

This paper describes the participation of team UAICS, from the Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi, in Task 1 and 5 at CLEF 2020. The remaining of this paper was organized as follows: Section 2 describes the state of the art, Section 3 gives a description of the tasks. Section 4 details the model we developed and the submitted runs and then Section 5 details the results we obtained, finally Section 5 concludes this paper and presents future work.

2 State of the art

Previous editions of CheckThat! i.e. 2019 and 2018 had fewer tasks; Task 1 was based on the same claim prioritization as task 5 whilst task 2 required to assess which web pages can be useful in human fact-checking and consisted of multiple subtasks. We will only be referring to task 1 as this task is also present in 2020.

In 2019 approaches for task 1 were very various, the best team "Copenhagen" [4] had a MAP score of .1660 and the system was based on learning dual token embeddings in conjunction with an LSTM [5]. The network has been pre-trained using previous Trump and Clinton debates while supervising it with ClaimBuster[2]. Other approaches are the following (in order of the ranking): team TheEarthIsFlat [6] used a feed-forward neural network with two hidden layers, team IPIPAN [7] used an L1-regularized logistic regression, team Terrier [8] used SVM [9] in conjunction with bag-of-words and named entities and team UAICS [10] used a Naïve Bayes classifier with bag-of-words features.

In 2018 the best team was still "Copenhagen" [11] with the lowest MAE of .7050, they used a similar approach as in 2019 being a convolutional neural network [12] and support vector machine. Other approaches range from random forests, logistic regression, and LSTM.

[2] https://idir.uta.edu/claimbuster/

3 Tasks description

In 2020 there have been 5 tasks that ran in English and Arabic, we only participated in 2 of them, Tasks 1 and 5. In this section, we will shortly present the two tasks we took part in.

Task 1 requires "*given a topic and a stream of potentially-related tweets, rank the tweets according to their check-worthiness for the topic*". This task runs in English and Arabic.

Task 5 requires "*given a political debate or a transcribed speech, segmented into sentences, with speakers annotated, identify which sentence should be prioritized for fact-checking*". This task is only in English.

3.1 Evaluation metric

Both tasks use MAP [13] as the official metric which calculates the usual mean of the average precision. Other measures used are the Mean Reciprocal Rank [14] which allows obtaining reciprocals of the rank of the first relevant document, as well as Mean Precision at k, which performs the average of k best candidates. Details on the measures used can be found in the task overview [1].

Evaluations are carried out on primary and contrastive runs. Each participant has the right to three models, one primary and two secondary (contrastive). We tried to take advantage of this by submitting 3 models in Task 5.

In previous years at CheckThat! the evaluation metric was MAE, as it could have been seen in Section 2.

4 Methods and runs

4.1 For Task 1

4.1.1 Training and test data

The data provided for this task contained tweets that were split into 2 main categories, train, and dev. The data was provided in both TSV[3] and JSON[4] files. We decided to only use the TSV files as we felt it was easier.

The datasets used were: "train" for the training of our model and "dev" to fine-tune the hyperparameters after evaluation. A training example can be seen in Table 1.

[3] https://www.imf.org/external/help/tsv.htm
[4] https://www.json.org/json-en.html

Table 1. Training example.

Topic id	Tweet id	Tweet URL	Tweet Text	Claim	Check worthiness
covid-19	1234964	https://twitter.com/Eric-Trump/status/1234964653014384644	Since this will never get reported by the media […]	1	1

We considered that the *tweet URL* and *tweet id* were irrelevant, so we did not include it in the data sent to our algorithm.

4.1.2 Preprocessing and feature extraction

Before feeding the data to the model, we had to preprocess the text. Csv[5] library was used to read from the files provided by the organizer after which we put the data in a list that contained in order the topic id, tweet id, tweet URL, tweet text, claim and label.

The data would then be sent to a tokenizer, we decided to use BertTokenizer[6] as this was the official method from Huggingface[7]. We would then pad the sentence to a maximum phrase limit that in our case was 121. Example tokenization is the following:

Input: "He poured orange juice on his cereal."
Output: [101, 2002, 8542, 4589, 10869, 2006, 2010, 20943,1012,102]

Fig. 1. Tokenization example.

After tokenization we loaded each individual field in a torch tensor[8] and inserted them in a TensorDataset[9] that contained: all the sentence ids, each individual tokenized sentence and the labels. A code snippet for this operation is the following:

```
all_topic_id_id = torch.tensor([f.topic_id_id for f in features],
dtype=torch.long)
all_tweet_text_id = torch.tensor([f.tweet_text_id for f in features],
dtype=torch.long)
all_claim_id = torch.tensor([f.claim_id for f in features],
dtype=torch.long)
dataset = TensorDataset(all_topic_id_id,all_tweet_text_id, all_claim_id)
return dataset
```

[5] https://docs.python.org/3/library/csv.html
[6] https://huggingface.co/transformers/model doc/bert.html#berttokenizer
[7] https://huggingface.co/
[8] https://pytorch.org/docs/stable/tensors.html
[9] https://pytorch.org/docs/stable/data.html#torch.utils.data.TensorDataset

4.1.3 Models

In designing the model, we decided to use BERT as a possible solution to the problem at hand. This model was chosen over other language models such as ULMFiT [15] as multiple papers demonstrate the performance benefits BERT has. For example, [16] trained an RNN on a very large text collection resulting in a 63.7% accuracy in the Winograd Schema Challenge [17] while [18] using a BERT model was able to achieve an accuracy of 72.5% on the same challenge. These results led us to choose the latter model as we feel it best fits our purpose.

The system consists of a pre-trained model called "bert-large-uncased"[10], it is a bi-directional transformer that contains 24 layers, 1,024 hidden layers, 16 heads, and 340 million parameters. The training has been done on lower-cased English text.

We used a combination of BertModel[11] and Adam[12] optimizer in order to get the best results. The hyperparameters are more or less standard, we tuned them empirically and arrived at the following best configuration: batch size 8, 5 epochs, and the Adam learning rate of 5e-5.

The pipeline of the algorithm implies first preprocessing (tokenize and pad the sentence in order to satisfy the condition of the model). Then, we shuffle the data in order to avoid overfitting, and we start the training with each epoch and we feed the data through the network. Then with backpropagation, we simply update the learning rate and tell the optimizer to update the parameters.

Evaluation of the trained network is done with the dev dataset, using the saved model in the previous step we feed the data through the network and compute loss on our trial data.

The experimental setup was done both locally and on the cloud. In the development stage, we trained the model locally using a computer with a 12 core CPU and 32 Gb of Ram which proved very inefficient, the training time took about two days which made us switch to a cloud setup. Using PyTorch[13] we made the switch to training the model on GPU using the Google Collaboratory[14] platform which lowered the training time to about an hour, this made a big difference as now we could make decisions regarding the model much faster, without waiting a long time for it to finish.

4.2 For Task 5

4.2.1 Training and test data

The data provided contained presidential elections debates and speeches from the United States in 2016. The data was of two main categories, training and test. The training had 50 files while the test had 20 files. The main difference from 2019 is that the organizers provided more training files but also more test scenarios. This can be seen

[10] https://huggingface.co/transformers/pretrained_models.html
[11] https://huggingface.co/transformers/model doc/bert.html#bertmodel
[12] https://huggingface.co/transformers/main classes/optimizer schedules.html#adamw
[13] https://pytorch.org/
[14] https://colab.research.google.com/

in the result of the candidates that now have a lower MAP compared to 2019 as the test file number has increased dramatically from last year.

We tried to further augment the models by taking files from 2019 that have not been included in 2020; we took training files but also test files with gold labels.

One training example with the available columns would be the following:

Table 2. Training example.

Line no.	Speaker	Text	Label
1	Trump	So Ford is leaving.	1

In the training of the model we ignored the Speaker and line number, we only fed the preprocessed text and label.

4.2.2 Preprocessing and feature extraction

Before sending the debate text to the machine learning algorithm we performed several preprocessing operations in a pipeline.

For all the models we first tokenized the text in order to break the phrase into individual terms. After tokenization, for the contrastive 2 submissions, Logistic Regression we used TF-IDF in order to extract the features in the form of a term frequency matrix. The implementation for TF-IDF was taken from Pyspark[15] and is a combination of 2 steps: HashingTF[16] and IDF[17]. The minimum document frequency for IDF was set to 10.

For the next two submissions, primary (Naïve Bayes) and contrastive 2 (Decision Tree) we decided to also use a term frequency matrix but with a different implementation; instead of using HashingTF we used CountVectorizer[18] which has a lower information loss and empirically it was observed these two algorithms perform better with this feature extractor. The settings of the CountVectorizer are the following: minimum term frequency is 1 and so is the minimum definition frequency, the maximum definition frequency is $2^{63} - 1$ and the vocabulary size is 2^{18} and for IDF we set a minimum term frequency of 3.

4.2.3 Models

After preprocessing the data and extracting the features they are sent to the machine learning models. We decided not to use any exterior resources for these models.

We trained the algorithms and tested them using the test data from 2019 where we attached the gold labels and used the provided organizers script to calculate MAP, RR,

[15] https://spark.apache.org/docs/latest/api/python/index.html
[16] https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/mllib/feature/HashingTF.html
[17] https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/mllib/feature/IDF.html
[18] https://spark.apache.org/docs/2.1.0/ml-features.html#countvectorizer

R-P and P@k. This gave us a rough estimation of the performance of the model but also a way to compare with other teams from last year [10].

The first and best model was based on **Naïve Bayes** which uses the default implementation from Pyspark, we fine-tuned it after multiple sessions of testing and used a multinomial model and set the smoothing to 1. Even though the model is quite simple it is very powerful. It can be seen from the results in Table 4 sometimes it is twice as accurate as other algorithms.

The second-best algorithm was **Logistic Regression**, initially this model performed poorly however we figured out that by increasing the minimum document frequency on IDF to 10 would increase its performance. The parameters of this model are the following: maximum iteration is set to 100, the regression parameter is 0, the tolerance value was 1e-6 and the aggregation depth was set to 2.

The third best was **Decision Tree**; this model uses a minimum document frequency in IDF of 3. We tried making it as best as we could and in order not to overfit the model we arrived at the following parameters: maximum depth was 30, we increased the maximum bins to 128 which allows the algorithm to consider more split candidates and make fine-grained split decisions, there were a minimum 5 instances per node and increased the maximum memory limit of the model to 4096Mb.

We trained the models locally, on CPU, the training time was rather fast, Decision Tree was the slowest.

5 Results

In this section, the results of the two task submissions will be discussed. Table 3 illustrates the results for Task 1, there were 12 teams and we ranked 11th with a MAP of 0.4950 while the best result had a MAP of 0.8064 (team Accenture). It should be noted here that contrastive 1 is better than our primary, we trained the primary with more epochs and wrongly evaluated as an increase in performance.

Table 4 presents the result in Task 5 where we ranked 2nd out of 3 teams with a MAP of 0.0515, the best being 0.0867 (team NLPIR01), and the worst 0.0183, almost 5 times worse than our submission.

Table 3. Task 1 Results

Sub.	MAP	RR	R-P	P@1	P@3	P@5	P@10	P@20	P@30
Accenture	0.8064	1.00	0.7167	1.0000	1.0000	1.00	1.0000	0.9500	0.7400
prim.	0.4950	1.0	0.4667	1.0	0.3333	0.4	0.6	0.6	0.46
con-1	0.5333	0.5	0.5167	0.0	0.3333	0.4	0.6	0.6	0.52

Table 4. Task 5 Results

Sub.	MAP	RR	R-P	P@1	P@3	P@5	P@10	P@20	P@30
NLPIR01 prim.	**0.0515**	**0.2247**	**0.0527**	**0.15**	**0.10**	**0.07**	**0.050**	**0.0375**	**0.0270**
con-1	0.0431	0.1735	0.0578	0.10	0.05	0.05	0.055	0.0450	0.0250
con-2	0.0328	0.1138	0.0282	0.05	0.05	0.03	0.035	0.0175	0.0190

For Task 5 the best result and the primary submission was of the Naïve Bayes model, contrastive 1 is 2nd place and it is the Logistic Regression algorithm, and finally contrastive 2 is based on a Decision Tree. The results are in accordance with what we have tested locally, we feel that the performance is good and that the models performed well in the evaluation stage.

5.1 Error analysis

The performance of both tasks is good; for Task 1 the main drawback of the model is the fact that we did not arrive at a finished product, we feel that the design of the model needs improvement, we do not believe it is able to extract relevant information thus an augmentation with a general knowledge ontology such as WikiData[19] would be a great addition.

For Task 5, the models are in a much more mature state as the performance show, we feel that they have reached their limit, the error stems from the lack of understanding of the sentence thus needing a much more complex system, probably based on a language model such as BERT.

6 Conclusion

In this paper, we proposed solutions for two of CLEF CheckThat! 2020 tasks, one approach is based on a bidirectional transformer and the others are based on machine learning. We achieved good results with all the submission and for future we would like to fine-tune our models in order to have a much better MAP, there is much room for improvement in Task 1 and for task 5 a language model approach would be interesting to see in action.

Acknowledgements

This work was supported by project REVERT (taRgeted thErapy for adVanced colorEctal canceR paTients), Grant Agreement number: 848098, H2020-SC1-BHC-2018-2020/H2020-SC1-2019-Two-Stage-RTD.

[19] https://www.wikidata.org/

References

1. Barrón-Cedeño, A., Elsayed, T., Nakov, P., Da San Martino, G., Hasanain, M., Suwaileh, R., Haouari, F., Babulkov, N., Hamdan, B., Nikolov, A., Shaar, S., Sheikh Ali, Z. (2020) Overview of CheckThat! 2020: Automatic Identification and Verification of Claims in Social Media. In Working Notes of CLEF 2020.
2. Arampatzis, A., Kanoulas, E., Tsikrika, T., Vrochidis, S., Joho, H., Lioma, C., Eickhoff, K., Névéol, A., Cappellato, L., Ferro, N. (2020). Experimental IR Meets Multilinguality, Multimodality, and Interaction. In Proceedings of the Eleventh International Conference of the CLEF Association (CLEF 2020). Lecture Notes in Computer Science (LNCS) 12260, Springer, Heidelberg, Germany.
3. Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2018) BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, abs/1810.04805.
4. Hansen, C., Hansen, C., Simonsen, J. G., Lioma, C. (2019) Neural Weakly Supervised Fact Check-Worthiness Detection with Contrastive Sampling-Based Ranking Loss. In Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum, Lugano, Switzerland, September 9-12, 2019 http://ceur-ws.org/Vol-2380/paper_56.pdf .
5. Hochreiter, S., Schmidhuber, J. (1997) Long short-term memory. In Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.
6. Favano, L., Carman, M., Lanzi, P. (2019) TheEarthIsFlat's submission to CLEF'19 CheckThat! challenge. In CLEF 2019 Working Notes. Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum. CEUR Workshop Proceedings, CEUR-WS.org, Lugano, Switzerland.
7. Gasior, J., Przyby la, P. (2019) The IPIPAN team participation in the check-worthiness task of the CLEF2019 CheckThat! lab. In CLEF 2019 Working Notes. Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum. CEUR Workshop Proceedings, CEUR-WS.org, Lugano, Switzerland.
8. Su, T., Macdonald, C., Ounis, I. (2019) Entity detection for check-worthiness prediction: Glasgow Terrier at CLEF CheckThat! 2019. In CLEF 2019 Working Notes. Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum. CEUR Workshop Proceedings, CEUR-WS.org, Lugano, Switzerland.
9. Cortes, C., Vapnik, V. N. (1995) Support-vector networks. In Machine Learning. 20 (3): 273–297. CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018
10. Coca, L., Cusmuliuc, C.G., Iftene, A. (2019) 2019 UAICS. In CLEF 2019 Working Notes. Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum. CEUR Workshop Proceedings, CEUR-WS.org, Lugano, Switzerland.
11. Wang, D., Simonsen, J., Larseny, B., Lioma, C. (2018) The Copenhagen Team Participation in the Factuality Task of the Competition of Automatic Identification and Verification of Claims in Political Debates of the CLEF-2018 Fact Checking Lab. CLEF 2018 Working Notes.
12. Valueva, M. V., Nagornov, N. N., Lyakhov, P. A., Valuev, G. V., Chervyakov, N. I. (2020) Application of the residue number system to reduce hardware costs of the con-volutional neural network implementation. In Mathematics and Computers in Simula-tion. Elsevier BV. 177: 232–243. doi:10.1016/j.matcom.2020.04.031. ISSN 0378-4754. Convolutional neural networks are a promising tool for solving the problem of pattern recognition
13. Beitzel, S.M., Jensen E.C., Frieder O. (2009) MAP. In LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA
14. Craswell, N. (2009) Mean Reciprocal Rank. In LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA

15. Howard, J., Ruder, S. (2018) Fine-tuned language models for text classification. CoRR, abs/1801.06146.
16. Trinh, T. H., Le, O. V. (2018) A simple method for commonsense reasoning. CoRR, abs/1806.02847.
17. Levesque, H. J., Davis, E., Morgenstern, L. (2012) The winograd schema challenge. In Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, KR'12, 552–561. AAAI Press.
18. Kocijan, V., Cretu, A. M., Camburu, O. M., Yordanov, Y., Lukasiewicz, T. (2019) A surprisingly robust trick for winograd schema challenge. CoRR, abs/1905.06290
19. Shaar, A., Babulkov, N., Alam, F., Barrón-Cedeño, A., Elsayed, T., Hasanain, M., Suwaileh, R., Haouari, F., Da San Martino, G., and Nakov, P. (2020). Overview of CheckThat! 2020 English: Automatic Identification and Verification of Claims in Social Media. In Working Notes of CLEF 2020.