Team Xenophilius Lovegood at SemEval-2019 Task 4: Hyperpartisanship Classification using Convolutional Neural Networks

Albin Zehe, Lena Hettinger, Stefan Ernst, Christian Hauptmann and Andreas Hotho
DMIR Group, University of Wuerzburg
<surname>@informatik.uni-wuerzburg.de

Abstract

This paper describes our system for the SemEval 2019 Task 4 on hyperpartisan news detection. We build on an existing deep learning approach for sentence classification based on a Convolutional Neural Network. Modifying the original model with additional layers to increase its expressiveness and finally building an ensemble of multiple versions of the model, we obtain an accuracy of $67.52\,\%$ and an F1 score of $73.78\,\%$ on the main test dataset. We also report on additional experiments incorporating handcrafted features into the CNN and using it as a feature extractor for a linear SVM.

1 Introduction

The goal of SemEval 2019 Task 4 is to determine whether a news article blindly follows a political argumentation or not, which is referred to as "hyperpartisan news" (Kiesel et al., 2019). Instead of predicting the exact political orientation, it focuses on whether an article is hyperpartisan in any way. This is a very topical issue since news are easily able to reach millions of people over the internet, and in recent years have been excessively used to influence the population, for example regarding elections. Specifically, one sided media coverage influences a lot of readers without their knowledge, demonstrating the necessity of automated detection of hyperpartisan news.

Approach In this work, we make use of deep learning models to address this task. We decided to adapt the sentence CNN proposed by Kim (2014), as it has been shown to be a strong baseline for text classification tasks. Since the model was originally designed for the classification of sentences, we had to make some modifications in order to deal with the longer texts provided in this task. While the very shallow model originally proposed by Kim (2014) is enough to adequately represent sentences, we found that it is not expressive enough to model entire news articles. Thus, we added a second convolutional layer and a batch normalization layer (Ioffe and Szegedy, 2015) to the model. Additionally, we specified a maximum length for articles, after which they are cut off. These modifications will be described in more detail in Section 3. We also experiment with including some hand-crafted features into the model in an attempt to improve the performance. Finally, we build an ensemble of multiple models to obtain our final results.

2 Feature Extraction

In order to train our models, we need to represent the input texts in some machine readable form. The data provided in the task contains multiple kinds of information that we use in different parts of our system, namely the CNN model and the hand-crafted features extracted to provide additional information.

2.1 CNN Model

On the one hand, our main CNN model is based purely on the text of the articles and requires little pre-processing. For this part, we built a specialized parser[1] to remove HTML tags and split the texts into tokens, which are then directly used as input to the CNN. We chose to allow some special characters like punctuation marks to support the model identifying different sentences. Contractions like *they're* or *he's* are split to obtain separate tokens which can then be mapped to existing ones.

An article is then represented as a sequence of one-hot vectors, where each dimension corresponds to a word in the vocabulary. This sequence is concatenated to form a matrix $M \in \mathbb{N}^{l \times v}$, where l is the length of the article and v is the vocabulary size.

[1] https://github.com/o8Gravemind8o/nlp_tokenizer

Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019), pages 1047–1051
Minneapolis, Minnesota, USA, June 6–7, 2019.

Word Embeddings As is common for NLP tasks, we use embedding vectors to represent the semantic meaning of tokens. Since the provided datasets are rather large and previous work has shown that domain specific word embeddings can greatly improve classifier performance compared to general embeddings (Hettinger et al., 2018), we train our embeddings on these datasets. More specifically, we use Word2Vec (Mikolov et al., 2013) as well as FastText (Bojanowski et al., 2017) to retrieve different embeddings and see how they perform in different approaches.

Dealing with Variable Article Length Due to CNNs not being able to process input of arbitrary length, we decided to represent articles with a fixed length of 2000 tokens. Articles that exceed this length are cut off at 2000 tokens. This saves a lot of training time and affects less than 3 % of the articles. In the same way, we pad articles shorter than 2000 tokens with zeros to achieve a consistent input size.

2.2 Hand-Crafted Features

On the other hand, we employ hand-crafted features partially based on the metadata that is contained in the HTML. We do this to enable our classifiers to use information that may not be contained in the raw text and also possibly recover information that is lost by the length limit we impose on the articles. For this purpose, we choose several kinds of information from the articles, inspired by Potthast et al. (2018). First, we count (a) every token in the article and (b) tokens which are placed between quotation marks. Furthermore, we use the corresponding HTML tag to count paragraphs and calculate the average number of tokens per paragraph. Finally, we use the overall number of hyperlinks as well as the number of internal and external links. These values are concatenated to form a feature vector that can be used as input to an SVM or as additional input to the CNN.

3 Model Architectures

In this section, we describe the architectures we evaluated in our experiments, starting with the base CNN model from Kim (2014) and extending this model step by step. We also describe an experiment to use the CNN model as a feature extractor for an SVM.

Base Model: Sentence CNN We use the sentence CNN from Kim (2014) as a starting point for our model, illustrated in Figure 1. Articles are fed into the CNN represented by the matrix described in Section 2.1. The first layer of the network then converts the one-hot representation to an embedding representation. To obtain a vector representation of the words, we use two different approaches described in Kim (2014), CNN-Rand and CNN-Static. With CNN-Rand, word vectors are initialized randomly and learned during training. CNN-Static uses the embeddings described in Section 2.1 and does not change them during training. The CNN extracts features from the articles through a convolution layer followed by max-over-time pooling. Classification of an article is obtained by flattening the max-pool feature map and passing the features through a fully connected layer.

First Extension: Article CNN As the base model discards all but one feature from the convolution activation map of each filter by using max-over-time pooling, a lot of features are lost. In the model's original task, which is sentence classification, this is not much of an issue. However, our experiments show that for the classification of articles (which are much longer than one sentence), we need to keep more information.

Therefore, we modify the sentence CNN and use this as a second model, which we call *Article CNN*. We add a second convolutional layer after the first one to learn features that cover a larger range in the article. Furthermore, we add a batch normalization layer after the first and after the second convolutional layer to speed up the training process (Ioffe and Szegedy, 2015).

Second Extension: Article CNN with Hand-Crafted Features In an attempt to incorporate additional information into our model, we provided the model with some handcrafted features described in Section 2.2. To make use of handcrafted features, we append them to the flattened output of the pooling layer of the Article CNN. Since we found that the model can not learn from the combination of CNN and hand-crafted features with only one dense layer at the end, a second dense layer is inserted before the first one. We refer to the resulting third model as *Article CNN HC*.

Model Variant: CNN as a Feature Extractor In an additional experiment, we use the CNN as a feature extractor for an SVM. To this end, we first

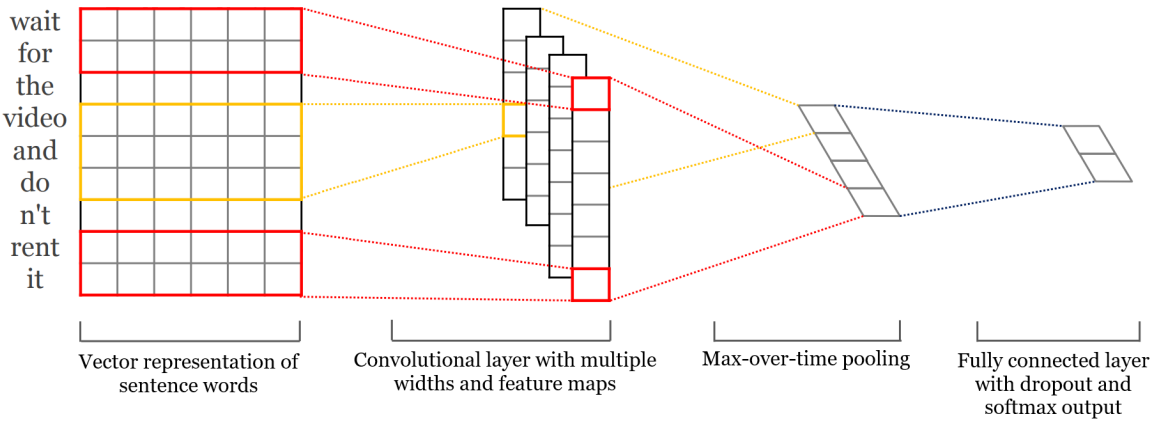

Figure 1: Architecture of sentence CNN by Kim (2014).

train the Article CNN regularly. We then convert an article to a feature representation by feeding it into the trained CNN and extracting the representation before the first dense layer. This vector is concatenated with the hand-crafted features described in Section 2.2 and used to train a linear Support Vector Machine (SVM) as an alternative to the neural classifier of the CNN.

4 Evaluation

After defining our models, we now shortly describe the datasets (for a more detailed description, see Kiesel et al. (2019)) before presenting the results of training and evaluation.

4.1 Data

For training and validation, the task provides 750 000 news articles, which are equally distributed into the two classes hyperpartisan and not hyperpartisan. 600 000 of these were used for training, the remaining 150 000 for validation. These articles have not been labeled individually, but according to their publisher, making them a form of weakly labelled data. The official evaluation was then performed on two concealed datasets, one with articles manually labeled by humans and one again labeled by publisher. The evaluation data contains 628 articles labeled individually and 4000 by publisher, with both being equally distributed into the two classes.

4.2 Metrics

We determine the performance of our models by measuring accuracy (Acc) and F1 score (F1). As accuracy is the official evaluation metric of SemEval Task 4, we optimize for this metric.

4.3 Training and Results

We chose to optimize the hyperparameters of our models by random search (Bergstra and Bengio, 2012). The hyperparameters with the best accuracy values of each architecture are shown in Table 1. All configurations use the CNN-static variant. Models were trained for a maximum of 5 epochs with a batch size of 256. We employed early stopping when the validation accuracy did not improve for 8 consecutive batches.

Results on the Validation Dataset First, we report the results obtained by our models on the validation dataset. All results on this dataset are shown in Table 2. The best configuration of the Sentence CNN achieves a maximum accuracy of 62.13 % and an F1 score of 70.47 %.

Our first extension, the Article CNN, increases the accuracy by 1.68 percentage points and F1 score by 6.09 percentage points. We attribute this to the increased model capacity, which enables the model to represent articles more adequately. With the max-pooling layer after the first convolution layer, the whole article is reduced to one value per convolution filter, which covers a maximum of 11 words (filter size is 11). The second convolution layer contains information of several outputs of the first layer, hence learning higher level features. As a result of that, less information about the article is lost by max-pooling.

Our second modification, the Article CNN HC, however, decreases the model's performance, as does using an SVM as a classifier instead of the

	Convolution 1		Convolution 2		Dense Layer		Miscellaneous	
Model	Filter Size	Filters	Filter Size	Filters	Dropout keep	Units	Activation	Embeddings
Sentence CNN	5	168	—	—	0.9	168	tanh	Word2Vec
Article CNN	9,10,11	306	7	199	0.837	4179	ReLU	FastText
Article CNN HC	7	405	9	210	0.079, 0.274	1890+7, 123	ReLU	Word2Vec

Table 1: Best hyperparameters obtained via random search. Article CNN HC has two dense layers, hence two numbers for dropout and units. We evaluated multiple filter sizes for Article CNN only, due to time constraints.

Model	F1	Acc
Sentence CNN	70.47	62.13
Article CNN	**76.56**	63.81
Article CNN HC	63.02	60.30
Article CNN + SVM	62.07	58.18
SVM HC	66.91	52.83
Ensemble 3	68.69	64.94
Ensemble 5	68.98	**66.01**

Table 2: Best results achieved on the validation dataset.

final fully-connected part of the CNN (Article CNN + SVM). We also trained an SVM purely on the hand-crafted features for comparison (SVM HC), leading to a lower accuracy but higher F1 score than both variants of Article CNN HC.

Finally, we used an ensemble of multiple models for prediction. To this end, we used either 3 or 5 of our best individual models and combined their predictions by majority vote. Because of the non-deterministic nature of training (shuffling of the input data and random initialization of the network) (Reimers and Gurevych, 2017), multiple versions of the same model with similar accuracy may rely on different features and make different mistakes. Thus, a combined prediction by the best models can improve overall accuracy. This is confirmed by the results of Ensemble 3 and 5 presented in Table 2, leading us to submit these ensembles as our final systems. Our best model, Ensemble 5, combines 5 individual Article CNNs.

Results on the Test Dataset For the final evaluation, participants were able to submit two models for evaluation on the non-public test sets. We submitted our two ensemble models comprised of multiple instances of the Article CNN. Additionally we were able to evaluate our single best Article CNN on both test sets, giving us the evaluation results but not appearing on the scoreboard. Results on the test dataset are shown in Table 3. As on the validation dataset, the ensembles outperform the

	Article		Publisher	
Model	F1	Acc	F1	Acc
Article CNN	70.26	64.81	67.81	**66.78**
Ensemble 3	72.05	65.29	**70.09**	66.08
Ensemble 5	**73.78**	**67.52**	69.85	66.28

Table 3: Results on the hidden test datasets.

Article CNN in prediction accuracy on the main test set. Our best performing model, the Ensemble 5, reaches an accuracy of 67.52 % on the by-article test set and 66.28 % on the by-publisher test set. This corresponds to rank 25 of 42 on the Main Leaderboard (by-article) and rank 4 out of 28 on the by-publisher Leaderboard.[2]

5 Conclusion

In this paper, we have described our approach for SemEval Task 4 to detect hyperpartisanship in news articles. We trained a CNN for sentence classification and improved its performance by adding a second convolution layer and batch normalization. Moreover, we combined this model with handcrafted features. However, this did not lead to an improvement of classification performance, nor did the use of an SVM as an alternative classifier. Finally, an increase of accuracy was achieved by combining our best models for ensemble prediction. Through this approach, we obtained an accuracy of 67.52 % and F1 score of 73.78 % on the by-article test dataset as well as 66.28 % accuracy and 69.85 % F1 score on the by-publisher data set. For future research, possible modifications to our Article CNN that may bring further improvement are using more channels for the input or different filter sizes for max pooling. Apart from that, using Transformer models (Vaswani et al., 2017) could be rewarding, as they have become the standard for many tasks in Natural Language Processing.

[2]We would like to note that, by F1 score, we would rank 14 out of 42 in the Main Leaderboard (by-article).

References

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Lena Hettinger, Alexander Dallmann, Albin Zehe, Thomas Niebler, and Andreas Hotho. 2018. Claire at semeval-2018 task 7: Classification of relations using embeddings. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. Semeval-2019 task 4: Hyperpartisan news detection. *Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval 2019)*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek Bevendorff, and Benno Stein. 2018. A stylometric inquiry into hyperpartisan and fake news. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 231–240.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.