HIIwiStJS at GermEval-2018: Integrating Linguistic Features in a Neural Network for the Identification of Offensive Language in Microposts

Johannes Schäfer
Institute for Information Science and Natural Language Processing
University of Hildesheim, Germany
johannes.schaefer@uni-hildesheim.de

Abstract

This paper describes our submission for the GermEval-2018 shared task on the identification of offensive language. We use neural networks for both subtasks: *Task I — Binary classification* and *Task II — Fine-grained classification*. We comparatively evaluate the use of typical textual features with extensions also considering metadata and linguistic features on the given set of German tweets. Our final system reaches 73.69% macro-average F_1-score in a cross-validation evaluation for the binary classification task. Our best performing model for the fine-grained classification reaches an macro-average F_1-score of 43.24%. Furthermore, we propose methods to include linguistic features into the neural network. Our submitted runs in the shared task are: **HIIwiStJS_coarse_[1-3].txt** for *Task I* and **HIIwiStJS_fine_[1-3].txt** for *Task II*.[1]

1 Introduction

The automatic analysis of social media microposts such as *Twitter*[2] messages (*tweets*) gained more and more interest in recent years due to the necessity to process their increasing amount and variety. The anonymity of the web allows users to overcome their inhibitions more quickly which fosters the use of offensive language. Operators of social media websites are required to filter overly hurtful, derogatory or obscene comments and strive to acquire methods to automatically identify potentially offensive posts.

Schmidt and Wiegand (2017) present a survey on hate speech detection which is closely related to the detection of abusive language. They give an overview of typically used methods and features for the task, ranging from surface, sentiment, linguistic and knowledge-based features to higher-level features making use of lexical resources or metadata information. They especially point out the variety of the task and the lack of comparability of different research systems typically based on supervised learning, as no benchmark dataset is available.

Abusive language in English online user content is detected by a system developed by Nobata et al. (2016). They tackle the problem of noisiness of the data in conjunction with a need for world knowledge by using a feature-rich regression model. They exploit both character n-gram features as well as a variety of linguistic features including automatic syntactic and semantic annotations.

A multi-level classification of abusive language is given by Razavi et al. (2010). They particularly focus on flame detection and use a combination of classifiers supported by a dictionary of insulting and abusing language.

Most related work operates on English texts for which also the largest amount of data is available. One of the few presented research works on German data is given by Köffer et al. (2018). They collected a dataset of user comments on news articles from the web with a focus on the refugee crisis in 2015/16. Additionally, they provide a labeled dataset with comments marked as hateful or non-hateful and demonstrate the transferability of approaches developed for English data to German.

Neural networks (NNs) have been on the rise only in recent years as they require vast amounts of data and processing power which both only became available recently in the field of natural language processing. In applications on micropost classification neural networks also have seen research. For example, Del Vigna et al. (2017) perform hate speech detection on Italian user posts in the so-

[1] The IDs 1-3 correspond to our developed neural network systems used for the prediction as follows: ID 1 - Baseline model; ID 2 - Text & Metadata model; ID 3 - Text & Metadata & POS model.

[2] https://twitter.com/

cial network *Facebook*[3] using recurrent neural networks (specifically a LSTM network) which they compare to an approach using support vector machines.

Twitter data is analyzed by Founta et al. (2018) in an approach to detect different types of abusive behavior. They analyze both textual and user properties from different angles of abusive posting behavior in a deep learning architecture. In their model they consider a variety of metadata and include it into a NN model which learns text sequence features using a recurrent neural network. They show that training the sub-networks of different input types requires specific attention since simply training all of them at once in a combined model leads to unwanted interactions.

In the present paper, we also utilize neural networks to train models which identify offensive language in microposts from *Twitter*. Neural networks have the advantage to work with a high input dimensionality where it is not clear which features might be helpful concerning the prediction task. Given enough training data, the network is able to learn a complex, non-linear encoding of the input specifically for the desired classification. A certain intuition when selecting the features is however advised, since too many unrelated features can introduce a high amount of noise to the model.

Our approach shows similarities to the methods presented by Founta et al. (2018), however, we adapt the configuration of the network to our classification task, dataset and to tweets in German. Thus, in this work we first present a task, domain and language adaptation of their methods. Typically, neural networks are designed to only operate on raw textual input and are then fine-tuned to be able to learn patterns themselves. In our work the model is additionally given automatically pre-computed linguistic annotations. We present possibilities and early research of including such annotations into a combined neural network.

In the following sections we introduce our models in detail (Section 2), describe our dataset and linguistic processing (Section 3) and finally report on experiments (Section 4).

2 Methods

We implemented our models in *Python* using the module *keras* with the *TensorFlow* backend. To train our NN models we use the *Adam* optimizer and as loss function binary cross-entropy for the binary classification and categorical cross-entropy for the fine-grained classification respectively. We train the models on our dataset using a batch size of 64 which we determined experimentally to be most suitable.

[3] https://www.facebook.com/

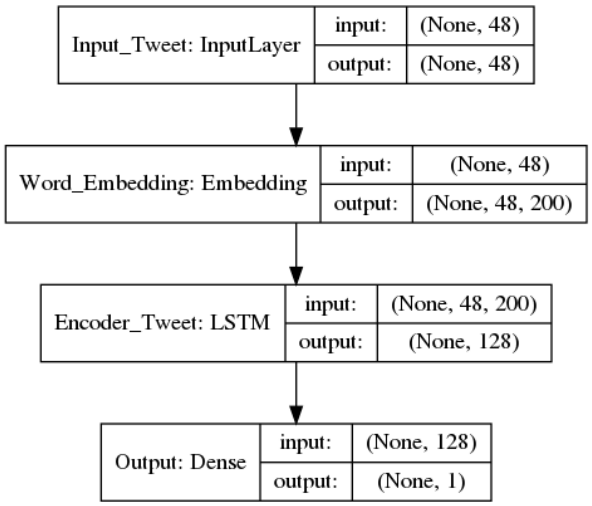

Figure 1: Baseline neural network, binary classification variant.

2.1 Baseline Model: Network Architecture

Our baseline model operates on the raw tweet input with a minimal amount of preprocessing. We compute word embeddings and feed them into a recurrent layer which is typically applied to sequence data. Finally, the output prediction is computed by a fully connected dense network. Figure 1 shows the overall structure of the baseline neural network with the specific dimensions[4] of the data going through the respective layers. In the following three sections we go over the detailed configuration.

2.1.1 Input and Embedding Layer

In the baseline model we use the raw tweet as input for the neural network in a simple input layer (*Input_Tweet* in Figure 1) which instantiates the *Keras* tensor.

Next, we decided to use word embeddings[5] to identify offensive language in tweets since we understand words as linguistically meaningful units in twitter messages; as these are also sufficiently

[4] Note that the unspecified ("None") dimension value in the figure corresponds to the number of samples which depends on the batch size (64 in our experiments).

[5] Brief experiments with character embeddings (also including convolutional layers capturing character n-grams) did not seem to lead to promising results on our dataset.

small, they lead to a feature vector of acceptable length and richness. Additionally, choosing word embeddings enables us to extract them from any tweet without relying on linguistic knowledge, only using a simple tokenizer. As further processing of our data in the neural network is more efficient when all input samples are of the same length, we (pre-) pad the tokenized tweets to sequences of 48 tokens. We determined this value specifically for our dataset by including the 95th percentile of all contained tweet lengths. Thus, with this method only 5% of the tweets are truncated.

As initial weights of the embedding layer we utilize pre-trained word embeddings which is a conventional method usually having the effect that the model converges faster. In our experiments we use the word embeddings provided by Cieliebak et al. (2017) which are trained with *Word2Vec* on 200 million German tweets using 200 dimensions and are available on the web[6].

When the weights of the embedding layer are set to not trainable, the total number of trainable parameters in our neural network is substantially reduced. This allowed us to design the remainder of the network in a very precise fashion, however, in this case the model cannot learn from out-of-vocabulary (OOV) words. In early experiments we found that only using the pre-trained embeddings as initial weights and allowing the model to fit these weights to the data during training leads to a better performance. Thus, we pre-compute only an embedding matrix using these weights which also contains randomized vectors for words in our dataset which are not contained in the embeddings. This approach also seems well-grounded since the used word embeddings are not specifically trained on abusive language and texts from social media in general tend to have a high number of OOV words.

Therefore, the output of our embedding layer for tweets (*Word_Embedding* in Figure 1) has the shape of: number of sequences, length of sequences (48), size of word vectors (200).

2.1.2 Recurrent Layer

As main neural network structure to encode the word sequence of a tweet we use a recurrent layer which sequentially processes data samples while constantly updating an internal state. Recurrent neural networks (RNNs) have proven to be highly efficient in modeling text since they intrinsically consider context information when learning predictions on word sequences. In early experiments we achieved the best performance with a RNN containing long short-term memory units (Hochreiter and Schmidhuber, 1997, LSTM), also known as a LSTM network which outperformed both a simple RNN and gated recurrent units (Cho et al., 2014, GRU). Even though our sequences are of relatively short length, we assume that the LSTM can track long-term dependencies nevertheless, for example, mentions of typical targets of insults at the start of a tweet with the actually insulting word being at the end of the tweet.

Experimentally, we determined a number of 128 units to be best performing for our LSTM network (*Encoder_Tweet* in Figure 1). To avoid overfitting, we set the recurrent dropout value of 0.5 in this layer. For further encoding we tested additional recurrent or fully-connected dense layers, however, did not achieve performance improvements.

2.1.3 Output Layer

The output layer (*Output* in Figure 1) of our neural network consists of a fully-connected dense layer which maps the output of the recurrent layer to a probabilistic prediction. To avoid overfitting to the training data, we apply L2 regularization in the kernel of this layer (with $\lambda = 0.01$).

For the binary classification task we use the sigmoid activation function and chose a single output unit which expresses the probability of the sample containing abusive language. The probabilistic prediction is transformed into a binary prediction using a 0.5 threshold.

For the fine-grained prediction we use the softmax activation function and as the number of output units the number of labels (4 in our dataset). The one label with the maximum probability is then selected as the predicted label for each sample.

2.2 Text & Metadata Model: Network Architecture

As second model we developed a neural network which combines textual sequence input in a sub-network similar to the above-mentioned baseline model with an additional metadata sub-network. An overview of the combined network is given in Figure 2. First, we describe in Section 2.2.1 the new metadata network and then in Section 2.2.2 we show how we include the metadata sub-network into the text sequence-based network.

[6] https://www.spinningbytes.com/resources/wordembeddings/

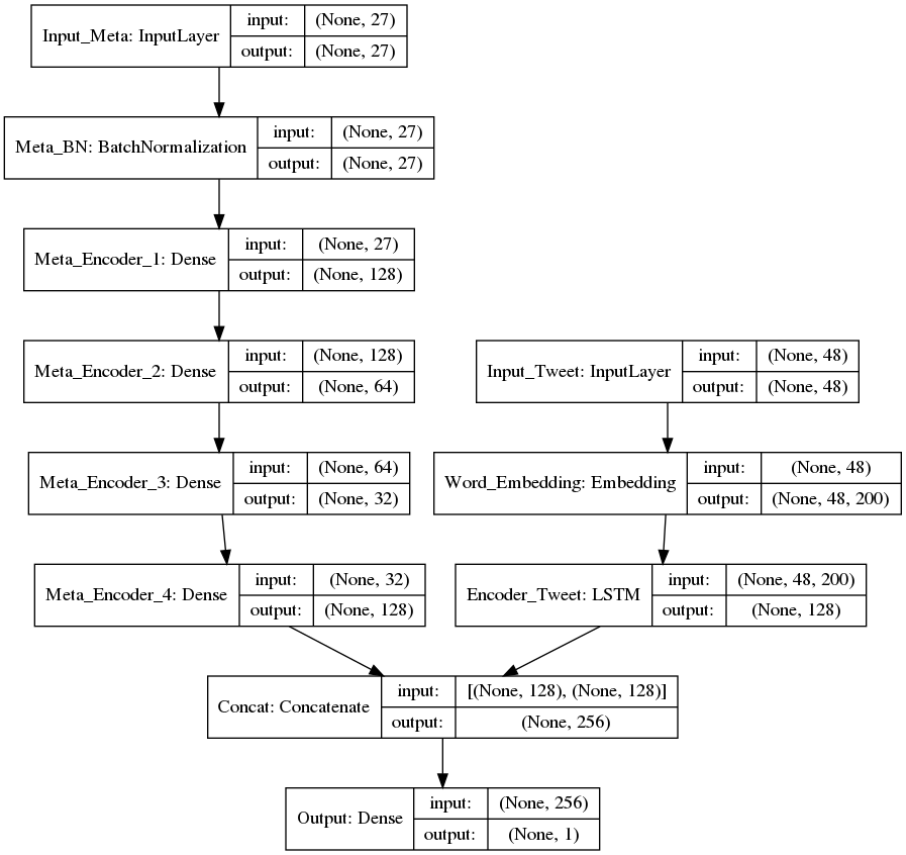

Figure 2: Text & metadata neural network, binary classification variant.

2.2.1 Metadata Network

In our setting we understand metadata as numerical data describing features of a single tweet going beyond its text, possibly with the aid of an external lexicon; however, not using its word sequence structure. For example, we extract the number of @-marked user name mentions and #-marked keywords from tweets, count special characters and attempt to match entries of pre-collected lexicons containing lists of profane words, words with known sentiment polarity or typical targets of abusive speech. The list of the 27 types of metadata considered in our experiments is given in Section 3.1.

The extracted numerical metadata features are fed into an input layer to instantiate the *Keras* tensor (*Input_Meta* in Figure 2). Next we apply a batch normalization layer (*Meta_BN*) transforming the values to a mean and unit variance of zero, which optimizes the neural network performance. To automatically compute a high-level encoding of our metadata features we utilize a technique known as bottleneck (Tishby and Zaslavsky, 2015; He et al., 2016) which consists of a sequence of several differently-sized fully-connected dense layers. We experimented with various configurations and found a sequence of three dense layers ((*Meta_Encoder_[1-3]* in Figure 2 with 128, 64 and 32 neurons respectively) to work best on our dataset. A forth dense layer (*Meta_Encoder_4*) is added which transforms the output to a tensor of the same dimensionality (128) as the output of the text sequence-based network. This supports the combination of the two sub-networks to a single neural network model which operates more efficiently on sub-networks of the same size.

For all dense layers in this network we use the *tanh* activation function as it works efficiently with standardized numerical data. An L2 regularization is also applied in all layers to reduce overfitting, however, with an relatively low $\lambda = 0.0001$. This can be justified since the input dimension of our metadata features is quite low (27), thus the model should not be overly complex to still be able to learn patterns.

2.2.2 Combined Text & Metadata Network

We combine the above-mentioned metadata network with the text sequence-based baseline net-

work described in Section 2.1 as follows. In front of the fully-connected output layer, we insert a concatenate layer which appends the output of the text encoder (*Encoder_Tweet* in Figure 2) to the output of the final layer of our metadata sub-network (*Meta_Encoder_4*). Finally, to predict the output probabilities we again use a fully-connected dense layer (*Output* in Figure 2 with the output dimension in the figure given as 1 for the binary classification). The configuration of this layer is unchanged from the baseline model with the exception that its input dimensionality is doubled (256) as we concatenate two tensors of size 128 (second dimension).

When training this network as a combined model at once, we struggled to achieve observable improvements in comparison to the baseline network. This is justified by the fact that the two paths might have a different convergence rate, which we observed in experiments. Thus, one path can dominate in predictions past a certain epoch and prevent the weights of the other path from getting significant updates. To avoid this problem, we use transfer learning. We first train both sub-networks separately, then freeze their weights and train the combined model using the pre-trained sub-networks. When training the sub-networks separately, we first remove the other respective sub-network and the concatenate layer as we can compute the output directly (e. g. when training the text sequence-based network we exactly train the baseline network as given in Figure 1). In a second step we remove the output layer of each sub-network, concatenate the output tensors and then add an output layer again to receive the network displayed in Figure 2. In the combined model, however, only the 257 weights of the final layer (*Output*) are trainable.

2.3 Text & Metadata Model extended with Linguistic Analyses

In this section we describe experimental methods where we integrate pre-computed linguistic analyses of the given text into our above-mentioned text & metadata neural network. We make use of the following three additional systems.

1. Compound splitter *COMPOST* (Cap, 2014).
2. Part-of-speech (POS) tag sequences of normalized tweets, computed using the Python module textblob-de[7].
3. Dependency parse trees for normalized tweets using the *mate* parser (Bohnet, 2010).

[7] http://textblob-de.readthedocs.io

We use 1. to reduce the number of unknown words when importing the pre-trained word embeddings for our data. We first split all compounds in a tweet before running the word embedding look-up. Thus, we reduce the dimensionality of the embedding layer by reducing the number of distinct words as we consider components instead.

For integrating the POS tag sequences into our network, we simply add another LSTM-based sub-network with the same architecture as the text sequence-based network as described in Section 2.1 to our combined network. This new sub-network operates on trainable embeddings (initialized randomly) of the POS tag sequence as input.

We integrate the dependency structures with two different methods into our combined model. First, we also use the tag sequence similar to the above-mentioned POS tags in an additional LSTM-based sub-network. In a second approach we only consider the 1000 most frequent combinations of a word with its dependency arc label and encode these in a 1-hot vector for each tweet. This vector can then be fed into a network of fully-connected dense layers similarly to the metadata sub-network.

Finally, a top-level model combines the outputs of the sub-networks in a concatenate layer in the same fashion as we integrated the metadata sub-network.

3 Data processing

The training dataset of the *GermEval-2018 Shared Task on the Identification of Offensive Language* consists of 5,009 tweets with two annotation layers. No user metadata is given. The first annotation layer marks the binary classification of tweets, i. e. they are either marked when containing offensive language using the label "OFFENSE" or in any other case using the label "OTHER". The second annotation layer is used for the fine-grained classification tasks where three subcategories of offensive language are marked. Thus, this layer has four labels in total: "PROFANITY", "INSULT", "ABUSE" and "OTHER". Before training the neural networks on the raw tweets we apply certain pre-processing techniques which we describe in this chapter.

To be able to operate on words instead of the character sequence input, we apply tokenization using the *TweetTokenizer*[8] implemented in the

[8] https://www.nltk.org/api/nltk.tokenize.html

nltk.tokenize Python module. This tokenizer is especially designed to process Twitter-specific expressions such as emoticons. It is introduced for English but, according to our observations, it also works fine on our German data.

We extract Twitter-specific mentions (marked by @) and #-marked keywords[9] from tweets using regular expressions. These lists are later used during the metadata extraction process.

To extract rather conventional linguistically structured sentences we apply a normalization step. Here, we remove *Twitter*-specific user name mention (@) and #-markers, line break markers, vertical bars inside words which were used to mark keywords when word affixes are present and replace xml-escaped symbols. Furthermore, we add missing whitespace characters after punctuation marks, replace ascii-emoticons with the corresponding Unicode characters and finally remove remaining special characters.

3.1 Metadata extraction

We extract metadata for each tweet from features going beyond its text sequence structure, however, only with the tweet text (and normalized text) as input. Typically used Twitter metadata features consider the author of a tweet, the number of his followers, the location, account age, etc. This group of metadata features are however not available in our dataset as the task focuses only on the linguistic content of microposts. All our metadata features are numerical while some take external source lexicons into account. We use the following external resources (all German language):

(i) List of strings matching typical targets of abusive speech, manually assembled according to the annotation guidelines of the shared task (Ruppenhofer et al., 2018) including: feminists, black people, muslims, jews, homosexuals (LGBT), refugees, members of political parties, etc.;
separate list for strings matching German public media names.

(ii) Lists of gender-specific names by the city of Cologne, available on the web[10].

(iii) Lists of positive, negative and neutral German polarity clues from the University of Bielefeld (Waltinger, 2010).

(iv) List of 1,782 swearwords from the web[11].

(v) Lists of words with positive and negative sentiment value from the University of Leipzig: SentiWS (Remus et al., 2010).

(vi) Lexicon of words with positive, negative and neutral sentiment values from the University of Zürich.[12]

Note that the latter two resources not only contain lists of words but additionally weights which we also consider for our metadata features.

In total we extract the following 27 metadata features for each tweet:

1. Length in number of characters;
2. Length of the normalized text in number of characters;
3. Number of words starting with an uppercase letter;
4. Number of user mentions (marked by @);
5. Number of user mentions in the first half of the tweet;
6. Number of user mentions in the second half of the tweet;
7. Number of matches of targets from list (i) in the normalized text;
8. Number of matches of public media-specific strings from list (i) in the list of mentions;
9. Number of matches of targets from list (i) in the list of mentions;
10. Number of female names in the mentions using list (ii);
11. Number of male names in the mentions using list (ii);
12. Number of #-marked keywords;
13. Number of matches of targets from list (i) in the list of keywords;
14. Number of matches of public media-specific strings from list (i) in the list of keywords;
15. Number of punctuation marks;
16. Number of reduplications of punctuation marks;
17. Number of special characters (mostly emoticons);
18. Number of words with uppercase letters only in the normalized text;

[9] According to `https://help.twitter.com/en/using-twitter/how-to-use-hashtags`, a hashtag written with a # symbol is used to index keywords or topics on Twitter while usernames are marked by the @ symbol. We assume that in our experiments these can be used specifically to find the target or typical topic of abusive comments.

[10] `https://offenedaten-koeln.de/dataset/vornamen`

[11] `http://www.insult.wiki/wiki/Schimpfwort-Liste`

[12] `http://bics.sentimental.li/files/8614/2462/8150/german.lex`

19. Number of matches of words with negative polarity according from list (iii) in the normalized text;
20. Number of matches of words with neutral polarity according from list (iii) in the normalized text;
21. Number of matches of words with positive polarity according from list (iii) in the normalized text;
22. Number of matches of swearwords from list (iv) in the normalized text;
23. Sum of negative sentiment values of matched words from list (v) in the normalized text;
24. Sum of positive sentiment values of matched words from list (v) in the normalized text;
25. Sum of negative sentiment values of matched words from lexicon (vi) in the normalized text;
26. Sum of positive sentiment values of matched words from lexicon (vi) in the normalized text;
27. Sum of neutral sentiment values of matched words from lexicon (vi) in the normalized text.

3.2 Linguistic Analyses

We run the external systems described in Section 2.3 on our data as follows. As system 1. computes word frequencies and determines split points according to a corpus, we add our tokenized tweets to the large German corpus *SdeWaC* (Faaß and Eckart, 2013) and input the combined corpus to the system. We finally use the output to map compounds of tweets to their components before computing word embeddings. The systems 2. and 3. are applied directly to the tokenized tweets.

4 Experiments

In this chapter we report on experiments using our different models on the *GermEval-2018* dataset. Additionally, we mention dataset-specific observations as well as configurations of the neural networks. We present results from a 10-fold cross-validation evaluation on the training data and describe our test runs with references to the filenames containing the respective test data predictions.

The vocabulary given our entire dataset (training and test data) consists of 9,812 distinct tokens with a frequency greater than one. As the embedding matrix contains all weights for all vocabulary entries, our embedding layer has 1,962,400 trainable weights (9,812 x 200 since the word vectors have 200 dimensions).

As the output predictions in the shared task are evaluated based on the macro-average F_1-score measure, which does not take the frequency of each class label into account, we optimized our model to predict all labels uniformly. We achieve this by adding class weights during training which we compute according to the inverse frequencies of the training data labels. Additionally, we smooth the weights by factor 5 to avoid the bias getting too strong, moving the values closer to the neutral weight (1). For binary classification this leads approximately to the following weights: 1.1 for the label "OTHER" and 1.4 for the label "OFFENSE". As the label frequencies are much more imbalanced for the fine-grained classification, we 'smooth' them using a factor of 0.5, effectively doubling the weights in comparison to their inverse frequency with: "PROFANITY": 144.4, "INSULT": 15.9, "ABUSE": 8.1 and "OTHER": 2.0. We want to note that using these weights does not lead to an optimized overall accuracy; however, it helps to balance the F_1-scores over the classes.

Baseline Model: In total the baseline network has 2,130,977 trainable weights for the binary classification (2,131,364 for the fine-grained classification). We find that the model converges during the binary classification experiments already after 4 epochs and during the fine-grained classification experiments after 10 epochs. The results of the 10-fold cross-validation on the training data are given in Table 1. The table shows the label-specific precision, recall and F_1-score values with the overall accuracy and macro-average F_1-score ($F_{1,\text{ macro-avg}}$). All these values are averaged over the 10 folds. The baseline model detects tweets marked to contain offensive language with an F_1-score of 61.88%. Overall the macro-average F_1-score is 71.93% for binary classification. Results for the prediction of the fine-grained classes are given in Table 2. Here, the macro-average F_1-score is considerably lower with 43.24%. Instances annotated with the label "PROFANITY" are most difficult to detect as the model only reaches an F_1-score of 13.21%. Considering all labels, the macro-average F_1-score during the 10-fold cross-validation on our training data of our baseline fine-grained classifier is 43.24%.

We produce the first two test runs using the exact same configurations as during the described cross-validation setting, however, training on the full training data and predicting the given test data (*HIIwiStJS_coarse_1.txt* using the baseline binary classifier and *HIIwiStJS_fine_1.txt* using the baseline fine-grained classifier).

Model	Label "OTHER" precision	recall	F_1-score	Label "OFFENSE" precision	recall	F_1-score	Acc.	$F_{1,\text{ macro-avg}}$
Baseline NN	80.45	83.82	81.94	65.86	59.32	61.88	75.64	71.91
Text&Meta	**81.49**	84.46	**82.92**	**67.09**	**62.19**	**64.45**	**76.98**	**73.69**
Text&Meta&POS	80.03	**85.55**	82.69	66.98	57.85	62.03	76.28	72.36

Table 1: 10-fold cross-validation result scores in % for the binary classification task.

Model	"OTHER" F_1-score	"ABUSE" F_1-score	"INSULT" F_1-score	"PROFANITY" F_1-score	Acc.	$F_{1,\text{ macro-avg}}$
Baseline NN	78.16	**48.24**	33.35	**13.21**	65.24	**43.24**
Text & Meta	**79.14**	48.04	**35.79**	7.57	**66.34**	42.63

Table 2: 10-fold cross-validation result scores in % for the fine-grained classification task.

Text & Metadata Model: The combined network consists of three models which are trained separately: 1. The text sequence-based sub-network which is trained exactly like the baseline network and converges after 4 epochs for binary classification (10 training epochs for the fine-grained classification). 2. The metadata sub-network (with 18,714 trainable weights) we find to converge after 40 epochs for the binary classification (100 training epochs for the fine-grained classification). 3. The combined model (1,028 trainable weights) which uses the pre-computed sub-networks converges after only 4 epochs for the binary classification and 6 epochs for the fine-grained classification.

The results for the binary prediction in the 10-fold cross-validation on the training data are given in Table 1. The evaluation shows that the extended model outperforms the baseline model by 1.78% macro-average F_1-score. Especially the prediction of tweets labeled as containing offensive language is improved with and F_1-score of 64.45% which is 2.57% more than to the baseline result.

The results for of the extended model the fine-grained prediction in the 10-fold cross-validation on the training data are given in Table 2. Here, the overall macro-average F_1-score is 0.61% lower than the score for the baseline while the accuracy is 1.10% higher. Observing the label-specific F_1-scores also shows an unclear pattern, as the values only improve for half of the labels.

We compute two test runs using the text & metadata network using the exact same configurations as during the described cross-validation setting, however, training on the full training data and predicting the given test data (*HIIwiStJS_coarse_2.txt* using the binary classifier and *HIIwiStJS_fine_2.txt* using the fine-grained classifier).

Text & Metadata Model extended with Linguistic Analyses: Early experiments on integrating the linguistic analyses in our described approach led to mixed results. We only report on a few experiments here as the research is still ongoing and most models still require fine-tuning.

Using the compound splitter to normalize tweets substantially reduces the size of vocabulary which speeds up training, however, the performance deteriorates. We assume that it might be necessary to train new word component embeddings using compound splits on a large corpus as initial weights.

Furthermore, simply integrating the sub-networks based on dependency parses does not seem to improve the performance of the model.

Finally, we report the results when using three sub-networks: the two sub-networks of our Text & Metadata model extended by and additional sub-network operating on sequences of POS tags. We train the POS-based sub-network separately for 50 epochs and the combined model for 6 epochs. The evaluation scores of our cross-validation are given in Table 1 for the binary classification. Note that this model reaches the highest recall for the label "OTHER" in comparison to the other models while it performs worse according to all other evaluation scores. The model seems to be overfitted to this label which might signal that this approach is not fully optimized. We compute two final test runs using this Text & Meta & POS-based NN configuration, training on the full training data and predicting the given test data (*HIIwiStJS_coarse_3.txt* using the binary classifier variant and *HIIwiStJS_fine_3.txt* using the fine-grained classifier variant[13]).

[13] We train the POS-based sub-network in 120 epochs.

5 Conclusion and Future Work

In this paper we described our system runs and methods for the identification of abusive language in microposts. When integrating further feature types we observed that fine-tuning the complex neural networks gets much more difficult and time-consuming. However, we manage to improve our baseline model macro-average F_1-score by 1.78% to 73.69% when adding metadata features. Furthermore, we presented early findings on using linguistic annotations in additional neural sub-networks, which requires more optimization steps. We plan to focus in future work on more in-depth analyses on the integration of linguistic annotations into the neural network to further improve the performance of the system in modeling offensive language.

Acknowledgments

We would like to thank Ulrich Heid for his valuable feedback and support.

References

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics.

Fabienne Cap. 2014. *Morphological processing of compounds for statistical machine translation*. Ph.D. thesis, Institute for Natural Language Processing (IMS), University of Stuttgart.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar.

Mark Cieliebak, Jan Milan Deriu, Dominic Egger, and Fatih Uzdilli. 2017. A Twitter corpus and benchmark resources for German sentiment analysis. In *5th International Workshop on Natural Language Processing for Social Media, Boston, MA, USA, December 11, 2017*, pages 45–51. Association for Computational Linguistics.

Fabio Del Vigna, Andrea Cimino, Felice DellOrletta, Marinella Petrocchi, and Maurizio Tesconi. 2017. Hate me, hate me not: Hate speech detection on facebook. In *Proceedings of the First Italian Conference on Cybersecurity (ITASEC17)*, Venice, Italy.

Gertrud Faaß and Kerstin Eckart. 2013. SdeWaC–A corpus of parsable sentences from the web. In *Language processing and knowledge in the Web*, pages 61–68. Springer.

Antigoni-Maria Founta, Despoina Chatzakou, Nicolas Kourtellis, Jeremy Blackburn, Athena Vakali, and Ilias Leontiadis. 2018. A unified deep learning architecture for abuse detection. *CoRR*, abs/1802.00385.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Sebastian Köffer, Dennis M Riehle, Steffen Höhenberger, and Jörg Becker. 2018. Discussing the value of automatic hate speech detection in online debates. In *Multikonferenz Wirtschaftsinformatik (MKWI 2018): Data Driven X - Turning Data in Value*, Leuphana, Germany.

Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153. International World Wide Web Conferences Steering Committee.

Amir H Razavi, Diana Inkpen, Sasha Uritsky, and Stan Matwin. 2010. Offensive language detection using multi-level classification. In *Canadian Conference on Artificial Intelligence*, pages 16–27. Springer.

Robert Remus, Uwe Quasthoff, and Gerhard Heyer. 2010. SentiWS – a Publicly Available German-language Resource for Sentiment Analysis. In *Proceedings of the 7th International Language Resources and Evaluation (LREC)*, pages 1168–1171.

Josef Ruppenhofer, Melanie Siegel, and Michael Wiegand. 2018. Guidelines for IGGSA Shared Task on the Identification of Offensive Language, March 12, 2018.

Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10.

Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *Information Theory Workshop (ITW), 2015 IEEE*, pages 1–5. IEEE.

Ulli Waltinger. 2010. GERMANPOLARITYCLUES: A Lexical Resource for German Sentiment Analysis. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, May. electronic proceedings.