Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019)

TUWienKBS19 at GermEval Task 2, 2019: Ensemble Learning for German Offensive Language Detection

Joaquín Padilla Montani
TU Wien
Institut für Logic and Computation
Favoritenstraße 9-11, 1040 Austria
jpadillamontani@gmail.com

Peter Schüller
TU Wien
Institut für Logic and Computation
Favoritenstraße 9-11, 1040 Austria
ps@kr.tuwien.ac.at

Abstract

The TUWienKBS19 system for German offensive language detection in the GermEval 2019 shared task is a stacking ensemble system. Five disjoint sets of features are used: token and character n-grams, relatedness to the, according to tf-idf, most important tokens and character n-grams within each class, and the average of the embedding vectors of all tokens in a tweet. Several base classifiers are trained independently on each of these features, yielding meta-level features which one maximum entropy model uses to perform the final classification. Our system achieved a macro-averaged F_1-score of $76,80\%$ on subtask I, and $51,86\%$ on subtask II.

1 Introduction

We describe the TUWienKBS19 system that participated in the GermEval Task 2, 2019, Shared Task on the Identification of Offensive Language.

This task is relevant for supporting humans when they moderate online content. In the pseudo-anonymous environment of microposts, abusive language is easily produced by users and it is an important objective to prevent that such content is broadcast to a large number of readers.

Our system is based on a stacked architecture where a set of base level classifiers is trained on a set of five feature groups, and the resulting trained models are forwarded to a meta-level classifier that determines the final outcome of the prediction. This architecture and its training method builds upon the system (Padilla Montani and Schüller, 2018) we submitted to GermEval 2018. This 2018 system was in turn inspired by the EELECTION system (Eger et al., 2017).

This year's system compares to its predecessor (Padilla Montani and Schüller, 2018) as follows:

- Feature extraction was kept as before, although several hyperparameters were readjusted using this year's training data.
- The set of base level classifiers used has been broadened, while keeping training times low.

This paper is organized as outlined below. In Section 2 we give details about the competition subtasks and evaluation metrics. In Section 3 we describe tweet preprocessing and the features we used. In Section 4 we describe the machine learning models we used and the stacked predictor model and we describe how we trained this architecture. Section 5 describes our submitted runs and provides the official competition scores for each run and subtask. We conclude the paper in Section 6.

The source code of our system, i.e., feature computation, training, and classification, is available online.[1]

2 Data and Subtasks

The GermEval Task 2, 2019, Shared Task on the Identification of Offensive Language[2] solicited the submission of systems to automatically classify German language microposts (in a Twitter dataset) with respect to their offensiveness. Such predictions are a valuable tool for assisting human moderators with the job of reducing the amount of hurtful, derogatory or obscene online content.

This year's edition of the Shared Task consisted of three subtasks:

- Subtask I: coarse-grained classification into the two classes "OFFENSE" and "OTHER" (where "OTHER" means non-offensive).
- Subtask II: fine-grained classification into the four classes "PROFANITY", "INSULT", "ABUSE", and "OTHER".

[1] https://github.com/jpadillamontani/germeval2019

[2] https://projects.fzai.h-da.de/iggsa/

- Subtask III: offensive tweets are further classified into the two subcategories "EXPLICIT" and "IMPLICIT".

In each of the subtasks, the classes are mutually exclusive. For example, in subtask II the "PROFANITY" class does not contain any insults, "ABUSE" does not insult a single concrete person but a whole group of people and is also abusive in a way that is not simply "PROFANITY". For further details see also the annotation guidelines (Ruppenhofer et al., 2018).

The submitted systems are evaluated according to the macro-averaged F_1-score of their predictions, i.e. each class contributes equally to the final score, independent from the number of samples in the class.

Our team participated in subtasks I and II. For these two subtasks, the training set contains 12536 tweets, where 8359 are marked as "OTHER" and the remaining ones as "OFFENSE". Of the offensive tweets, 2305 are marked as "ABUSE", 1601 as "INSULT", and 271 as "PROFANITY".

3 Features

We implemented feature extraction using the libraries *scikit-learn* (Pedregosa et al., 2011) for tf–idf computations, *NLTK* (Bird et al., 2009) for tokenization and stemming, and *gensim* (Řehůřek and Sojka, 2010) for managing precomputed word embeddings.

3.1 Preprocessing

Our preprocessing approach first removes all handles (@username) and replaces the special characters "#-_,;:/+)<>&" and line break characters by spaces. The substring "'s" (as in "geht's") is also replaced by a space.

For tokenization we used *NLTK*'s `TweekTokenizer` with `reduceLen=True`. This parameter means that repetitions of the same character are shortened to at most three letters (e.g., "cooooool" is normalized to "coool").

For features for which we applied stemming, *NLTK*'s `GermanStemmer` was utilized.

Table 1 gives an overview of the groups of features we used and the type of preprocessing used in each case. We describe each feature group in the following.

Special Preprocessing indicates which additional preprocessing was done beyond handle removal, special character replacement and tokenization. For creating character-level features we concatenated (Join in Table 1) the resulting tokens with spaces into one string for extracting character-level n-grams, i.e. we always used the tokenizer (even for character-level features) to make use of its `reduceLen` feature.

3.2 Character and Token N-Gram Features

The feature groups CNGR and TNGR are similar, so we describe them together. Both operate on a lowercased version of the input, and TNGR additionally performs stemming on each token.

CNGR extracts all character-level n-grams of length 3 to 7, while TNGR extracts all stemmed token-level n-grams of length 1 to 3.

In both cases, we performed tf-idf over all extracted n-grams. Only n-grams with a document frequency between 0.01 and 0.0002 at the token level, and only those with a document frequency between 0.02 and 0.0001 at the character level were kept (i.e., those that are rare enough to carry some signal, but frequent enough to have a potential to generalize over unseen data). The described document frequency thresholds were tuned by means of a grid search on a 90%/10% split of the training data, with the aim to maximize prediction scores of the base classifiers (see Section 4).

We used the tf-idf score of the relevant n-grams as input features (realized with *scikit-learn*'s `TfidfVectorizer`).

3.3 Word Embedding Features

We used a pretrained word2vec-style skip gram word embedding with 100 dimensions and window size 5, created from a large collection of German language tweets from the years 2013 to 2017 by Heidelberg University.[3]

For each tweet, we created 100 real-valued features by taking the average embedding of all tokens in the tweet, normalized to unit length with the ℓ^2-norm.

Whenever a word embedding is required, i.e., for feature groups TIMP and EMB, and whenever the token is not in the vocabulary of the pretrained list of word embeddings, we performed a fallback operation. We searched for the largest prefix and the largest suffix of the token of length 3 or greater where we know a word embedding. If we find such

[3] `http://www.cl.uni-heidelberg.de/english/research/downloads/resource_pages/GermanTwitterEmbeddings/GermanTwitterEmbeddings_data.shtml`

Symbol	Name	Level	Special Preprocessing	Word Embeddings
CNGR	Character N-Grams	C	Lowercase + Join	-
CIMP	Important N-Grams	C	Join	-
TNGR	Token N-Grams	T	Lowercase + Stemming	-
TIMP	Important Tokens	T	-	min/max cos distance
EMB	Word Embeddings	T	-	average

Table 1: Groups of features used for classification. Handle removal, special character replacement and tokenization is used for all features. C and T stand for character and token level, respectively.

Subtask	m	Feature	k
I	2	CIMP	2000
I	2	TIMP	2500
II	4	CIMP	500
II	4	TIMP	1000

Table 2: Number of important types selected for each subtask and feature group.

affixes with embeddings, we use the embeddings of these affixes as if they were separate tokens in the tweet.

As an example, the word "Nichtdeutsche" (non-Germans) in the dataset does not exist in some pretrained word embedding models, so we encounter an OOV (out-of-vocabulary) exception. Our method would use as a fallback two word embeddings for affixes "Nicht" (not) and "deutsche" (German+Adj) because both affixes are present in the word embedding model. This fallback significantly reduced the number of OOV exceptions when extracting these features.

3.4 Important N-Gram and Token Features

These two groups of features are based on the same idea: to perform tf-idf over the whole dataset, select the k most important types relative to each of the m classes ($m = 2$ in subtask I, $m = 4$ in subtask II). We determine importance by ranking features according to their average tf-idf value in all documents in the respective class. Based on the resulting list of $k \cdot m$ most important type/class combinations we create a feature for each $k \cdot m$ combination. For CIMP each type is a character n-gram, while for TIMP each type is a token. Intuitively this selects the most distinguishing types per category.

Table 2 shows the number of important types selected for each subtask and each feature group. These values were adjusted with a grid search on a 90%/10% split of the training data in order to maximize prediction scores of the base classifiers (see Section 4).

So far we have only discussed how important types are selected. We next describe which features are generated from these important types.

For TIMP, for each important type t in a tweet we obtain its word embedding $\vec{t}$ and compute the maximum and the minimum cosine distance from $\vec{t}$ to all other embeddings of other types in the same tweet. We use the same OOV-fallback described in Section 3.3. This yields a minimum and a maximum feature for each important type and each class: $2 \cdot k \cdot m$ real features for each tweet.

For CIMP we have no embedding information, therefore we create for each important type t a Boolean feature that indicates whether t is contained in the tweet or not. This yields a feature for each important type and each class: $k \cdot m$ Boolean features for each tweet.

By creating a set of features for each class, we increase the signal that can be learned for the "PROFANITY" class in subtask II, since this class contains a very small set of samples.

4 Classification

Our system is a stacking ensemble which builds upon the system (Padilla Montani and Schüller, 2018) we submitted to GermEval 2018. That system in turn was inspired by the EELECTION system of Eger et al. (2017).

We implemented most of the classification using the library *scikit-learn* (Pedregosa et al., 2011) and refer to class and function names of scikit-learn in the following (unless explicitly stated otherwise).

4.1 Base Classifiers

For each subtask and each of the 5 feature groups discussed in Section 3, we independently trained a varying number of base classifiers, selected out of:

Subtask	Feature	Base Classifiers
I	CNGR	ete, etg, lre, mnb, gbm
I	TNGR	ete, etg, lre, mnb
I	CIMP	lre, mnb, gbm
I	TIMP	ete, etg
I	EMB	ete, etg, lre, gbm
II	CNGR	ete, etg, lre, mnb
II	TNGR	ete, etg, lre, mnb
II	CIMP	lre, mnb, gbm
II	TIMP	ete, etg
II	EMB	ete, etg, lre, gbm

Table 3: Base classifiers used for each subtask and each feature group.

(ete) an ensemble of random forests trained on samples of the training set (Geurts et al., 2006) using information gain as criterion for scoring the sample splits (class `ExtraTreesClassifier` with `criterion=entropy`),

(etg) another ensemble of random forests trained using Gini impurity for scoring sample splits (class `ExtraTreesClassifier` with `criterion=gini`),

(lre) a MaxEnt model with balanced class weights (class `LogisticRegression`),

(mnb) a multinomial naive Bayes classifier (class `MultinomialNB`), and

(gbm) a gradient boosting model with decision trees as base learners from the library *LightGBM* (Ke et al., 2017) (class `LGBMClassifier`).

We incorporated in the ensemble the base classifiers from the above list which were able to achieve good individual cross-validation performance after fine tuning and were also relatively fast to train. Table 3 details, for each subtask and feature group, which of the base classifiers were used. We trained a total of 18 base level classifiers in subtask I, and 17 in subtask II.

Each base classifier was trained on 90% of the training data, and used to predict class probabilities on the remaining 10%. We performed this process 10 times in a cross-validation manner to obtain predictions for all tweets in the training data. Furthermore, we then also trained each base classifier on the whole training data, and used this models to predict class probabilities for the test data. This process generates the meta-level features that are used by the meta classifier, as described in the following section.

4.2 Meta Classifier

For subtask I we generated 36 meta-level features per tweet, using the probabilistic class predictions of 18 base classifiers (two classes). In subtask II we have 68 meta-level features per tweet, according to the probabilistic class predictions of 17 base classifiers (four classes).

On these features and the known true classes of the training tweets we trained a maximum entropy model (class `LogisticRegression`). We used balanced class weights and fine tuned the C parameter for each subtask using stratified cross validation, i.e. ensuring stable class ratios in each fold.

5 Submission

We submitted three runs for subtask I and three runs for subtask II, named `TUWienKBS19_coarse_#.txt` and `TUWienKBS19_fine_#.txt`, respectively, where # is the run number (1, 2 or 3).

Run 2 corresponds to the full ensemble system as described in the previous sections. Run 1 differs from run 2 in that the CIMP features (and the associated base classifiers) were disabled, since in our pre-competition testing of the ensemble using cross validation we obtained the best results when not using these features. Training these ensemble systems takes around 10 minutes for each subtask and for each run, on a regular desktop computer.

Run 3 is a lightweight single model, namely a MaxEnt model with balanced class weights (class `LogisticRegression`) trained on our best performing group of features: character level n-grams (CNGR). This run only takes a few seconds of training per subtask.

Table 4 shows the official macro-averaged F_1-score on the testing data for each of our submitted runs.

6 Conclusion

We presented the TUWienKBS19 submission to the GermEval Task 2, 2019, Shared Task on the Identification of Offensive Language. We submitted runs

Subtask	Run	F_1-score
I	1	**76,80**
I	2	76,75
I	3	71,42
II	1	51,23
II	2	**51,86**
II	3	46,94

Table 4: Official results of our system runs.

for subtask I (binary classification) and subtask II (fine-grained classification). Our approach used a stacking ensemble system which was based on our submission from last year's shared task. We utilized five groups of features, some operating at the token level and some at the character level, and we also made extensive use of pretrained word embeddings.

Our system is built with a major challenge from this competition in mind: the evaluation mode in combination with the class imbalance in the training data. The competition evaluation uses macro-averaging, i.e., each class counts the same. At the same time, in subtask II, there is one class ("PROFANITY") with only 271 tweets as samples within a training set which contains 12536 tweets. Our tuning efforts were focused on managing this class imbalance.

Acknowledgement

We thank the organizers of the competition. This work has received financial support from the European Union's Horizon 2020 research and innovation programme under grant agreement 825619 (AI4EU).

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Steffen Eger, Erik-Lân Do Dinh, Ilia Kutsnezov, Masoud Kiaeeha, and Iryna Gurevych. 2017. EELECTION at SemEval-2017 Task 10: Ensemble of nEural Learners for kEyphrase ClassificaTION. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017)*, pages 942–946.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning*, 63(1):3–42.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NIPS 2017)*, 30:3146–3154.

Joaquín Padilla Montani and Peter Schüller. 2018. TUWienKBS at GermEval 2018: German Abusive Tweet Detection. In *Proceedings of GermEval 2018, 14th Conference on Natural Language Processing (KONVENS 2018)*, pages 45–50.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.

Josef Ruppenhofer, Melanie Siegel, and Michael Wiegand. 2018. Guidelines for IGGSA Shared Task on the identification of offensive language. `http://www.coli.uni-saarland.de/~miwieg/Germeval/guidelines-iggsa-shared.pdf`.