TUWienKBS at GermEval 2018: German Abusive Tweet 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 TUWienKBS system for abusive tweet detection in the GermEval 2018 competition is a stacked classifier. Five disjoint sets of features are used: token and character n-grams, relatedness to the, according to TFIDF, most important tokens and character n-grams within each class, and the average of the embedding vectors of all tokens in a tweet. Three base classifiers (maximum entropy and two random forest ensembles) are trained independently on each of these features, which yields 15 predictions for the type and/or level of abusiveness of the given tweets. One maximum entropy meta-level classifier performs the final classification. As word embedding fallback for out-of-vocabulary tokens we use the embeddings of the largest prefix and suffix of the token, if such embeddings can be found.

1 Introduction

We describe the TUWienKBS system that participated in the GermEval 2018 competition for abusive tweet detection.

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 three types of classifiers is trained on a set of five feature groups, and the resulting fifteen trained models are forwarded to a meta-level classifier that decides the final outcome of the prediction. This architecture and its training method is inspired by the EELECTION system (Eger et al., 2017) however our features and classifiers are different.

In particular we produce features based on a per-class selection of, according to TFIDF, most important characteristics of tokens and character n-grams. For each class we create the same number of features in this feature category and we found that this helps with the imbalanced training set of 5009 tweets where one of four classes to be predicted contains only 71 samples.

This paper is organized as follows. In Section 2 we give details about the competition tasks and evaluation metrics. In Section 3 we describe tweet preprocessing and the features we use. In Section 4 we describe the machine learning model we use and the stacked predictor model and we describe how we train this architecture. Section 5 describes our submission files and provides an evaluation of our model and features on training data. In Section 6 we describe additional experiments we performed that did not yield improvements of scores. We conclude the paper in Section 7.

2 Competition Tasks

The GermEval 2018 Shared Task on the Identification of Offensive Language[1] solicited the submission of systems that classify German 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.

The competition contained two tasks:

- Task 1: coarse-grained classification into the two classes "OFFENSIVE" and "OTHER" (where "OTHER" means non-offensive), and
- Task 2: fine-grained classification into the four classes "PROFANITY", "INSULT", "ABUSE", and "OTHER".

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

Proceedings of GermEval 2018, 14th Conference on Natural Language Processing (KONVENS 2018)
Vienna, Austria – September 21, 2018

Each micropost is tagged with exactly one class of Task 1 and with exactly one class of Task 2. The classes are mutually exclusive, in particular, 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", see also the annotation guidelines (Ruppenhofer et al., 2018).

The competition evaluation uses macro-averaging of the F1-score of the predictions as final score, i.e., each class contributes equally to the final score independent from the number of samples in the class. The training set contains 5009 tweets where 3321 are marked as "OTHER" and the remaining ones as "OFFENSE" in Task 1. Of the offensive tweets, 1022 are marked as "ABUSE", 595 as "INSULT", and 71 as "PROFANITY".

This imbalance in the training set gave rise to several decisions we made while creating our system, we discuss these in particular in Sections 3.4, 6.3, and 7.

3 Features

We implemented feature computation using the libraries Scikit-learn (Pedregosa et al., 2011) for TFIDF computations, NLTK (Bird et al., 2009) for tokenization and stemming, and GenSim (Řehůřek and Sojka, 2010) for managing precomputed word embeddings.

3.1 Preprocessing

Tweet preprocessing removes all handles (@username) and replaces the characters "#-_,;:/+)<>&" and line break characters by spaces and we replace the substring "'s" (as in "geht's") by a space.

We use NLTK's `TweekTokenizer` with `reduceLen=True`. That means repetitions of the same character are shortened to at most three letters (e.g., "cooool" is normalized to "coool").

For features with stemming we use the German stemmer of NLTK.

Table 1 gives an overview of the groups of features we use. We describe these in the following.

Special Preprocessing indicates which additional preprocessing is done beyond handle removal and tokenization. For creating character-level features we concatenate (Join in Table 1) the resulting tokens with spaces into one string for extracting character-level n-grams. We always use 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 perform TFIDF over all extracted n-grams, keep only those with a document frequency between 0.01 and 0.0002 (i.e., those that are rare enough to carry some signal, but frequent enough to have a potential to generalize over unseen data). The 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 use the TFIDF score of the relevant n-grams as input features (realized with `TfidfVectorizer`).

3.3 Word Embedding Features

We use the pretrained word2vec model `twitter-de_d100_w5_min10.bin` with 100 dimensions and window size 5, created from Twitter data of 2013–2017 by Josef Ruppenhofer.[2]

For each tweet, we create 100 real-valued features by taking the average embedding of all tokens in the tweet, normalized to unit length with L2 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 perform a fallback operation. We search 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 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 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 reduces the number of OOV exceptions in the training set from 1903 to 90 and in the

[2] `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 and tokenization is used for all features. C and T stand for character and token level, respectively.

Task	m	Feature	k
Task 1	2	CIMP	3200
Task 1	2	TIMP	1250
Task 2	4	CIMP	370
Task 2	4	TIMP	170

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

testing set from 1069 to 51.

We also experimented with other pretrained word embedding models but none of them achieved a comparable performance. Combining above mentioned word embeddings with other embeddings increased performance in single classifiers, however it decreased performance when these models were used as part of the ensemble described in Section 4.

3.4 Important N-Gram and Token Features

These two groups of features are based on the same idea: to perform TFIDF over the whole dataset, select the k most important types relative to each of the m classes ($m = 2$ in Task 1, $m = 4$ in Task 2). We determine importance by ranking features according to their average TFIDF 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, a related analysis is described in the blog of Thomas Buhrman.[3]

Table 2 shows the number of important types selected for each task 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 next section).

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 Task 2 which contains a small set of samples.

4 Classification

Our system is a stacked ensemble system inspired by the EELECTION system of Eger et al. (2017).

We implemented most 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 of the 5 feature groups discussed in Section 3, we train three independent classifiers:

- a MaxEnt model with balanced class weight (class `LogisticRegression`),

[3] https://buhrmann.github.io/tfidf-analysis.html

Proceedings of GermEval 2018, 14th Conference on Natural Language Processing (KONVENS 2018)
Vienna, Austria – September 21, 2018

- 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`), and
- another ensemble of random forests trained using Gini impurity for scoring sample splits (`criterion=gini`).

For `ExtraTreesClassifier` we use 100 and 150 estimators for Task 1 and Task 2, respectively. This yields $5 \cdot 3$ distinct *base classifiers*, i.e., feature/classifier combinations.

We train each base classifier on 90% of the training data and perform predictions on the remaining 10%. We perform this process 10 times in a cross-validation manner to obtain predictions for the whole training data. To obtain more reliable results we repeat the whole process 10 times with different random seeds for determining the cross-validation folds. At that point we have 15 base classifiers and their predictions for each tweet and each class in the training data.

4.2 Meta Classifier

Using predictions of 15 base classifiers for each class, we create 30 meta level features per tweet for Task 1 (two classes) and 60 meta level features per tweet for Task 2 (four classes).

On these features and the known true classes we train a maximum entropy model (`LogisticRegression`). We use one-vs-rest classification, balanced class weights, and tuned parameter $C = 0.17$ for Task 1 and $C = 0.2$ for Task 2.

5 Submission and Pre-Competition Evaluation

We submitted a single run for Task 1 and a single run for Task 2 to the competition, named `TUWienKBS_coarse_1.txt` and `TUWienKBS_fine_1.txt`, respectively.

The source code of our system, i.e., feature computation, training, and classification, is available online.[4]

Based on 10-fold cross-validation with stratified folds (i.e., ensuring stable class ratios in each fold) we performed a pre-competition evaluation of our system and its features on the training data. Table 3 shows that the ensemble achieves a macro-averaged F1 score of 81.72 on the training data for coarse-grained prediction (Task 1), where the most important feature in the ensemble is TIMP, followed by CNGR. Table 4 shows results for fine-grained prediction (Task 2) where the ensemble obtains a score of 61.89, again with TIMP and CNGR as most important feature groups in the ensemble.

Features	F1 score	F1 reduction
ALL	81.72	
without TIMP	79.92	1.80
without CNGR	81.14	0.58
without CIMP	81.15	0.57
without TNGR	81.43	0.29
without EMB	81.69	0.03

Table 3: Evaluation on training set (Task 1).

Features	F1 score	F1 reduction
ALL	61.89	
without TIMP	59.77	2.12
without CNGR	60.24	1.65
without CIMP	60.43	1.46
without EMB	61.61	0.28
without TNGR	61.67	0.22

Table 4: Evaluation on training set (Task 2).

Across both tasks we can say that CNGR features are more useful than TNGR features, however with TIMP and CIMP the situation is reversed: TIMP is the most important feature group in both tasks. The reason is that TIMP uses word embeddings and min/max cosine distances from important types to tokens in the tweet at hand, while CIMP only uses membership in the tweet. This makes TIMP a more powerful feature than CIMP.

EMB and TNGR features contribute only little to the overall ensemble score. If we would use only EMB features to predict the class this would yield reasonable results even without an ensemble, while we would obtain worse results when using only TNGR features (these results are not shown in tables).

Altogether, word embeddings are a crucial component of our system and we use them in different ways in TIMP and EMB feature groups.

[4] `https://github.com/jpadillamontani/germeval2018`

6 What did not work?

While creating our submission for the competition we experimented with several methods that did not improve the system score.

6.1 Feature Selection for Character N-grams

Generating n-grams of length 3–7 at the character level yields more than 200.000 features. We tried to reduce this with several feature selection functionalities implemented in `feature_selection.SelectKBest` in Scikit-learn (Pedregosa et al., 2011) with χ^2 (`chi2`) or ANOVA (`f_classif`) as feature scoring functions. Any reduction in the dimensionality impacted the score negatively.

6.2 Stop Word Removal

We removed stop words from the German stop word list of NLTK ("stopwords corpus") and from another list of publicly available German stopwords.[5]

Stop word removal impacted the score negatively, when applying stop word removal before computing token n-grams as well as when applying stop word removal before the computation of tweet embeddings.

6.3 Under- and Oversampling

To overcome the imbalance in the training set (see also Section 2) we tried several sampling methods for re-balancing the dataset.

We used the `imblearn` package[6] (Lemaître et al., 2017) in particular the classes `RandomUnderSampler` and `RandomOverSampler` to undersample the largest class (OTHER) and to oversample the smallest class (PROFANITY), respectively.

This decreased evaluation scores.

6.4 Deep Learning

To our ensemble we added three architectures based on Keras and TensorFlow.[7] These experiments replicated successful approaches for tweet classification and applied them to the GermEval dataset.

In particular we tried the LSTM and CNN methods[8] of Badjatiya et al. (2017) and the Convolution+GRU method[9] of Zhang et al. (2018).

We trained these models, evaluated them individually and also integrated them into the ensemble (we trained the probabilities as we did for the other classifiers).

All three architectures performed similarly to the other, classical, classifiers, reaching F1 scores around 76 for Task 1 and around 55 for Task 2. Adding these deep learning classifiers to the ensemble decreased its overall score, so in the end we excluded them from the ensemble.

6.5 Using sent2vec instead of word2vec

Instead of using word2vec pretrained word embeddings, we experimented with sent2vec[10] models of Lee et al. (2017).

The features generated this way scored significantly worse than the normal averaging of word2vec embeddings. We also tried combining both word2vec and sent2vec features by simply concatenating their vectors, but this still performed worse than the averaging approach we used in the final submission.

7 Conclusion

Our system combines existing approaches that have been reported to work and includes a group of features that is, to the best of our knowledge, novel: the group of "Important N-Gram and Token Features" (Section 3.4). These features (TIMP and CIMP) are generated from the most important (according to the average of their TFIDF scores) tokens (respectively, character n-grams) and this importance is computed within each class that we aim to predict. Essentially, we identify features that are most suitable for distinguishing documents *within each class* and not across classes. Our experiments showed that these features on their own already obtain high prediction scores on both tasks. In the ensemble, feature group TIMP causes the largest drop in prediction score when removed, making it an important component of the prediction.

A major challenge in this competition was the evaluation mode in combination with the class imbalance in the training data. The competition evalu-

[5] `https://github.com/stopwords-iso/stopwords-de/blob/master/stopwords-de.txt`

[6] `http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html`

[7] `https://www.tensorflow.org/guide/keras`

[8] `https://github.com/pinkeshbadjatiya/twitter-hatespeech/`

[9] `https://github.com/ziqizhang/chase`

[10] `https://github.com/UKPLab/germeval2017-sentiment-detection`

ation uses macro-averaging, i.e., each class counts the same. At the same time, in Task 2, there is one class ("PROFANITY") with only 72 tweets as samples within a training set which contains 5009 tweets. Due to this imbalance, making mistakes in this one class has a higher weight on the result than making mistakes in other classes, and we focused our tuning efforts on managing this class imbalance (partially, but not exclusively, by creating the above mentioned class-wise important features).

References

Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly.

Steffen Eger, Erik-Ln 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.

Ji-Ung Lee, Steffen Eger, Johannes Daxenberger, and Iryna Gurevych. 2017. UKP TU-DA at GermEval 2017: Deep learning for aspect based sentiment detection. In *Proceedings of the GSCL GermEval Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*, pages 22–29.

Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

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`.

Ziqi Zhang, David Robinson, and Jonathan Tepper. 2018. Detecting hate speech on twitter using a convolution-GRU based deep neural network. In *The Semantic Web*, pages 745–760.