Changes in Baselines of Hatespeech Detection with pretraining and Transformers

Anonymous ACL-IJCNLP submission

Contents

Abstract

In this project we aim to construct a new representative baseline for Deep Neural Nets on the HatefulTwitter dataset proposed by [Davidson et al., 2017]. For this we implement a Transformer-based Architecture on the data and try to push the baselines proposed by the authors of the HatefulTwitter dataset. Then we will aim to provide some evaluation of explainability and performance, also with respect to industry usecases.

1 Introduction

In their paper [Davidson et al., 2017] propose a tweet dataset with 25.000 entries and use some basic methods to establish baselines, especially interesting is that BoW approaches seem to have major problems distinguishing between hatespeech and offensive language since the distributions are so similar. This might indicate that context sensitive methods are to be used. For example Transformers/BERT-models yield such a functionality. As a convention we will use **bold font** to refer to codebase related entities, for example: ”...in the notebook **data_pipeline** one finds the function **preprocess_tweet** which handels preprocessing”.

1.1 The Research Question

2 Review of [Davidson et al., 2017]

In this section the central question: ”What did Davidson et al. actually do?” and what are the results against which we will benchmark.

2.1 the Model

The jist of it is the following: They use a dataset with about 25.000 tweets. Then use Tidf-vectorizers on the tokens and on the POS-tags of the tweet to get a vectorized version of the tweet on the syntactic and on the word-level. Then calculate a few more features based on reading-ease levels, counts of hashtags,mentions,urls and sentiment. All these get fed into a feature selecter via logistic regression with L1-regularization (using the liblinear solver) for dimensionality reduction and then predictions are generated using logistic regression.

2.2 the Results

[Davidson et al., 2017] claims that their final model has an overall precision of 0.91, an overall recall of 0.9 and a f1-score of 0.9. They also claim a precision of 0.44 and recall of 0.61 (which implies a F1-score of about 0.525) on the hate class. These claims come with an asteriks, because these numbers do not represent the performance on a test set but rather using the entire data to train and evaluating effectively on the trainingset. It is impossible to really judge how much overfitting inflates the numbers here. What we were able to replicate is a model which achieves overall precision 0.84, recall of 0.75 and F1-score of 0.78 on a testtest of 10% of the total data using weighted averages, which is significantly worse, than the reported values. And a precision of 0.21, recall of 0.6 and an F1-score of 0.31 on the hateclass in the testset. These values are to be taken with caution since we don’t cross validate here, so there is a chance this is sampling bias.

Since we aim to compare the embeddings only we reran the sourcecode of Davidson with just the vectorizers on the tweet and the pos-tags. This yielded a model with overall precision of 0.85, recall of 0.74, F1-score of 0.78 and a precision of 0.20, a recall of 0.61 and a F1-score of 0.3 on the hateclass again evaluated on 10% of the data as a test set. Which again seems odd since sentiment features should yield a bunch of interesting information for the feature selection.

Table 1: classification report for our reporduction of Davidsons model with features

class	precision	recall	f1-score	support
hateful	0.21	0.6	0.31	134.0
offensive	0.94	0.74	0.83	1900.0
neither	0.61	0.81	0.69	446.0
metric	precision	recall	f1-score	support
accuracy	nan	nan	0.75	2480.0
macro	0.59	0.72	0.61	2480.0
weighted	0.84	0.75	0.78	2480.0

Table 2: classification report for our reporduction of Davidsons model without features

class	precision	recall	f1-score	support
hateful	0.2	0.61	0.3	134.0
offensive	0.95	0.73	0.83	1900.0
neither	0.62	0.83	0.71	446.0
metric	precision	recall	f1-score	support
accuracy	nan	nan	0.74	2480.0
macro	0.59	0.72	0.61	2480.0
weighted	0.85	0.74	0.78	2480.0

The confusion matricies can be seen in 1 and 2.

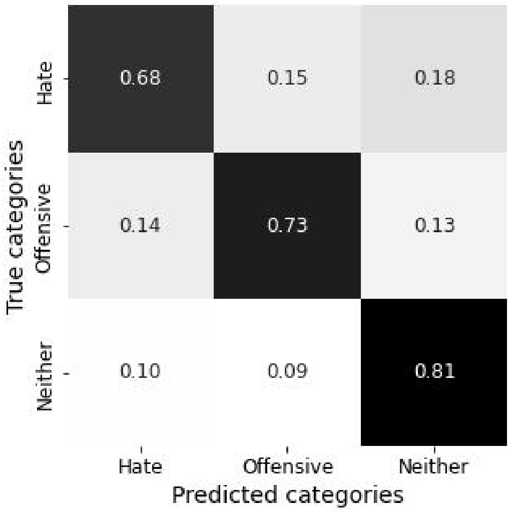

Figure 1: Confusion of our version of Davidsons model with features

3 The Model and Implementation

3.1 The preprocessing

For preprocessing of the Tweets we used the Spacy **en_core_web_sm** pipeline with **EntityRuler** Component added. Which uses identical RegEx to Davidson to identify urls mentions and excess whitespace. Then the entire tweet is lemmatized via the **Lemmatizer** component in the **en_core_web_sm** pipeline and urls are replaced with a ”[URL]” token, as well as mentions being replaced by a ”[MENTION]” token. The entire procedure is handeled in the function **preprocess_tweet** in the **data_pipeline** notebook. Here some possible improvements would be to also have a dedicated ”[HASHTAG]” token as well as dealing better with elongated versions or slightly modified versions of the same tokens. For example: turning ”Shuuuuuuuuut upppp” or ”$hut up” into ”shut up”.

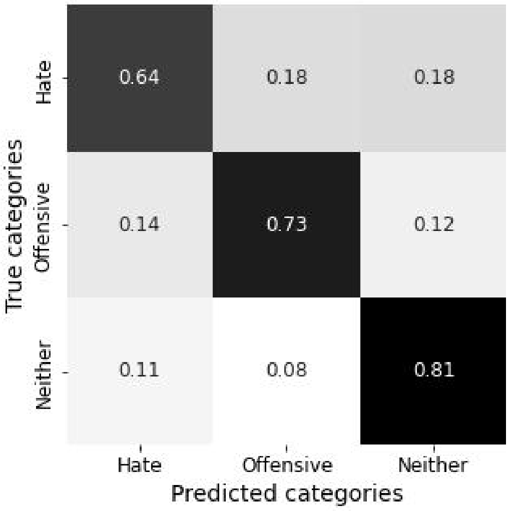

Figure 2: Confusion of our version of Davidsons model with no features

3.2 The Model

For the model we used a **distil-bert-uncased** pretrained Distilbert model with a classification head for finetuning using the Huggingface **AutoModelForSequenceClassification** class. In addition we used the appropriate BERT-WordPiece tokenizer adding two custom tokens ([URL],[MENTION]) to the tokenizer. The model gets called by the **Trainer** class via a **model_init** method, since otherwise the classification head weights are not connected to the **seed** parameter of the **Trainer** class. This turns out to be a great problem to get reproducable results. Especially in our experiments, this turned out to be a problem, since the best model we trained was not able to be replicated, with the set seed since, we simply got lucky with the classification head weights. Which is exactly why we now use a **model_init** method. And also why it shouldn’t be expected to get the exact weights we got once the training procedure is followed. Which is why the Pytorch model is provided completely in the repository to make the results reproducable.

3.3 Finetuning

We used a custom lossfunction to mediate classimbalances. For this we subclassed the **Trainer** class and overwrote the **compute_loss** function with a custom function. We use CategoricalCrossEntropy with weights for each class given by:

$$w_i = 1 - \frac{count(class_i)}{count(data)}$$

For training we used the following Hyperparameters: First we finetuned with:

- learning_rate = 1e-5
- per_device_train_batch_size = 16
- per_device_test_batch_size = 16
- num_train_epochs = 5
- weight_decay=0.01
- evalution_strategy = ”steps”
- eval_steps = 500
- fp16 = True
- load_best_model_at_end = True
- metric_for_best_model = eval_hatef1
- seed = 42

Then we finetuned another pass with changes only in:

- learning_rate = 5e-6
- num_train_epochs = 2
- seed = 42

the rest of the unspecified hyperparameters were the default value or are paths/logging settings which aren't training relevant and can be looked up in the **train_model** notebook.

3.4 Results

This yielded the following results using sklearns **classification_report**, on a test set of 10% of the total data.

Table 3: Classification report for our finetuned model without external features

class	precision	recall	f1-score	support
hateful	0.58	0.75	0.66	134.0
offensive	0.97	0.96	0.96	1899.0
neither	0.96	0.95	0.96	446.0
metric	precision	recall	f1-score	support
accuracy	nan	nan	0.94	2479.0
macro	0.84	0.89	0.86	2479.0
weighted	0.95	0.94	0.95	2479.0

3 shows that one can expect a relatively basic transformer model to outperform the baseline proposed by [Davidson et al., 2017]. The most significant improvement on the hateclass, that is pretty stable, under multiple training runs (different seeds and random classifier head weights) is the precision. It is not uncommon to see with finetuning for a few epochs a precision of 0.5 and a recall of 0.5 on the hateclass of the testset. This still is a 0.06 improvement in precision over Davidsons reported numbers (which are evaluated on the trainingset). Here one sees a form of Precision/Recall Trade-off.

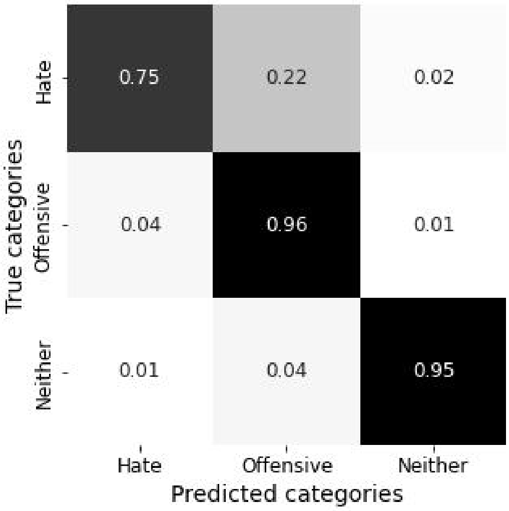

Figure 3: Confusion of the finetuned model

4 Evaluation

4.1 Comparison between [Davidson et al., 2017] and our model

4.2 What did the model learn

4.3 Cost of Implementation and viability for industry purposes

4.4 Conclusions

References

T. Davidson, D. Warmsley, M. W. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. *CoRR*, abs/1703.04009, 2017. URL `http://arxiv.org/abs/1703.04009`.