Hate speech identification and classification

Final Project Report

NLP4Web

by

Oliver Wandschneider (7048312)
Finn Carstensen (7065446)

Table of contents

Research problem

Today, throughout the internet there still exists a lot of insults, hate speech and other inappropriate or offensive content, referred to in the further report generally as hate speech. That is why the big social network companies are consistently increasing resources to find those contents. As this topic still has an obvious high relevance, we wanted to investigate in finding hate speech on the web ourselves.
With existing annotated data from various sources, our goal was to build a binary classifier that determines if a post, sentence or text is hate speech or not. In addition to that, if the text is hate speech, we wanted to categorize it in different subtopics of hate speech (e.g., racism, sexism etc.).

Data sources

For testing and training our neural network classifiers, we needed a big amount of annotated data. Since we wanted to concentrate on building the classifiers, we did not try to find and annotate data ourselves. There are quite a lot of projects out in the world that had a similar or related research goal, so we collected a handful of datasets that fitted our requirements the best. They are in different shapes and contain different data that helps identify if a text is hate speech and if so which category it belongs to, so we needed to do some preprocessing/standardization here. We decided on a standard data format that we wanted to use in our classifiers: Every single kind of text has additional information for the binary decision if it is hate speech or not, and the category it belongs to.
We also wanted to try out hate speech related keywords for our classifier, for that we also collected different datasets. We combined them into one list of words containing no duplicates. In the following, we will briefly describe which datasets we used and how we made them applicable for our task. All datasets can be found in the datasets folder in our project repository. The code that handles the standardardization for the individual datasets can be found in the standardization folder. Table 1 shows a brief description and the existing attributes of the original data and what we did as standardization to make the data usable.

Filename	**Description**	**Original format (CSV columns)**	**Standardization**
1_labeled_data.csv	Collection of posts, annotated by multiple people. The annotators had to decide whether a given text is hate speech, offensive language or neither of them.	,count,hate_speech,offensive_language,neither,class,tweet	Translated to a binary decision if it is hate speech or not. If the majority of annotators decided on hate speech or offensive language, we considered the text as hate speech.
3_en_dataset_with_stop_words.csv	Collection of posts, annotated with readers sentiment and proposed hate speech target (as well as other information)	HITId,tweet,sentiment,directness,annotator_sentiment,target,group	Use sentiments and targets to decide if a post is hate speech and use the target for categorization
5_train.csv	Huge collection of texts that was given for a competition for hate speech classifiers. Multiple binary decisions for different categories exist.	id,comment_text,toxic,severe_toxic,obscene,threat,insult,identity_h	If the text was annotated with any hate speech category, consider it as hate speech and bring the category to our

		ate	format. Multiple categories mean multiple entries in our dataset. No category means no hate speech.
6_NAACL_SRW_2016.csv	IDs of Twitter posts with a hate speech class (racism, sexism, none). Text not included. Not all posts for given IDs still exist, many seem to be already removed	id,class	Fetch post texts from Twitter API if present. If class is sexism or racism, consider it as hate speech.
4_harassment-corpus.csv	Collection of words that are related to different hate speech categories	Sexual,Racial,Appearance,Intelligence,Politics,Generic	Combine words from all categories to one collection for binary hate speech classification
7_bad-words.csv	Collection of bad words in english language	word	Use as presented
8_Terms-to-Block.csv	Collection of bad words in english language	word	Remove punctuation

Table 1: Datasets

Methods used

With the datasets we did some preprocessing, which is common in any natural language processing task. This step is based on knowledge gained from the NLP4Web course. We did not do much of tokenization or token labeling, because these steps seemed not necessary or useful for our specific task.
For model engineering, we used Jupyter Notebooks, because they give us a very handy way of doing steps one by one and always being able to use the previously calculated results. In addition to that, they are much more readable than simple commented source code. For computing, we had the advantage of having a CUDA-capable GPU that reduced computation times a lot. We did all the computing locally.
Our method of choice for creating the aforementioned classifiers was using Neural Networks by utilizing the Tensorflow package for Python. To get started with the task and to develop a feeling for the general approaches we looked into a Tensorflow tutorial that deals with sentiment analysis using an RNN with LSTM-layers and vector-embeddings. We used parts of the tutorial for our own classifier. We then iteratively improved our model by changing the input, model parameters and model architecture.
To create a nice little user interface, for showing and testing the performance of our models, we used the Flask Python library.

Text preprocessing

After we brought our different datasets in the same CSV format (standardization), we needed to preprocess the text to remove artifacts from the web as well as unneeded information. The not preprocessed datasets can be found in our datasets folder as "<nr>_standard.csv". A spot check of the entries in the datasets shows that there are quite a lot of symbols and strings that will most probably not be useful for training a model. It also seems like most annotations are understandable and appropriate for our task.

The preprocessing pipeline consists of multiple steps. At first, we remove any mentions of users for post data by removing any word that is led by an “@” symbol. Then we remove any encoded symbols that are present in some datasets and do not represent words, in the form of “&#<code>”. Also links are unwanted, so we remove any kind of links to webpages. No, since we removed all elements containing punctuation, we can remove the rest of punctuation from the text. Finally, the text is transformed to lowercase and whitespaces are stripped. The results are stored as “<nr>_preprocessed.csv” in our datasets folder.
As we wanted to find the best configuration for our classifier model, we also removed all the stopwords in a different file. The files with names “<nr>_no_stopwords.csv” were sent through the described preprocessing pipeline and then, using the NLTK stopwords corpus, freed from all the stopwords.

Binary classifier

Initial setup

Now that we have our dataset of texts annotated with the binary decision and hate speech class, the first step is to build the binary classifier. Here, we use the preprocessed datasets, that contain no stopwords, which can be found as “<nr>_no_stopwords.csv” in our datasets folder.
As already explained, we used an official tutorial, that shows how to build a model for sentiment analysis on movie reviews, as a basis. We then created the first iteration of our classifier by adapting and extending the code to fit our data and task. In the following, the pipeline that the data runs through is described.
First, we need to load all four of the preprocessed datasets. Now we can split the resulting dataset into training and testing data. We shuffle the data to get a good distribution of the different data and split it into 80 percent for training and 20 percent for testing. The “is_hatespeech” label of our dataset is converted to a one hot encoding. Finally, the dataset is converted to an instance of a Tensorflow dataset object. This allowed easy batching and shuffling.
Afterwards, we let Tensorflow create a vocabulary of length 1000 from the text of the datasets, which contains the most frequent words and is used for the first layer of our model, the text vectorization. The number 1000 seemed to be a good starting point, regarding computing time and quality of results. The text vectorization is the first layer of our model and converts any input string to a numerical representation.
The second layer takes the vectorized string and generates an embedding with length 64 masking out the zero. With this more generalized form we can go on in our pipeline. The output of the embedding layer is computed in an Long Short Term Memory (LSTM) RNN layer with the same output size.
Finally, we dense the output of the previous layers to get our binary decision, whether the computed text is considered hate speech or not. We do this in two steps. In the first step, we use a simple linear activation function (in detail: RELU), in the second step we use Softmax to get a more stochastic distribution.
Since the initial model is now finished, we could do some testing. We used the previously generated testing data to check our models performance. The results show that we already got a decent accuracy (~91,6%) with the initial model. That shows that the approach is very

suitable for the task. Nonetheless we try to further improve the models performance by trying different other approaches and parameters, which are described in the following section. All validation results can be found in table 2. Values for precision, recall and F1-score are always the macro values, meaning they are not individually weighted and both classes (hate speech and no hate speech) are taken into account with the same weight. That is because the amount of data for no hate speech is bigger and we do not want to let this bias the results. The model name is correlating with the filename of the corresponding notebook in the models folder in our repository. In the notebooks you can also find detailed reports about the models performances.

Model name	Description	Loss	Accuracy	Precision	Recall	F1-Score
standard_model	First computable binary classifier	0.210	0.916	0.92	0.88	0.90
	With double RNN layer	0.214	0.914	-	-	-
stopwords_model	Standard model using datasets containing stopwords	0.197	0.921	0.92	0.88	0.90
stopwords_model_big_vocab	Again using stopwords in datasets, doubled vocab size	0.190	0.927	0.93	0.90	0.91
keywords_model	Using predefined vocabulary for text vectorization	0.226	0.927	0.92	0.89	0.91
multi_feature_model	Use all improvement strategies	0.210	0.927	0.94	0.88	0.91

Table 2: Test results

Performance improvements

The first thing we tried to improve the performance, as also presented in the Tensorflow tutorial, is to add another RNN layer. As you can see in table 2, this gave no improvements at all.
Then, the first question we had is if removing the stopwords from the datasets is really helpful. Of course, they do not contain any important information that might be carried in hate speech. But maybe the model works better when we keep them. So we use our standard model and use the “<nr>_no_stopwords” datasets as input. On this data, the model indeed performs slightly better, the accuracy was improved by about half a percent.
The next question we had is what happens if we increase the size of the vocabulary used in our text vectorization. So we doubled the value to 2000 and recomputed our model. That gave an additional half a percent improvement in accuracy. Also precision, recall and F1-score slightly increased. We are already at a very high level of performance, so little improvements are quite difficult to achieve.

Keywords approach

In the first model, we used a vocabulary for text vectorization that is adapted from the words that are frequently used in our dataset. These words can literally be anything. So a different

approach would be to just use words that are really related to hate speech. It would be quite difficult to get those words out of the dataset, because there are no linguistic rules for bad words in the English language. To get a good vocabulary that is really containing bad words, it is necessary to use a predefined dataset. We found three datasets for this task on the web. Together, they contain more than 3000 words. The words are also present in different forms, as especially in the web they are used in short, special or abbreviated forms (e.g., f**ker - motherf**ker - butt-f**ker - f**ktard - etc.). It is again necessary to do some preprocessing on the datasets to bring them in the same format, which is just a list of the words, and remove unwanted artifacts. After the preprocessing, the datasets were combined into one single dataset (bad_words.csv), with all duplicates removed. 2866 words remain and can be used as the vocabulary for the text vectorization.
For this approach we use the datasets without stopwords, since the vocabulary should not contain any of them. Interestingly, the performance of this model is pretty similar to the previous model, only a little bit worse. More thoughts about this will follow in the evaluation part.
Now that we tried some different settings to get the best possible classifier, we combined all the learnings in one big model. That means, we are using both of the vectorization methods and concatenate the computed outputs at the end. For the adapted vocabulary vectorization we also use a big vocabulary. This model again scores some better results, even if the improvements are now very small.
Because this model combines all the previous iterations of binary classifiers, we will present the complete pipeline from raw datasets to the trained model for it in image 1. It is just a visual summary of what was described until now.

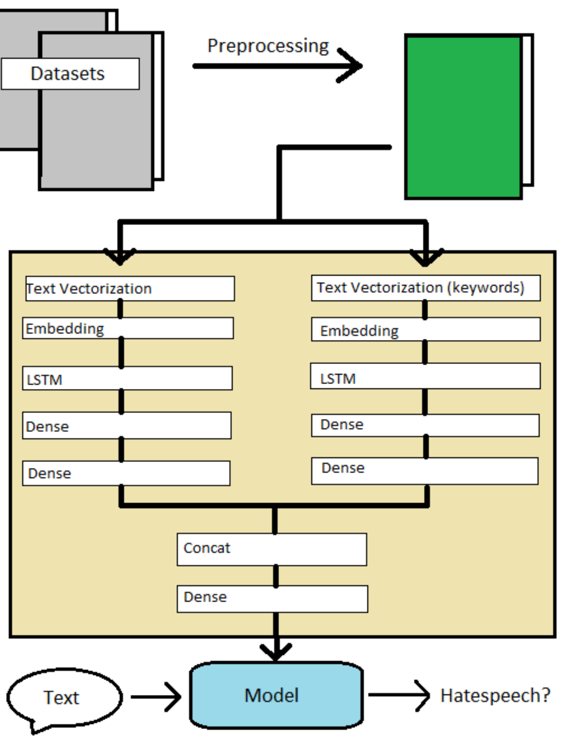


Image 1: Binary classification pipeline

Data augmentation

Because the performance of our binary classifier stagnated and was not increasing significantly at the beginning, we also thought about the possibility of augmenting the data using another dataset. For this we chose the “goemotions”-dataset which contains about 58k curated reddit comments that were labeled with the emotions that they convey. This label was not further taken into account. The text of each sample was isolated and labeled as no hate speech in order to extend our dataset with a large amount of samples that are not hate speech. Adding the aforementioned data to both the training and the testing data gave the accuracy a significant boost of roughly 4 percentage points from ~91% to 94.5%.
However when adding the additional data to the training set but omitting it from the testing, in order to check if the classification of the “is_hatespeech=true” samples became better, the resulting accuracy dropped down to 42% and the loss increased extremely.
Even though at first it seemed very promising, we dropped the idea as the resulting classifier performed worse than before.

Binary classifier evaluation

As stated in the above chapters, we started with a quite performant model for hate speech identification. It was only possible to improve the classifier by about one percent. We must admit that we sometimes had slightly different outcomes when training our models. For example the last one, for which we finally had an accuracy of 92.7%, previously got a whole percent better performance. We might have messed things up regarding the environment or the seeding of random dataset shuffling. Of course, the outcomes are also highly dependent on our model architecture and parameters. There will probably be better architectures and settings available, but the presented ones were the best we got. We tried a lot of different setups, especially for the parameters, but since there are nearly endless possibilities, it is hard to find the best one. We think that our results are still very acceptable.
Let us compare this model with the even a little better performing model using text with stop words and a bigger vocabulary for text vectorization. In image 2 you can see the confusion matrix of the test results for the multi feature model. In image 3 you see the results for the other model. The first model could identify text that is no hate speech slightly better, the second one did a better job on hate speech. The classifier using only keywords as vocabulary was also better in finding hate speech, as you can see in image 4. But as already implied, a list of keywords might never be complete, you will always find another word that is not in the dataset yet. In addition to that, some hate speech might have its message put out in the way it is formulated and not in specific words.
It is hard to say which configuration of the presented ones is the best, but we think the model using two features (adapted text vectorization and keyword text vectorization) is the most promising. On the one hand, hate speech containing any of the keywords can easily be identified, because the keywords will most probably not be used in normal text. On the other hand, if hate speech does not contain any of the predefined words, we still have the second encoding that checks it for the most common structures.

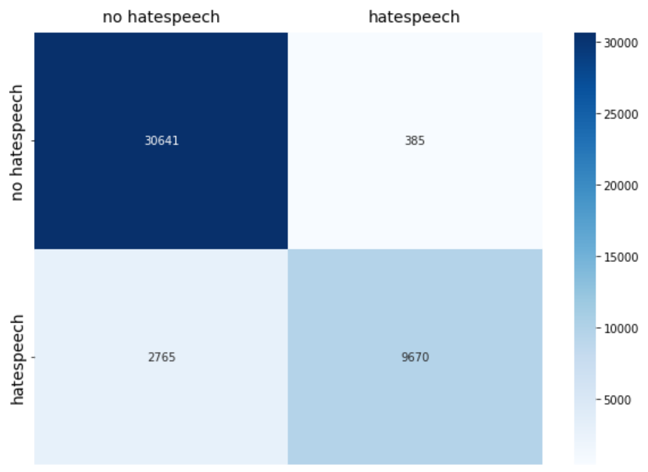


Image 2: Confusion matrix for multi feature classifier

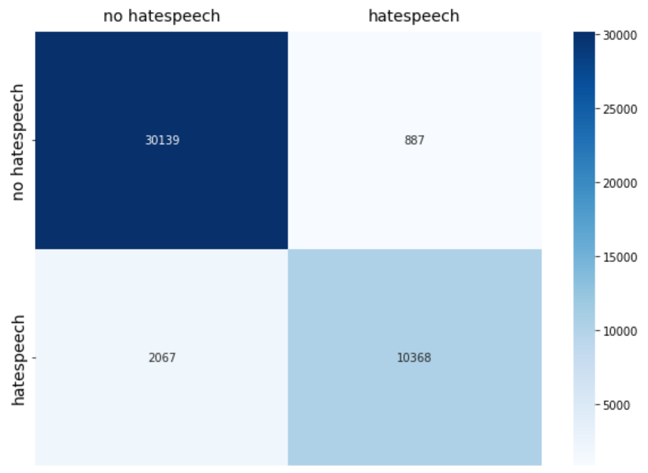


Image 3: Confusion matrix for big vocab classifier using stopwords

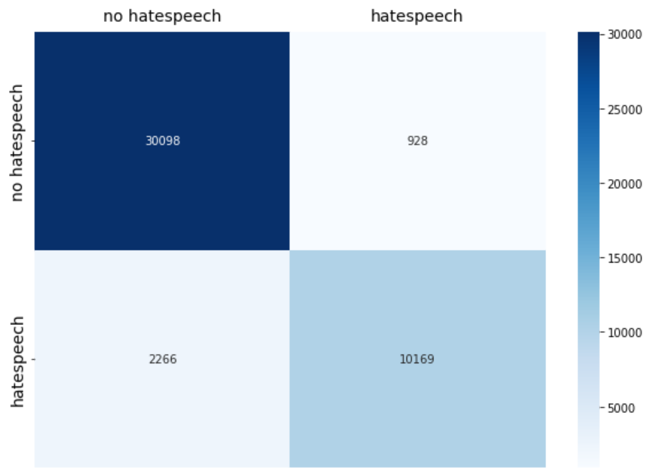
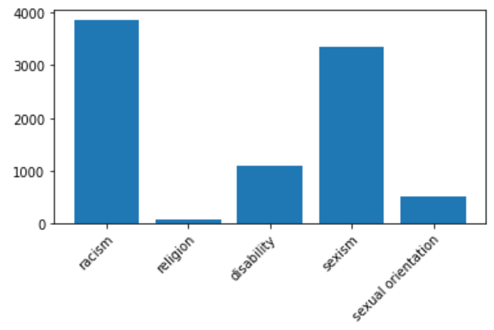


Image 4: Confusion matrix for the keywords classifier

Multi-class classifier

Initial setup

For the multi-class classifier a very similar model structure was used, but the hyperparameters were adjusted. For the output layer a softmax activation was used to get our classification to be in a similar form to a probability distribution. Additionally, as this model further classifies hate speech, the datasets were filtered to remove all entries that were not labeled as hate speech. The distribution of the labels that were can be seen in the following plot.



After training the models, the next step was to test them using the remaining 20% of the data.

Performance improvements

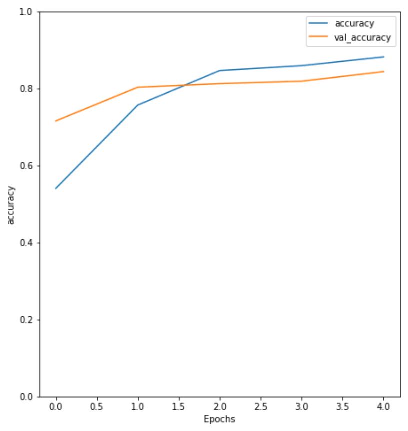
The initial performance of the Multi-class classifier wasn't very good and came in to about 80% accuracy with the same initial model architecture as the binary classifier but switched up to allow for a one hot encoding of the classes.
The general model was largely left as is, but the hyper parameters were tuned and through repetitive testing moved towards increasing the quality of the results.

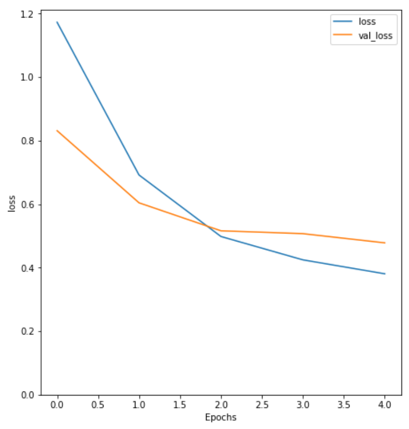
LR	**#Epochs**	**Testing Accuracy**	**Loss**
0.0025	7	0.845	0.618
0.005	7	0.841	0.649
0.00075	7	0.833	0.571
0.00075	5	0.858	0.456

In the table above you can see the last five experiments with the hyper parameters before finding the final and current best configuration for this particular setup.

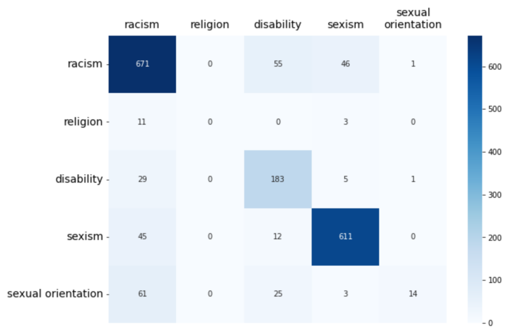
Multi-class classifier evaluation

The achieved results vary greatly with respect to the total amount of data that was available for each of the classes. The best performing classes, 'racism' and 'sexism' are also the ones with the highest amount of data available. The remaining classes, perform O.K at best, and horrible in the worst case, as 'religion' had very little data available and the 'sexual orientation' class also performs rather badly. More training data for these classes would help to improve the results.



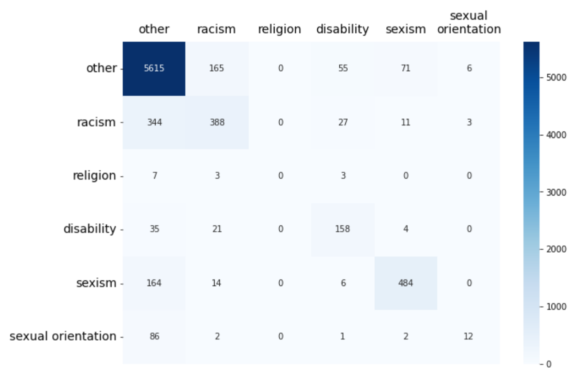


```
                     precision    recall  f1-score   support

             racism       0.82      0.87      0.84       773
           religion       0.00      0.00      0.00        14
         disability       0.67      0.84      0.74       218
             sexism       0.91      0.91      0.91       668
 sexual orientation       0.88      0.14      0.24       103

           accuracy                           0.83      1776
          macro avg       0.66      0.55      0.55      1776
       weighted avg       0.83      0.83      0.82      1776
```



Experimenting with real-world data

While the previous results seem O.K in theory, after experimenting with the Reddit-API and classifying comments, quite the oversight was obvious. Because at first we omitted the ‘other’-class from the classifier as it isn’t as accurately labeled, a lot of comments that were maybe somewhat offensive, but definetly not part of the main-three classes, would be labeled as ‘religion’ or ‘sexual orientation’.
To work around this, all the samples labeled ‘other’ were added to training and testing again and these were the results:



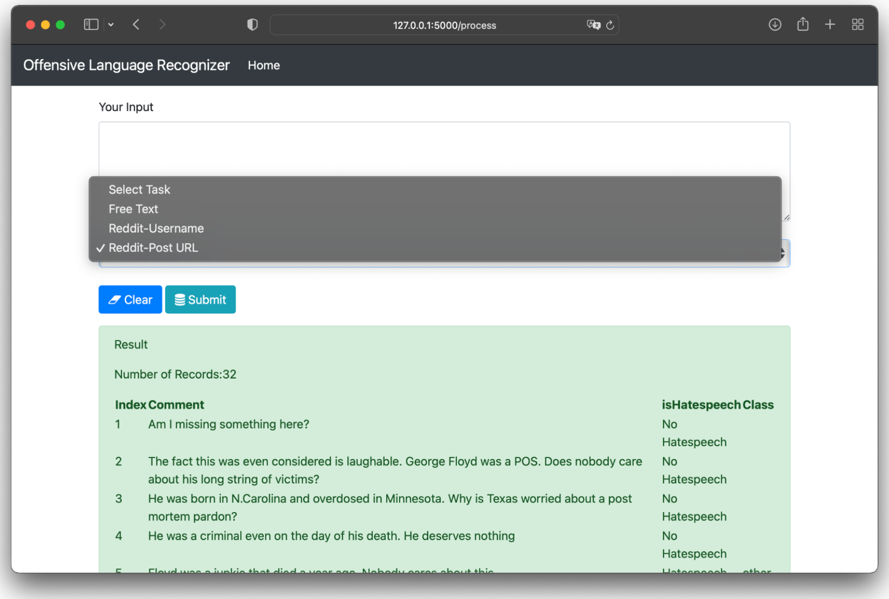
The large amount of data that is labeled as ‘other’ skews the overall results heavily, and it can be seen that ‘other’ probably also contains samples that should be accurately labeled as ‘racism’ as there are a lot of misclassifications of the two.

```
                    precision    recall  f1-score   support

             other       0.90      0.95      0.92      5912
            racism       0.65      0.50      0.57       773
          religion       0.00      0.00      0.00        13
        disability       0.63      0.72      0.68       218
            sexism       0.85      0.72      0.78       668
sexual orientation       0.57      0.12      0.19       103

          accuracy                           0.87      7687
         macro avg       0.60      0.50      0.52      7687
      weighted avg       0.86      0.87      0.86      7687
```

The exact results show that ‘other’ performs great, but that the ‘racism’ class’s results got way worse from 0.84 F1-Score to 0.57, the same goes for ‘sexism’ but with a less severe decrease. Again, we would lead this change back to the labeling of the data that is currently labeled as ‘other’.

UI and Reddit

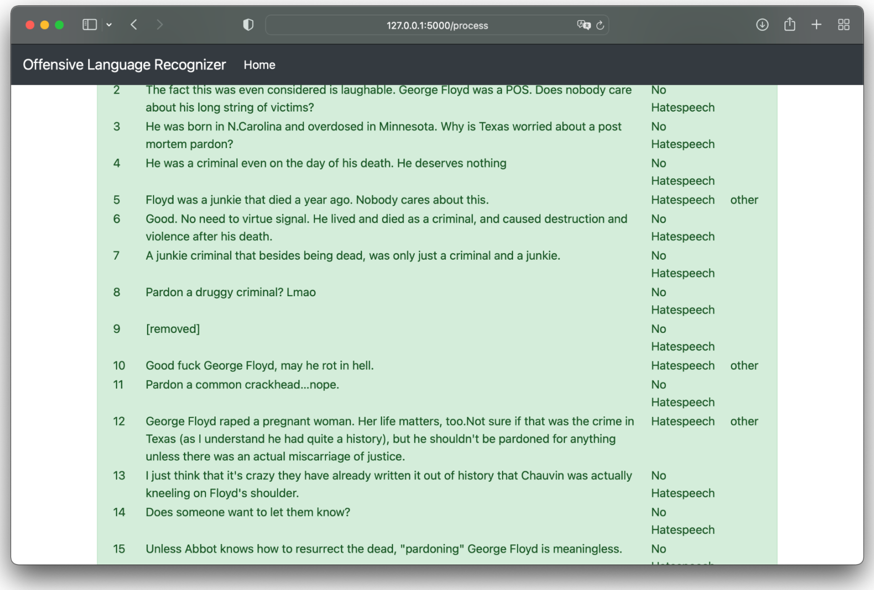
In order to be able to try the models on some real-world data, we implemented functionality to utilize the Reddit-API to retrieve all the comments a specified user has posted, or all comments that were commented below a post. The user can be entered by their username and the post by providing its URL.
To make the specification of our parameters easily accessible, we use a Flask-Server, that handles the input of either an URL or username and displays the results that were generated using our classifier-models.
The models are used in two phases. Phase one is to check whether or not the entered comment is hate speech or not, and phase two is triggered if phase one returns “is_hatespeech==true”.
Additionally it is possible to just enter a sentence manually and have it classified.



Result

Number of Records:32

Index	Comment	isHatespeech	Class
1	Am I missing something here?	No Hatespeech	
2	The fact this was even considered is laughable. George Floyd was a POS. Does nobody care about his long string of victims?	No Hatespeech	
3	He was born in N.Carolina and overdosed in Minnesota. Why is Texas worried about a post mortem pardon?	No Hatespeech	
4	He was a criminal even on the day of his death. He deserves nothing	No Hatespeech	
5	Floyd was a junkie that died a year ago. Nobody cares about this	Hatespeech	other



Index	Comment	isHatespeech	Class
2	The fact this was even considered is laughable. George Floyd was a POS. Does nobody care about his long string of victims?	No Hatespeech	
3	He was born in N.Carolina and overdosed in Minnesota. Why is Texas worried about a post mortem pardon?	No Hatespeech	
4	He was a criminal even on the day of his death. He deserves nothing	No Hatespeech	
5	Floyd was a junkie that died a year ago. Nobody cares about this.	Hatespeech	other
6	Good. No need to virtue signal. He lived and died as a criminal, and caused destruction and violence after his death.	No Hatespeech	
7	A junkie criminal that besides being dead, was only just a criminal and a junkie.	No Hatespeech	
8	Pardon a druggy criminal? Lmao	No Hatespeech	
9	[removed]	No Hatespeech	
10	Good fuck George Floyd, may he rot in hell.	Hatespeech	other
11	Pardon a common crackhead...nope.	No Hatespeech	
12	George Floyd raped a pregnant woman. Her life matters, too.Not sure if that was the crime in Texas (as I understand he had quite a history), but he shouldn't be pardoned for anything unless there was an actual miscarriage of justice.	Hatespeech	other
13	I just think that it's crazy they have already written it out of history that Chauvin was actually kneeling on Floyd's shoulder.	No Hatespeech	
14	Does someone want to let them know?	No Hatespeech	
15	Unless Abbot knows how to resurrect the dead, "pardoning" George Floyd is meaningless.	No	

Who did what

We did not strictly split our work into parts so that everybody has his own topics. We both worked on building and improving the models, while Oliver had a bigger focus on the multi-class classifier and Finn on the binary classifier. Finn also focused on dataset research and preprocessing, while Oliver additionally worked on setting up the Flask server and integrated the Reddit-Api for interactive classification into it.

Literature and sources

Project repository

https://github.com/Caramba997/hatespeech_classifier

Used datasets

1. 1_labeled_data.csv: https://github.com/t-davidson/hate-speech-and-offensive-language
2. 3_en_dataset_with_stop_words.csv: https://github.com/HKUST-KnowComp/MLMA_hate_speech
3. 4_harassment-corpus.csv: https://github.com/Mrezvan94/Harassment-Corpus
4. 5_train.csv: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data?select=train.csv.zip
5. 6_NAACL_SRW_2016.csv: https://github.com/zeeraktalat/hatespeech
6. 7_bad-words.csv: https://www.kaggle.com/nicapotato/bad-bad-words
7. 8_Terms-to-Block.csv: https://www.kaggle.com/sahib12/badwords
8. goemotions: https://github.com/google-research/google-research/tree/master/goemotions

Sources of information

1. Tensorflow tutorial: https://www.tensorflow.org/text/tutorials/text_classification_rnn