Heidelberg University
Institute of Computer Science
Database Systems Research Group

Report for the lecture Text Analytics

Hate Speech Detection

`https://github.com/fidsusj/HateSpeechDetection`

Mentor: John Ziegler

Team Member: Christopher Klammt, 3588474,
Applied Computer Science
iv249@stud.uni-heidelberg.de

Team Member: Felix Hausberger, 3661293,
Applied Computer Science
eb260@stud.uni-heidelberg.de

Team Member: Nils Krehl, 3664130,
Applied Computer Science
pu268@stud.uni-heidelberg.de

Abstract

Plagiarism statement

Contents

1 Introduction

One current research area in the field of text analytics is hate speech detection. Many approaches from the recent past make use of neural network architectures to deal with such classification problems. One of the most common approaches is to use uni- and bidirectional long short-term memory (LSTM) networks, a recurrent neural network architecture that can process input of arbitrary length and remembers context information [6, 18, 17]. The paper [8] states, that even a simple gated recurrent unit (GRU) architecture can perform as good as more complex units. [1] repurposes the famous bidirectional encoder representations from transformers (BERT) language model to perform classification tasks for hate speech detection. Besides that, other approaches use convolutional neural networks (CNNs) to extract typical hate speech patterns [2, 16, 10] or even deep belief network algorithms [13]. Using neural network approaches means to automatically learn representative features for the classification task. On the other hand, the papers introduced in section 2 use a different approach by solving the classification task with manually extracted features. Nevertheless, none of the papers combines the different achievements of such recent research and compares it to a baseline neural network architecture, which is what this work is dedicated to.

After a definition of the term "hate speech" in section 3 different classifiers will be trained on a holistic, hand-crafted feature set based on recent publications in the field of hate speech detection. The task includes building and preprocessing a training corpus as well as introducing and explaining the different kinds of features. How well the different classifiers perform compared to a neural network approach as a baseline and several statistical insights into typical hate speech artifacts will be presented in section 4. The results of this work in section 5 should show which features work best for which classifier and which problems can be addressed with conventional machine learning methods and which not as opposed to neural network approaches. A summary over the achievements earned will be drawn in section 6.

2 Related Work

As the goal of this work is to solve a hate speech classification problem with manually extracted features, recent work was evaluated proposing different feature sets for this task.

[19] categorizes features into four different groups. Sentiment-based features give information about the polarity of a document, which is important as many hate speech documents stand out by being mostly negative. Sentiment features count the occurrences of punctuations, capitalized words, interjections, etc. A dictionary of typical hate speech words can be obtained by extracting most common unigrams from a given corpus building the unigram features. Pattern features represent the last feature group containing common syntactic patterns based on PoS tags. The approach presented in this paper achieved an accuracy of 87.4% using combined features from these four groups. The classifiers used were SVM, Random Forest and J48graft.

[14] concludes character n-grams, word n-grams, negative sentiment-based scores and syntactic-based features as a decent feature set to train classifiers on. Looking at the results, an optimized support vector machine with character n-grams performed best with 0.894 TPR, while optimized gradient boosting performed best with word n-grams, giving a 0.867 TPR.

[7] divides features into generic text mining features and specific hate speech detection features. Typical generic text mining features are dictionaries of insults typical for hate speech, swear words, profane words, verbal abuse etc., n-grams, lexical syntactic based template features, that capture grammatical dependencies within a sentence, topic classifications with latent dirichlet allocation, or sentiment polarity scores. On the other hand specific hate speech detection features do not rely on common abstract concepts known in the field of text analytics, but come with purpose built frameworks to detect these features. Using the Stanford lexical parser along with a context-free lexical parsing model one can identify language which is used a lot in hate speech. Other examples of specific hate speech features are the objectivity-subjectivity relations of the language as hate speech is more related to subjective communication, focus on particular stereotypes, intersectionism of oppression or declarations of superiority of the in-group.

Other related work like [9], [11] and [4] once more stress the importance of word and character n-grams for hate speech detection tasks. [4] even uses count indicators for hashtags, mentions, retweets and URLs and is especially important as a part of the dataset used in this work originated from the project behind this paper. The best performing model achieved 91% overall precision, 90% recall and a 90% F1-score, but the model is biased towards classifying tweets as less hateful or offensive than the human supervisors.

3 Approach

The epistemic research interest in this work is to clarify the question, whether conventional machine learning methods combined with suitable features can outperform neural network based approaches.

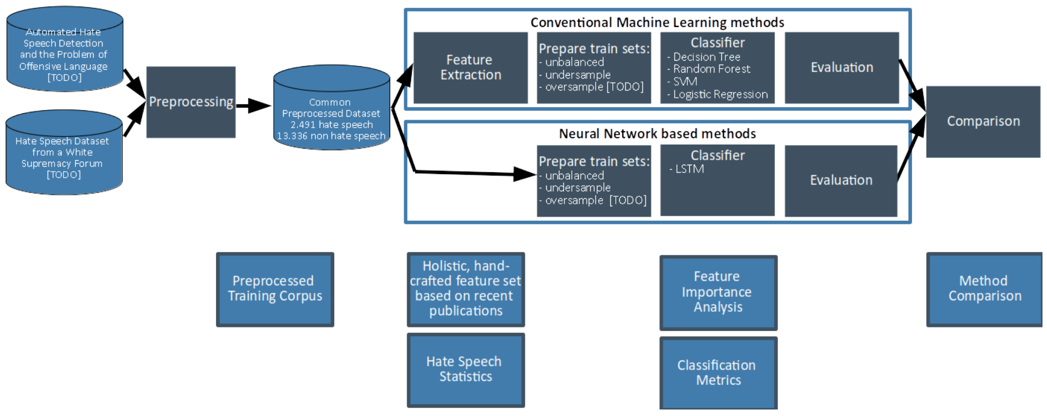

Automated Hate Speech Detection and the Problem of Offensive Language [TODO]
Hate Speech Dataset from a White Supremacy Forum [TODO]
Preprocessing
Common Preprocessed Dataset 2.491 hate speech 13.336 non hate speech
Conventional Machine Learning methods
Feature Extraction
Prepare train sets: - unbalanced - undersample - oversample [TODO]
Classifier - Decision Tree - Random Forest - SVM - Logistic Regression
Evaluation
Neural Network based methods
Prepare train sets: - unbalanced - undersample - oversample [TODO]
Classifier - LSTM
Evaluation
Comparison
Preprocessed Training Corpus
Holistic, hand-crafted feature set based on recent publications
Hate Speech Statistics
Feature Importance Analysis
Classification Metrics
Method Comparison

Figure 1: Approach

Figure 1 visualizes our approach (on the top) and the resulting novelties achieved through this work (at the bottom). The following chapters describe the different steps in detail. First the data is merged and preprocessed resulting in a preprocessed training corpus. The next step differs between conventional machine learning methods and neural network based approaches. For conventional machine learning methods an explicit feature extraction is necessary. Therefore achievements of recent publications were combined to build a holistic, hand-crafted feature set. Based on these features, hate speech statistics could be further analyzed. Due to the fact that the common preprocessed corpus is unbalanced, an unbalanced, oversampled and undersampled dataset was created for further investigation. Finally the classifiers are trained, evaluated and the results are compared. Now not only the question whether conventional machine learning methods can outperform neural network based approaches could be answered, but as well which feature were most important for which classifier.

3.1 Definition of hate speech

Various definitions exist to define hate speech. This work complies with the definitions provided by the two datasets used to label the documents.

> **Ona De Gibert et al.:** Hate speech is commonly defined as any communication that disparages a target group of people based on some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic [5].

> **Thomas Davidson et al.:** A language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group [4].

One can conclude that hate speech is always targeted towards a specific group with the intention to infringe others dignity, often based on group characteristics like race, color or gender.

3.2 Data

There are two data sets used for the project. The first one uses data from the *Twitter API* [4][1]. It consists of a sample of around 25k tweets that were identified as hate speech based on a previously composed hate speech lexicon without regarding context information. Subsequently, each document in the corpus got labeled with one of the three categories *hate speech*, *offensive language* or *neutral*. The workers were instructed to follow predefined definitions of each category and to take context information into consideration. Each tweet was assessed and labeled by three or more workers. The majority of tweets were classified as offensive language (76% at 2/3, 53% at 3/3), only 5% were coded as hate speech. In the end 1.430 hate speech documents, 4.175 neutral documents and 19.196 offensive language documents make up the entire dataset. The data is provided offline as a CSV or pickle file.

The second data set uses data from *Stormfront*, a white supremacy forum [5][2]. One document represents a sentence that is either labeled as hate or not hate. In total, 1.196 sentences containing hate and 9.507 sentences being non-hate are provided. Once again the documents were labeled manually by human actors following previously specified guidelines, on request additional context information was provided. The documents are given offline as normal text files with annotations stored in a separate CSV file.

3.3 Features

The choice of features to use can be divided into five groups inspired by [19].

[1]https://github.com/t-davidson/hate-speech-and-offensive-language
[2]https://github.com/Vicomtech/hate-speech-dataset

One group of features is the unigrams features group inspired by [4, 14, 7, 9, 11]. During the training phase we extract the top 100 words for hate speech and neutral speech based on the TFIDF scores. These words are given as dictionaries during the feature extraction phase. We therefore count the occurrences of typical hate speech and neutral speech words in each document as a feature. A similar approach extracts typical hate speech n-grams based on word stems using with NLTK during the training phase and builds n-gram dictionaries. In case n-grams surpass a certain threshold they are added to the dictionary. This threshold is 10 for unigrams, 8 for bigrams and 2 for trigrams. The number of typical hate speech unigrams, bigrams and trigrams detected is added as a feature.

Another feature group are the semantic features taken from [4, 19]. They comprise the number of exclamation marks, question marks, full stop marks, interjections[3], all capital words, quotation marks as an approximation for the number of quotes, laughing expressions[4] and the number of words in general.

To expand the set of features to cover syntactical characteristics of hate speech documents, PoS tag patterns were extracted belonging to the group of pattern features [14, 7]. A sliding window approach with a custom definable window size range is used to extract PoS tag patterns for each document during the training phase. In case a hate speech pattern occurs more than 500 times in the training set it is added to a hate speech pattern dictionary. The number of hate speech patterns detected in each document in taken as a feature.

Sentiment-based features inspired by [14, 7] were implemented by extracting the polarity scores for each document using NLTKs SentimentIntensity-Analyzer from VADER (Valence Aware Dictionary and sEntiment Reasoner). VADER is a lexicon and rule-based sentiment analysis tool specifically developed for social media sentiment analysis.

The last feature added to the holistic, hand-crafted feature set is the topic detected from gensims latent dirichlet allocation algorithm [7]. For each document a numerical topic of either zero or one is added to the set of features based on which topic received the highest probability score.

The feature extraction of the previously mentioned features is done within a reusable pipeline, which makes it easy to add new features. Each feature is implemented as a class following a predefined interface. By adding the feature class to a list in the *FeatureExtractor* the feature is automatically extracted as part of the pipeline. The pipelines input are raw text documents, stemmed documents, lemmatized documents and PoS tags each in its own dataframe

[3]recognized by NLTKs PoS tags

[4]based on the regular expression r"(a*ha+h[ha]*|o?l+o+l+[ol]*)|(lmao)"

column and the output is a dataframe containing all extracted features as numerical values.

3.4 Classifiers

In this work five classifiers are compared on the different datasets (unbalanced, undersampled, oversampled). Four of them are conventional machine learning methods (Decision Tree, Random Forest, SVM, Logistic Regression) and one neural network based approach (LSTM).

Each classifier is trained on the training set and evaluated on the test set. So the same steps are necessary for each classifier. That is why a reusable pipeline is developed. The pipeline is developed with the open-closed principle in mind. It is open for extensions and closed for changes. So when adding a new classifier only a few lines of code need to be adapted and steps such as finding the optimal model through hyperparameter tuning and the evaluation are done automatically as part of the pipeline.

As the methods need it, there are slight differences between the training of the conventional machine learning methods and the neural network based approaches. Figure 2 illustrates this.

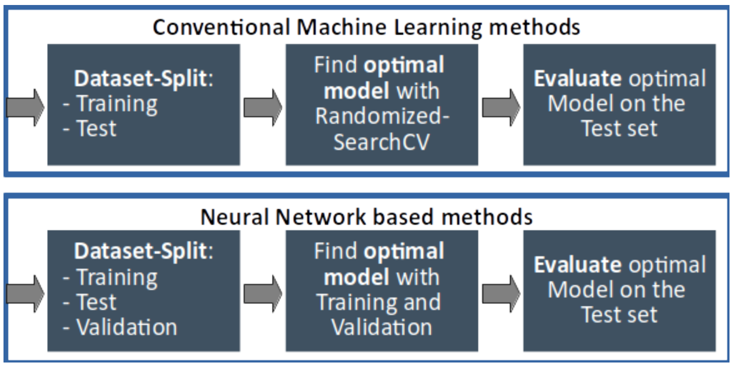

Conventional Machine Learning methods
Dataset-Split: - Training - Test
Find optimal model with Randomized-SearchCV
Evaluate optimal Model on the Test set
Neural Network based methods
Dataset-Split: - Training - Test - Validation
Find optimal model with Training and Validation
Evaluate optimal Model on the Test set

Figure 2: Classifier pipeline

For the conventional machine learning methods the dataset is split into training (0.8) and test set (0.2). In order to find the optimal model *Randomized-SearchCV* is executed on the defined hyperparameter search space. The hyperparameter search space was chosen based on the classifiers documentation, papers and by an empirical examination. For the decision tree [12] and for the random forest [15] were observed. For SVM we used the sklearn C-Support Vector Classification implementation and concentrated the grid search on the kernel (linear, poly, rbf or sigmoid) as this has the biggest impact on performance. Tuning the kernel already used up quite some computational time, so we did not look into further parameters. In future work one could have a closer look at parameters such as the regularization parameter C or some kernel specific coefficients. To train the logistic regression model, hyperparameters were set according to the recommendations from sklearns documentation page. To try out different solver algorithms, the L2-norm penalty for regularization was chosen. As the number of samples surpass the number of features the normal primal formulation of the regression problem was used. The search space for solver algorithms was restricted to *lbfgs*, *sag* and *saga* as others did not make sense in the training circumstances. Also different regularization strength were tested with parameter C between 0.8 and 1.2 with step width 0.1. Finally the model is evaluated on the test set.

For neural network based approaches the dataset is again split into training (0.8) and test set (0.2). Then the training set is further split up into training (0.8) and validation (0.2). In the next step the optimal model is found by using the training and the validation set. The final step is equal to the final step in conventional machine learning, were the model is evaluated.

To enable a performant execution Pythons multiprocessing library is used to parallelize the execution.

3.5 Evaluation

For evaluating the classifiers standard metrics such as accuracy, precision, recall and F1 score are used.

The accuracy specifies how many data instances are correctly classified. Only looking at the accuracy, is not that informative, because generally a lot of instances are correctly classified as non hate speech, which leads to a high accuracy. That is why also precision and recall are observed. Precision specifies how many of the predicted hate speech instances are really hate speech. A low precision means that many instances are classified as hate speech, although they are not. So looking at the precision enables us to detect if the trained model could be used as a censoring system and undermine the freedom of speech. The recall specifies how many hate speech instances are correctly classified by the model. A low recall means that there are many false negatives (a lot of hate speech instances are not detected). For taking into account precision and recall one can look at the F1 score.

4 Experimental setup

4.1 Data preparation

To prepare a central dataset, both single datasets had to be transformed into a common format. For the central dataset only the class and the text content of each tweet respectively each forum contribution was considered.

The first dataset “Automated Hate Speech Detection and the Problem of Offensive Language” was entirely given as a .csv file and contains around 25k tweets, that were either labeled as hate speech, offensive language or neither of both. To determine the right label three independent evaluators classified each tweet, the final label got assigned by the majority vote. As for the first approach, one is only interested in hate speech and neutral tweet classification, all offensive language documents in the dataset were dropped. Some tweets were retweets that were commented additionally by a user. An example is shown below:

> """@DevilGrimz: @VigxRArts you’re fucking gay, blacklisted hoe"" Holding out for #TehGodClan anyway http://t.co/xUCcwoetmn"

The original tweets can be found in between the ""..."". As it could not be distinguished whether the original tweet or the retweet contains hate speech, these documents were filtered out in the first version of the preprocessing pipeline. Same goes for tweets that cite other users without using the retweet option. Once the semantic feature for the number of quotes was added, the removal of quoting tweets was reverted again. Probably it is still expected behavior in social media platforms to remove contributions that build up on hate speech.

The second dataset “Hate Speech Dataset from a White Supremacy Forum” was not entirely given as a .csv file. Only the document annotations were given in a .csv file, all forum contributions were stored in separate .txt files. Only documents which could not be assigned to a single class (label "idk/skip") or referred to other documents (label "relation") were dropped.

The resulting common dataset was stored in a .csv file. It contains 2.626 hate speech documents and 13.670 non hate speech documents. The dropped offensive language documents make up 19.196 instances.

4.2 Corpus building

The common dataset is loaded from the .csv file into a pandas dataframe. After doing basic preprocessing like lowercasing and removing emojis and other irrelevant characters, spacy is used to build a tokenized corpus. The language model from spacy decides about stop word, punctuation and white space removal. No hard coded logic or stop word lists are used in this process. This keeps URLs or other tokens including punctuation as one token.

Regarding tokenization using the lemmas instead of word stems works better as for instance *we'll* becomes *["we","will"]* and not *["we", "'ll"]*. The lemmatized tokens were later on stemmed as well in order to keep the range of possible unigrams for one and the same semantic word instance restricted for the task of n-gram feature extraction. PoS tags are added to the dataframe using NLTKs PoS tagger as problems using spacys PoS tagger were encountered. PoS-tags are necessary for the hate speech pattern extraction.

Depending on the method the inputs vary. For the conventional machine learning methods the inputs are the extracted features as numerical values added to the dateframe by the already mentioned *FeatureExtractor* class, whereas in neural network based approaches the inputs are the raw textual contents of each document and the embeddings are learned automatically. The labels are in both cases numerical values.

Then we perform the dataset balancing. As already mentioned, the acquired dataset is an imbalanced one, which can lead to a decrease in performance and accuracy with machine learning classification. As a comparison the paper [14] also recognizes the class imbalance and tries to reduce it by applying a synthetic minority oversampling technique called SMOTE [3]. In general there are a few possibilities to tackle the challenge of unbalanced classes:

- changing the performance metric (e.g. F1-score instead of accuracy)
- undersampling, i.e. deleting instances from the over-represented class
- oversampling, i.e. adding copies of instances from the under-represented class
- generating synthetic samples (e.g by using SMOTE)

In our experiments we chose to try the different approaches and applied a simple undersampling by randomly deleting instances from the over-represented class (neutral posts), as well as an oversampling using SMOTE. SMOTE does an oversampling by generating synthetic samples in the feature-space of the under-represented class, resulting in posts that statistically fall in between the different hate speech posts. Additionally, we used the imbalanced dataset to train a classifier to compare how this affects the performance. Finally we receive unbalanced, undersampled and oversampled datasets, which we use for the further experiments. In addition for all experiments, we use the F1-score instead of the accuracy as our performance metric, which should somewhat handle unbalanced classes by itself.

Unfortunately, SMOTE only works in feature space, so we could not provide an oversampled balanced dataset for our neural network approach as this works directly on the hate speech and non-hate speech posts. This could be addressed in future work.

4.3 Data insights

To better understand the dataset we are working with, the following section provides some insights into the preprocessed data corpus. In an in-depth analysis of the data, we had a look at the length of hate speech posts versus non-hate speech posts. This can be seen in Figure 3.

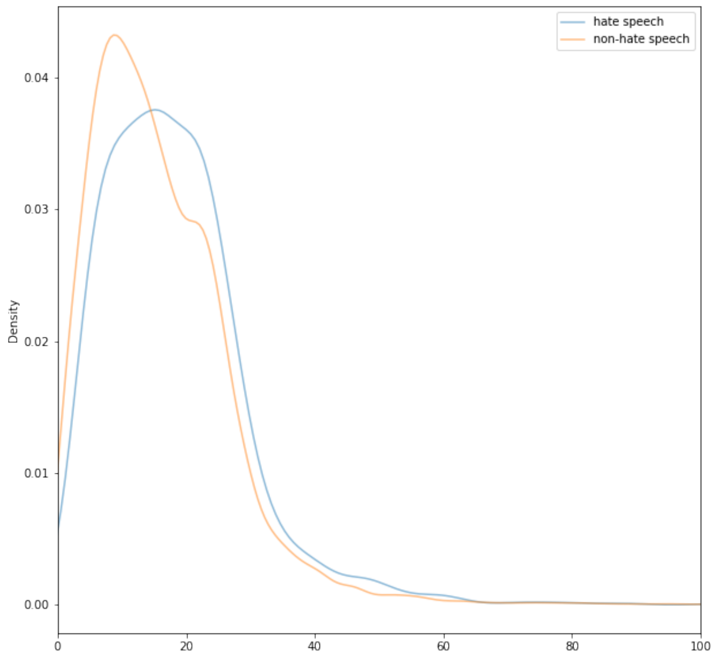
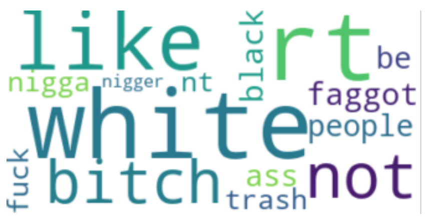


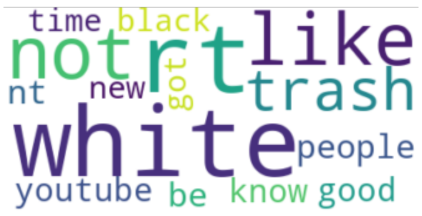
Figure 3: Density distribution of the length of a post (tweet or forum post)

Here one can see, that the hate speech posts contain more words (tokens before cleaning) than non-hate speech posts. In average a hate speech post contains 18.18 words, whereas a non-hate speech post only contains 15.85 words. Unlike expected, the hate speech posts are longer than the non-hate speech posts.

A more interesting look at the data is given by the most commonly used words per class. As can be seen in the word clouds in Figure 4 there are some obvious differences, such that the hate speech posts use words like “bitch”, “faggot” or “nigga”. But interestingly enough, the non-hate speech posts also sometimes consist of the words “trash” or “white”.



(a) hate speech



(b) non-hate speech

Figure 4: Word clouds

For a better insight with what data we are dealing, a few examples are shown in the following.
Examples for non-hate speech (neutral sentences):

- "billy that guy wouldn’t leave me alone so i gave him the trudeau salute"
- "this is after a famous incident of former prime minister pierre trudeau who gave the finger to a group of protesters who were yelling antifrench sayings at him"
- "askdems aren’t you embarrassed that charlie rangel remains in your caucus"

Examples for hate speech:

- "california is full of white trash"
- "and yes they will steal anything from whites because they think whites owe them something so it’s ok to steal"
- "why white people used to say that sex was a sin used to be a mystery to me until i saw the children of browns and mixed race children popping up all around me"

One can clearly see the hate expressed in the hate speech examples and see their discriminating nature.

4.4 Experimental details

The optimal hyperparameters of the conventional Machine Learning methods are learned automatically as part of the pipeline. Nevertheless the learned optimal hyperparameters are listed in this chapter to enable the replication of our results. For readability only the hyperparameters, which differ from the default values are listed.

Optimal hyperparameters for the unbalanced dataset:

- Decision Tree: max_leaf_nodes=15, min_samples_leaf=10
- Random Forest: criterion='entropy', max_depth=10, max_features='log2'
- SVM: kernel='linear'
- Logistic Regression: C=1.2, solver='lbfgs'

Optimal hyperparameters for the undersampled dataset:

- Decision Tree: class_weight='balanced', criterion='entropy', max_depth=10, max_leaf_nodes=15, min_samples_split=40
- Random Forest: max_depth=10, max_features='sqrt'
- SVM: kernel='linear'
- Logistic Regression: C=1.2, solver='saga'

Optimal hyperparameters for the oversampled dataset:

- Decision Tree: class_weight='balanced', criterion='entropy'
- Random Forest: criterion='entropy', max_features='sqrt'
- SVM: all default parameters
- Logistic Regression: C=1.2, solver='lbfgs'

5 Results and analysis

For answering our research question, whether classical Machine Learning methods combined with suitable features can outperform neural network based approaches, the following results were achieved:

- Investigation of feature importances (chapter 5.1)

- Hate speech statistics (chapter 5.2)
- Comparison of the classifier results (classical Machine Learning methods vs neural network based approaches) (chapter 5.3)
- Oversampled and undersampled datasets (chapter 5.4)

5.1 Feature importances

Table 1: Feature importance scores per classifier

feature	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Unigrams	0.096833	0.110481	-0.116233	-0.340829
Bigrams	0.010989	0.030529	-0.163003	-0.288648
Trigrams	0.015111	0.041289	-1.389380	-2.009609
Hateful Words	0.165297	0.238533	-0.281878	-0.550001
Neutral Words	0.078015	0.060794	-0.092602	0.211546
Exclamation Marks	0.015078	0.014925	-0.16408	-0.39866
Question Marks	0.015837	0.012568	0.077828	0.034885
Full Stop Marks	0.050472	0.039024	0.015701	0.018517
Interjections	0.001223	0.002545	0.084760	-0.255209
All Caps Words	0.030842	0.025809	0.036716	0.119023
Quotation Marks	0.007124	0.011336	-0.083534	-0.197908
Words Total	0.176739	0.107484	0.023188	0.048423
Laughing Expressions	0.002860	0.006506	-0.172888	-0.150478
Pattern Count	0.094512	0.056026	0.005469	-0.017724
Topic	0.030162	0.012664	-0.024878	0.002830
Sentiment	0.208907	0.229479	0.447129	1.180669

5.2 Hate speech statistics

After extracting all the features, we had a closer look at them to identify which features are characteristic for hate speech. Firstly, the semantic features in general do not signify whether a post is hate speech or not. Neither the number of exclamation marks, question marks, full stop marks, interjections or all caps words show any sign of signifying hate speech. These features are evenly distributed regarding hate speech versus non-hate speech. The only semantic features which indicate hate speech are the number of words and - to a very small degree - the number of laughing expressions. As already mentioned in subsection 4.3 the more words a post consists of, the likelier it is to be classified as hate speech (illustrated in Figure 4). Although, there are only very few laughing expressions identified per post (most do not contain any), there is a tendency for hate speech posts to contain more laughing expressions, such as “haha”, “lol” or similar.

Slightly more telling is the topic feature we trained using LDA with only 2 topics. It seems to have somewhat trained to classify into hate and non-hate - as we hoped. The hate speech posts are more likely to be classified as topic 0 than non-hate speech posts. But this difference is not really significant.

A more interesting and characteristic feature seems to be sentiment-based. As described, we extracted a sentiment-score (polarity) for each post using vader and this clearly differentiates between hate speech and non-hate speech, as shown in Figure 5. This shows, that a negative sentiment-score indicates a post being rather likely to contain hate speech. The more positive the sentiment-score is, the less probable it is classified as hate speech.

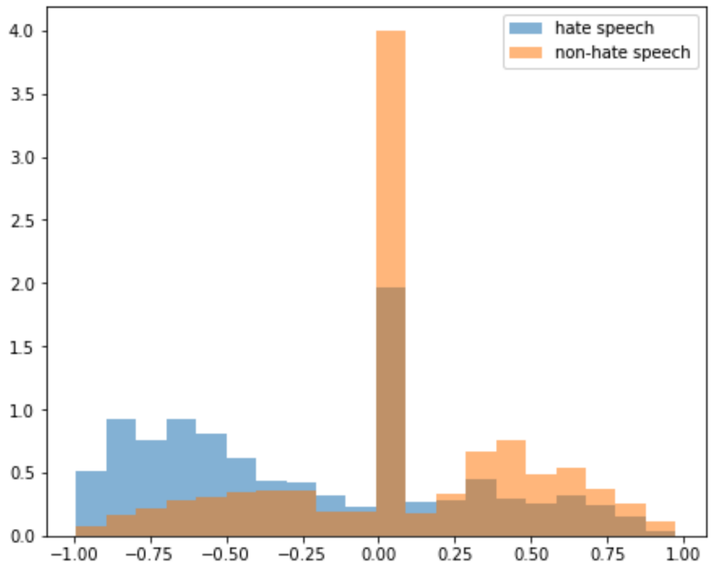


Figure 5: Normalized distribution of sentiment score for hate speech vs. non-hate speech

Further meaningful features were found using a dictionary approach by using the training data to generate a dictionary for hate speech and neutral words. The number of hateful words is distributed such that hate speech posts contain significantly more, whereas the number of neutral words does not differ much. Examples for the most common hateful words found in hate speech posts are “fag”, “bitch”, “ass” or “nigga”. These words basically did not occur in non-hate speech posts. The most common neutral words are less informative, as the suffixes “ll” and “ve” are the most common ones for hate speech and non-hate speech posts.

Furthermore, we had an extensive look at unigrams, bigrams and trigrams for hate speech and these are significantly overrepresented in hate speech posts compared to non-hate speech posts. This especially holds true for the unigrams such as “white” which appears in 15% of hate speech posts or “not” appearing in 9% of hate speech posts. Both of these appear only half as often in non-hate speech posts. The identified bigrams only show up in a very small percentage of posts, but significantly less in non-hate speech posts. Most common bigrams for hate speech are “white trash”, “look like” and “ass nigga”.

Lastly, the feature pattern-count can somewhat indicate hate speech, as the mean amount is higher for hate speech compared to non-hate speech. As one can see in Figure 6 hate speech tends to contain more patterns (which of course were trained by using the hate speech data).

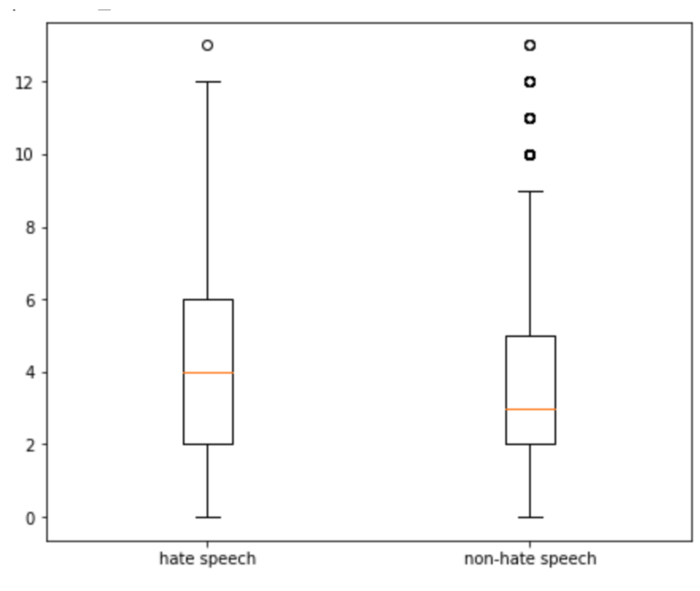


Figure 6: Boxplots comparing the number of patterns occurring in hate speech vs. non-hate speech

But maybe it would be worthwhile to have a closer look at the patterns and adjust them for some future work. Because when we look at single patterns, some occur more often in hate speech and some more often in non-hate speech (in our dataset). For example the pattern “adjective, noun (JJ, NN)" occurs in 56% of hate speech and in 48% of non-hate speech, whereas the pattern "determiner, noun" occurs in 5% less hate speech posts than non-hate speech posts. So a more thorough analysis of these patterns could benefit this feature a lot, maybe by using individual features for each pattern. For example the biggest difference in occurrences is achieved by the pattern "personal pronoun, non-3rd person singular present verb (PRP, VBP)" with a 10% difference.

5.3 Comparison of the classifier results

Table 2 shows the performance metrics of the classifiers for the unbalanced dataset.

Table 2: Classifier results for unbalanced dataset

classifier	**precision**	**recall**	**accuracy**	**F1**
Decision Tree	0.8756	0.9821	0.8671	0.9258
Random Forest	0.8809	0.9894	0.8782	0.9320
SVM	0.8697	0.9927	0.8684	0.9272
Logistic Regression	0.8831	0.9832	0.8760	0.9305
LSTM	0.9219	0.9567	0.8950	0.9390

All classifiers perform about equally well. And the achieved results are good with about 93% for the F1-score. In the unbalanced case the conventional Machine Learning methods can definitely keep up with the neural network baseline.

5.4 Oversampled and undersampled datasets

Table 3 shows the performance metrics of the classifiers for the undersampled dataset and table 4 for the oversampled dataset.

Table 3: Classifier results for undersampled dataset

classifier	**precision**	**recall**	**accuracy**	**F1**
Decision Tree	0.7202	0.7710	0.7431	0.7448
Random Forest	0.7261	0.8043	0.7573	0.7632
SVM	0.7193	0.8375	0.7621	0.7739
Logistic Regression	0.7246	0.8238	0.7621	0.7710
LSTM	0.9219	0.9567	0.8950	0.9390

Table 4: Classifier results for oversampled dataset

classifier	**precision**	**recall**	**accuracy**	**F1**
Decision Tree	0.7973	0.7919	0.7924	0.7946
Random Forest	0.8844	0.8557	0.8701	0.8698
SVM	0.7648	0.8081	0.7767	0.7859
Logistic Regression	0.7573	0.8081	0.7713	0.7819
LSTM	not measured			

In the undersampled case all conventional Machine Learning methods perform about equally well. Compared to the results from unbalanced dataset the conventional classifiers perform worse. In contrast the neural network baseline can keep up with the results from the unbalanced case. So in this undersampled case the conventional Machine Learning methods cannot keep up with the neural network baseline.

In the oversampled case the results are located between the undersampled and the unbalanced case. An outstanding result is the Random Forest, which performs better then the other conventional Machine Learning methods. As already mentioned SMOTE is not able to generate new textual features for generating an oversampled dataset for the neural network baseline. That is why these results could not be measured.

6 Conclusion

Coming to a conclusion, our main research question was whether conventional machine learning approaches combined with suitable features can outperform neural network based approaches. The short answer to this is, that the classical machine learning methods based on our hand-crafted feature set is definitely able to compete with our neural network baseline. For the unbalanced dataset the LSTM baseline only slightly - and rather insignificantly - outperforms the different classical methods, which all perform quite similarly. They also perform quite well with an F1-score around 93%. But it is important to note, that we used grid search to optimize these methods by tuning the hyperparameters, so there is not much room for improvement. For our neural network approach this does not hold true, as we only used a basic implementation without much fine-tuning or testing different network architectures.

Furthermore, while answering this research question we also created a solid hand-crafted feature set based on recent publications. And we developed an easily extendible pipeline incorporating the preprocessing of the underlying data as well as making it easy to add more features and classifiers.

Another finding of our work is the insights into the hate speech statistics and feature importance which lets us have a look at what hate speech is "made" of and indicators for it. The most important features are sentiment and unigram features, with a few words that clearly indicate hate.

As an outlook, there are some improvements and further investigations we would like to evaluate in future work. For one it should be very interesting to expand the currently binary classification into hate speech and non-hate speech to a ternary classification. As this makes the classification more complex this should be easier to achieve with neural network architectures, but it may be interesting to further evaluate the boundaries of the conventional machine learning classifiers. As already mentioned, the hyperparameter tuning for the SVM was not as extensive as it could be, so there might be room for improvement here. Additionally, the gained knowledge from the analysis of the hate speech statistics could help to further improve hate speech patterns based on PoS-tags to achieve better results and a more clear differentiation between hate speech and non-hate speech posts. One could also implement google's bad word list as a separate feature, similar to the hate speech dictionary.

References

[1] Hind Saleh Alatawi, Areej Maatog Alhothali, and Kawthar Mustafa Moria. “Detecting White Supremacist Hate Speech Using Domain Specific Word Embedding with Deep Learning and BERT”. In: *arXiv* (2020). URL: `https://arxiv.org/abs/2010.00357`.

[2] Pinkesh Badjatiya et al. “Deep Learning for Hate Speech Detection in Tweets”. In: *Proceedings of the 26th International Conference on World Wide Web Companion* (2017). DOI: `10.1145/3041021.3054223`.

[3] Nitesh V. Chawla et al. “SMOTE: Synthetic Minority Over-Sampling Technique”. In: *arXiv* (2011). URL: `https://arxiv.org/abs/1106.1813`.

[4] Thomas Davidson et al. “Automated Hate Speech Detection and the Problem of Offensive Language”. In: *arXiv* (2017). URL: `https://arxiv.org/abs/1703.04009`.

[5] Ona De Gibert et al. “Hate Speech Dataset from a White Supremacy Forum”. In: *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)* (2018). DOI: `10.18653/v1/W18-5102`.

[6] Wyatt Dorris et al. “Towards Automatic Detection and Explanation of Hate Speech and Offensive Language”. In: *Proceedings of the Sixth International Workshop on Security and Privacy Analytics* (2020). DOI: `10.1145/3375708.3380312`.

[7] Paula Fortuna and SÃ©rgio Nunes. “A Survey on Automatic Detection of Hate Speech in Text”. In: *ACM Computing Surveys* (2018). DOI: `10.1145/3232676`.

[8] Antigoni Maria Founta et al. “A Unified Deep Learning Architecture for Abuse Detection”. In: *Proceedings of the 10th ACM Conference on Web Science* (2019). DOI: `10.1145/3292522.3326028`.

[9] Aditya Gaydhani et al. “Detecting Hate Speech and Offensive Language on Twitter Using Machine Learning: An N-Gram and TFIDF Based Approach”. In: *arXiv* (2018). URL: `https://arxiv.org/abs/1809.08651`.

[10] Prashant Kapil, Asif Ekbal, and Dipankar Das. “Investigating Deep Learning Approaches for Hate Speech Detection in Social Media”. In: *arXiv* (2020). URL: `https://arxiv.org/abs/2005.14690`.

[11] Shervin Malmasi and Marcos Zampieri. "Detecting Hate Speech in Social Media". In: *Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP)* (2017). DOI: 10.26615/978-954-452-049-6_062.

[12] Rafael Gomes Mantovani et al. *An Empirical Study on Hyperparameter Tuning of Decision Trees*. 2019. arXiv: 1812.02207 [cs.LG].

[13] Iqbal Zulfikar Muhammad, Muhammad Nasrun, and Casi Setianingsih. "Hate Speech Detection Using Global Vector and Deep Belief Network Algorithm". In: *1st International Conference on Big Data Analytics and Practices (IBDAP)* (2020). DOI: 10.1109/IBDAP50342.2020.9245467.

[14] Oluwafemi Oriola and Eduan KotzÃ©. "Evaluating Machine Learning Techniques for Detecting Offensive and Hate Speech in South African Tweets". In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.2968173.

[15] Philipp Probst, Marvin N. Wright, and Anne-Laure Boulesteix. "Hyperparameters and Tuning Strategies for Random Forest". In: *WIREs Data Mining and Knowledge Discovery* 9.3 (2019), e1301. DOI: 10.1002/widm.1301.

[16] Pradeep Kumar Roy et al. "A Framework for Hate Speech Detection Using Deep Convolutional Neural Network". In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.3037073.

[17] Arum Sucia Saksesi, Muhammad Nasrun, and Casi Setianingsih. "Analysis Text of Hate Speech Detection Using Recurrent Neural Network". In: *International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)* (2018). DOI: 10.1109/ICCEREC.2018.8712104.

[18] Syahrul Syafaat Syam, Budhi Irawan, and Casi Setianingsih. "Hate Speech Detection on Twitter Using Long Short-Term Memory (LSTM) Method". In: *4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)* (2019). DOI: 10.1109/ICITISEE48480.2019.9003992.

[19] Hajime Watanabe, Mondher Bouazizi, and Tomoaki Ohtsuki. "Hate Speech on Twitter: A Pragmatic Approach to Collect Hateful and Offensive Expressions and Perform Hate Speech Detection". In: *IEEE Access* 6 (2018). DOI: 10.1109/ACCESS.2018.2806394.