TDDE16 Project - Hate speech identification in tweets, comparing Convolutional Neural Networks, TF-IDF + Logistic Regression and XGBoost

Filip Cornell, filco306

January 21, 2019

Abstract

This article covers the topic of tweet classification, with the intent of classifying different tweets into either being offensive, using offensive language or not being offensive at all. For this, several methods to compare the results are attempted. The tweets are preprocessed and converted to vector representations using Word2Vec or TF-IDF combined with n-grams. Secondly, a convolutional neural network is investigated using a grid-search approach for different values of several hyper-parameters. As a comparison, smaller investigations of the performance of logistic regression and XGBoost are also made. The results show that a TF-IDF approach, combined with logistic regression, achieve similar results as a CNN trained using Word2Vec-representation, with significantly less training time, while XGBoost achieves an almost impeccable precision specifically on hateful tweets, but quite poor results on other metrics compared to the other two methods.

1 Introduction

The internet's ability to permit everyone to speak their mind and share their knowledge opens up tremendous possibility. The amount of information possible to consume has never been greater for the regular person. Social medias such as Facebook, Twitter and Instagram allow each and everyone to live a social life not only in real life, but also online. The internet is thus for most people an enriching experience, and some might even argue that it is the greatest invention of all time [3].

However, not everything the internet has brought can be considered completely positive. The possibility to speak freely also opens up for the less charming sides of the internet; harassment, hate speech and cyberbullying. This is a rising problem, and the possibility to control and regulate this by manually checking harassing posts is not only becoming increasingly difficult, but practically infeasible. As of November 2018, more than 500,000 posts were posted each minute on Facebook, yielding a massive amount of information [1]. If only 1 % of these were to be reported as offensive, this would lead to Facebook's employees having to manually check more than 2.6 billion posts each year - a huge cost of resources. This has been a increasingly pressing issue, and the German government even threatened Facebook to fine € 50 million annually if they would not take serious measures tackle the issue of removing hateful and offensive posts [17]. Manual reviews' inefficiency is not only problematic in terms of resources, but also speed, as the posts are left online until removed, causing the damage it was intended for. Detecting and identifying offensive and hateful posts on social media in a quick, automatic and efficient way is thus a pressing issue, and a lot of work has been done previously within the area. In this paper, we investigate how some models previously tried can be combined in a new way and see what results this might bring. The intent is to use a Convolutional Neural Network

(CNN) together with the word2vec embedding and compare this to the performance of models a TF-IDF-based approach, combined with logistic regression. The method XGBoost is also tested, but only under simple conditions. Although several papers before has used CNNs together with word2vec to classify tweets, few seem to so far have used word2vec as word embeddings (although it has been done for sentences before [10]), but rather GloVe [4, 5, 8]. The paper starts with an introduction to the theory of word2vec, CNNs and short introductions to logistic regression and XGBoost, followed by what previously has been done on classification of posts on social medias. This is followed by the method explained, and a more thorough explanation of the specific model used in this task. The results are then presented, followed by a discussion and conclusion.

2 Theory

In this section, the theory behind the models and tools used are presented, as well as previous work within the field.

2.1 Word2vec

In order to train a neural network to be used for natural language processing (NLP), the text must be mapped from text to vectors using a word embedding [13]. One technique to do this is word2vec, which uses Skip-gram and Continuous Bag-of-words (CBOW) [13]. To shortly summarize it, the set of words are trained through a neural network, and converted into vectors of an arbitrary dimension set by the user.

2.2 Convolutional Neural Network (CNN)

A popular approach when analyzing and classifying text data is to use a CNN, originally mainly used in image and sound recognition [4]. Although CNN:s can be formed in many ways, it has a basic, relatively intuitive structure. First, of all, CNN:s are not fully connected, but rather has one neurone's output going to only one other neurone in the next layer [2]. The first layer is one with convolutional filters. On the matrix inputted, a sliding window, also known as a kernel, feature detector or filter is applied. For each operation the sliding window multiplies each value in the sliding window with the original matrix and sums these values up, creating a new output matrix, also called a feature map [2]. These are sent to a pooling layer which reduces the size of these outputs by summarizing different subregions of the feature map. There are several methods for doing this; one can take the maximum value (max-pooling) or the average value of the sub-region [7]. This basic structure can be used in several layers, along with other types of layers, but these two layers constitute the basic components of a CNN. If the mission of the task is to classify the input, the last layer usually consists of one neurone with the SoftMax function as activation function to classify into the different classes [13]. The SoftMax function can be written as seen in equation 1, where z_i denotes the output of the second last layer, and k denotes class K.

$$softmax(z)[i] = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \forall k \in K \tag{1}$$

When using a CNN, the input size of the features is fixed. The whole sentence (in the NLP case) is thus inputted as a 2D-matrix into the neural network, with the words as rows, and their features as columns. Since, in practice, sentences can differ in length, one must pad the sentences with values to make them fit into the network. In image classification, the squares sent in usually represent images or compressed such [4].

Two common activation functions in the layers in between are the tanh function (equation 2) and Rectified Linear Units (ReLU) function (equation 3). The tanh function, the previous norm, has been shown to have disadvantages compared to the ReLU function in terms of convergence, as the gradient does not saturate as quick. It has been shown that ReLU can converge up to six times faster than tanh [12].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

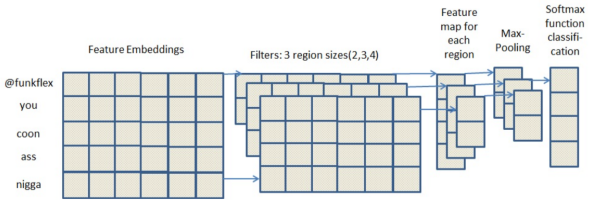

Figure 1: Hate-speech classifier

Figure 1: The layout of the CNN as proposed by Gambäck et al. [8]
. Figure originates from their paper.

$$f(x) = max(0, x) \tag{3}$$

2.3 XGBoost

XGBoost is a method that recently has gained popularity on internet, being the conquering method in many competitions on the site `Kaggle`[1] [6]. The method, similar to the Random Forest method, classifies through modelling a number of trees and summing up the result of the different trees. In the method, each point $\mathbf{x}$ is classified according to the function as seen in equation 4, where $\{f_1, .., f_K\} = \mathcal{F}$ is the family of trees trained through optimizing the function given in equation 5. To explain a few terms, we have that K is the number of trees in $\mathcal{F}$, T_k is the number of leaves for tree f_k and $l(y, \hat{y})$ is the loss function for the problem, which differs depending on the problem.

$$\hat{y} = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}), f_k \in \mathcal{F} \tag{4}$$

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \gamma T_k + \frac{1}{2}\lambda ||w_k||^2 \tag{5}$$

2.4 Logistic regression

Logistic regression is one of the most common approaches in classification. In multiclass classification, a function similar to the SoftMax function (1) is used. The parameters $\beta_{0:p}$ are fitted on the p dimensions to optimally classify according to equation 6.

$$y_i = \frac{e^{X_i^T \beta}}{\sum_{j=1}^{p} e^{X_j^T \beta}} \tag{6}$$

2.5 Related work

Many different approaches have been tried in the field of identifying and classifying hateful speech on the internet before. The first research papers on the topic were published as early as 1997 [15], in which a tree-based classifier managed to reach an accuracy of 88.2 %. The data set was based 720 manually added notes on web page posts from then popular web pages. [15] Over the years, there has been a clear trend of classifying posts on the most popular social medias, such as Twitter and Facebook. In 2016, Waseem and Hovy [18] apply logistic regression combined with character n-gram, achieving an F1-score of 74 %. They are also able to extract the most indicative n-gram features in the set, indicating that features inside "muslim" or "islam" are the most indicative features for racism, while *"bitc"*, *"sex"* or *"xist"* indicate sexual harrassment.

On the same dataset as Waseem and Hovy [18], Gambäck et al. [8] applies a Convolutional Neural Network (CNN) to a set of tweets. To obtain the feature embeddings for the model, `word2vec` and character n-grams were used, while word embeddings were obtained by using `word2vec` and random word vectors. In the neural network, a SoftMax layer calculated the class probability. In between, a pooling layer is utilized to convert each tweet into a vector of fixed length. This pooling layer is followed by a max-pooling layer, used to capture the most important latent semantic factors in each tweet. The model resulted in a precision, at best, of 86 % using random vectors, and an F-score of 78.29 %, obtained through using character n-grams.

Badjatiya P. et al. [5] investigates the possibility to apply three different neural network architectures on classifying tweets; CNN, Long Short-Term Memory (LSTM) and FastText. On these, word embeddings

[1] `https://www.kaggle.com/`

with GloVe- or random beddings are applied. [5] On these, gradient boosting (GBDT) is also applied as learning method. The best result is achieved with LSTM, random embeddings and GBDT, achieving an F-score of 93 %. The GBDT makes a significant difference, seeing that the F-score of LSTM and random embeddings achieves an F-score of 84.8 %.

3 Data

To make this comparable to the recreation of previous work done, the same public datasets are used. With this, we can compare this implementation to the performance of logistic regression, based on a TF-IDF approach [9]. The first two datasets are fetched from `data.world`, where the tweets has been determined as offensive, hate speech or not offensive by different users.[2] [3] The second dataset[4] is a dataset used in three previous works [9, 8, 18], where the data is classified as hateful or not, but rather divided into categories

Investigating the data further after merging the three, the two data sets have a skewedness towards neither offensive nor hateful and offensive tweets, as seen in table 1.

Table 1: Distribution of tweet classes in the entire dataset

Class/Dataset	**1**	**2**	**3**	$\sum$
Normal	7274	4163	11501	22938
Offensive	4836	19190	0	24026
Hate speech	2399	1430	5290	9119
Total	14509	24783	16791	56083

The first dataset contains 20 columns. However, only 2 are significantly interesting for the analysis.

- `does_this_contain_hate_speech` is classified into "The tweet uses offensive language but not hate speech", "The tweet contains hate speech"
- `tweet_text` contains the actual tweet.

The other columns display information such as the confidence of the labelling, based on the judgements from the manual labellers who has labelled the data, as some might have labelled some tweets differently. These would be interesting to incorporate in future work, but is left in this paper for simplification purposes and lack of time. Instead, the column `does_this_contain_hate_speech` will be used.

The second dataset contains only 7 columns, with two columns chosen for analysis in this work.

- `class` - the class as judged by majority of users.
- `tweet` - a string containing the actual tweet to be analyzed.

The data has been labelled by users on CrowdFlower, who has made judgments on whether the tweets are offensive, contains hate speech or neither. Other columns includes information such as number of votes on each class, tweet id and other information. Again, these columns would be interesting to use in further analysis, but is left for simplification.

The third dataset only contains two columns; `class` and `id`, where class refers to "racism" or "sexism", indicating hate speech, or "neither", indicating a normal tweet. Just like A. Gaydhani et al [9], we consider both "racism" and "sexism" to be hatespeech, and "neither" to be a normal tweet. It is important to note that some tweets were labelled as both "neither" and either "sexism" or racism in the dataset, making it impractical to use these as neither. These were thus removed.

It is important to note that since the golden standard has been, in all the datasets, manually labeled, some tweets may have been classified incorrectly. In particular, the distinction between offensive language and hatespeech can sometimes be ambiguous. As an example, the tweet *"Who can I talk to about my feelings. No homo."*[5] has been marked as hate speech, while the tweet "[username] *proof you inbred fuck*"[5], has been marked as offensive. It is thus important to note that it is not probable to achieve a perfect

[2] `https://data.world/crowdflower/hate-speech-identification`

[3] `https://data.world/ml-research/automated-hate-speech-detection-data`

[4] Available at `https://github.com/ZeerakW/hatespeech`

[5] The tweets given as examples does not reflect any of the author's personal beliefs or values.

score, as these definitions of what is hate speech and what isn't has to be thoroughly defined on the training and test set; definitions that might differ when tweets originate from different datasets and investigations, as they do here.

3.1 Preprocessing

First, the datasets are merged. The following preprocessings are then made (in given order).

1. All characters are set to lowercase.
2. All usernames in the tweets are removed.
3. All escaped characters, such as "&gt" and "&lt" are removed.
4. All url:s are removed.
5. The words are stemmed using the Porter Stemmer from the package `nltk`.
6. Some common words are changed, such as "im" to "i'm", "lil" to "little" and so on.
7. All stopwords are removed.
8. All non alpha-numeric character are removed.

Note that step 4, the stemming, will not be performed when using pre-trained word-embeddings.

4 Method

To briefly describe the method for the neural network, each tweet is converted to be represented as a matrix in numerical form using the word embedding `word2vec` using the package `gensim`. If a tweet is shorter than the highest amount of words found in the set after the preprocessing made as described in section 3.1, padding is added to have the same dimensions of every tweet. The data is then split up using different seeds and then CNN is trained using 80 %, of which 20 % of the total amount of data is used for validation. The model is trained in 10 epochs, and is then tested on the remaining 20 % (the test set). The metrics used to measure the performance are then calculated based on the confusion matrix retrieved from the test sets, and the weighted average (weighted by class representation) of each metric will be retrieved.

In order to perform a comparison with A. Gaydhani et al. [9] and ensure that the comparison is fair, we also create their best performing model; logistic regression with a regularizer C set to 100, solved with `liblinear`, using the package `sklearn`. A. Gaydhani et al. uses an n-gram and TF-IDF approach, with n ranging between 1 to 3. TF-IDF is useful since it is able to reduce the negative influence of tokens that are not really informative and does not appear frequently. Each n-gram feature is then weighted in proportion to their TFIDF value. The same approach will here be tried out, to compare the three methods. Thus, the method is here highly inspired by their previous work, available in a Github repository. [6]

XGBoost is only run in a quick fashion, as a first preliminary investigation regarding its performance. It will be run with standard settings in the package `xgboost` in python, with a train and test set, showing possible results and will be evaluated on whether the preliminary results show room for improvement.

All code is available to be looked at and tried out in the Github repository[7].

4.1 Word embedding

The word embedding `word2vec` is applied to the whole dataset. We use two embeddings: one with pre-trained word vectors, trained on more than 2B tweets[8], and one which we train the vectors through the `word2vec`-functions in `gensim` on the entire dataset.

The pre-trained vectors have a dimension of 25, 50, 100 or 200 respectively. When these are used, words will not be stemmed to match the words in the pre-trained vectors.

The instances are run with vector dimensions of

[6] Available at `https://github.com/adityagaydhani14/Toxic-Language-Detection-in-Online-Content/`

[7] `https://github.com/Filco306/Hatespeech-Classification-CNN`

[8] Available at `https://nlp.stanford.edu/projects/glove/`

25, 50, 100 and 200 on the word vectors and set a minimum count of 2 for a word to be included when we train the word vectors on the dataset. In other words, if a word does not exist at least twice, we will not include it as a vector, and it will be considered non-informative and the corresponding vector will be set to a 0-vector. This might yield a loss of information, but as we do allow the CNN to keep training the vectors through `Trainable = True` as input argument to the embedding layer, these might be further trained.

4.2 CNN

The CNN used in this paper has the same architecture as Yoon Kim's for sentence classification [10]. This architecture has previously also been used by Marija Kekic for another sentence classification problem at Kaggle[9]. As input, we will have an NxM matrix with the word embeddings, where N is the maximum amount of words contained in a tweet, and M is the number of dimensions for the embedded vectors. An embedding layer is used, which is initialized with either vectors trained on the dataset with Word2Vec from `gensim` or the pretrained vectors from `GloVe`[8]. The word vectors will pass into three convolutional 2-dimensional layers with convolutions of 3, 4 and 5 respectively with rectified linear units as activation functions due to its faster convergence rate as mentioned by Krishevsky [12]. The output of these ones, are passed on to its respective max-pooling layer, which then passes it on the last classification layer with only one neurone, the Softmax layer. In figure 2, the architecture for the network can be seen. Dropout is also used on the second to last layer. Dropout is a regularization method that prevents overfitting by randomly dropping neurones and their connections from the CNN as it is trained, to prevent that the neurones co-adapt [16].

During training, gradient descent with the ADAM algorithm will be used, which has shown to be computationally efficient and suitable for problems with large datasets [11].

[9]Work available at `https://www.kaggle.com/marijakekic/cnn-in-keras-with-pretrained-word2vec-weights`

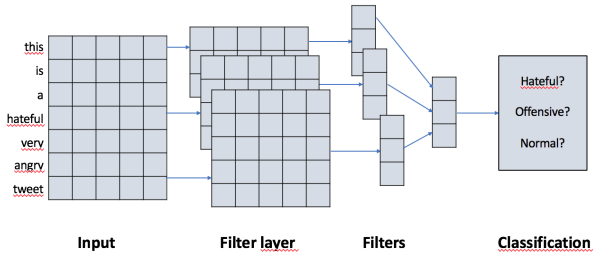

Figure 2: The network architecture used in this paper.

4.3 Metrics

To make this comparable to previous works [8, 5, 9], the same metrics used in those will be used here. That is, the overall accuracy, the precision, recall and the F1-score, three measures commonly used in classification problems. As we have three classes here, it should be clarified that the measures are calculated as follows.

$$P_k = \frac{M_{ii}}{\sum_j M_{j,i}} \tag{7}$$

$$R_k = \frac{M_{ii}}{\sum_j M_{i,j}} \tag{8}$$

$$F1_k = \frac{2 \cdot P_k \cdot R_k}{P_k + R_k} \tag{9}$$

The average scores are then calculated by multiplying the classes' scores with their corresponding percentual representation in the test set.

5 Results

The results can be seen in the tables below. Only the best results of the different settings are presented. To see all results of all runnings, please see the result files in the GitHub repository[10].

Table 5 shows the instances achieving the best overall accuracy, which were chosen for display. Table

[10]`https://github.com/Filco306/Hatespeech-Classification-CNN`

Model	No.
Logistic regression, C = 100	1
XGBoost, pretrained vectors	2
CNN pretrained vectors	3
CNN, trained vectors	4

Table 2: Table showing which model corresponding to which number in other tables.

Type of tweet	Class
Hate speech	0
Offensive	1
Neither	2

Table 3: Explanation which number corresponds to which.

2 shows which model corresponds to which number in the other tables, and table 3 shows which class corresponds to which number in other tables. Table 4 show the different settings that was used for the models whose results are displayed.

Tables 6, 7 and 8 each show the precisions, recalls and F1-scores for each of the respective class for each model chosen.

Model	Dim. size	l_2
1	459384	-
2	25	-
3	100	1
4	100	0.1

Table 4: Values for the models picked out yielding the highest accuracy.

Model	Acc.	Precision	Recall	F1
1	0.867	0.631	**0.747**	0.684
2	0.716	**0.846**	0.525	0.648
3	**0.868**	0.76	0.63	**0.69**
4	0.865	0.62	0.74	0.68

Table 5: The best results during different circumstances of each corresponding model implementation.

Model	0	1	2
1	0.471	0.955	**0.844**
2	**0.997**	0.752	0.679
3	0.468	**0.968**	0.834
4	0.458	0.959	0.838

Table 6: The precisions of the different models selected for the different classes.

Model	0	1	2
1	0.676	**0.867**	0.925
2	0.143	0.802	0.854
3	**0.71**	0.855	**0.945**
4	0.676	0.863	0.926

Table 7: The recalls of the different models selected for the different classes.

Model	0	1	2
1	**0.556**	**0.91**	0.883
2	0.250	0.777	0.756
3	0.564	0.908	**0.886**
4	0.546	0.908	0.88

Table 8: The F1-scores of the different models selected for the different classes.

6 Discussion

On an overall level, the results seen in 5 are fairly promising. However, none of the models seem to match state-of-the-art performance on metrics such as F1-score, precision or recall, when comparing to Badjatiya P. et al. [5] or Gambäck et al. [8]. This might be caused by the difference in data sets used, but also further adjustments to the models made by Badjatiya et al. not made during this investigation. It is however noteworthy that when running the instances of the CNN, the metric optimized by the model was the accuracy. Further investigations should thus focus on rather trying to optimize the precision, recall or F1-score to obtain better results on these metrics, something which is possible through simple modifications of the code. Thus, it is not excluded that better results can be obtained with the right modifications.

What is significant is that the precisions and recalls vary greatly between the classes for all models. Even with oversampling, the precision for the hate speech class does not exceed 0.71 for any CNN instance, and the maximum recall value achieved by any CNN instance is 0.73, while both of these metrics perform much better, at least above 0.83, on the two other classes. This shows how the CNN is sensitive to an imbalanced data set, and should be mitigated. However, although logistic regression has been shown empirically to not be as sensitive as neural networks imbalanced data sets, it does not perform significantly better than any of the CNN instances on any of the measures. The results are in fact very similar on most measures, and none of the methods can be considered completely superior to the other. This, combined with the fact that oversampling did not help to improve the results, indicate that further modifications not identified in this work should be made, and the CNN's sensitivity to imbalance might not be the cause of the CNN's imperfect results. However, the significant difference in training time of the two methods puts logistic regression in favour to further investigate, and see if further improvements could be made to increase the metrics examined. However, the fact that the CNN can be easily modified to rather focus on obtaining a higher precision or recall opens up for further customisation of the model to obtain the practical results one wishes for.

As seen in table 6, XGBoost has a precision near 1 on hateful tweets. This indicates significant differences in the properties and advantages of the different models, as XGBoost tends to favour precision. Although its recall is very poor, this can possibly be adjusted through further adjustments and investigations. As mentioned, XGBoost was only run in a simple fashion as an initial investigation with no tuning over different parameters which indicates a possibility to significantly improve the results, simply by (for example) using cross-validation to tune the hyper-parameters of the model. Possible adjustments might lead to more balanced results between recall and precision, possibly yielding a better F1-score. Out of a practical perspective, a model favouring recall over precision might be preferred, as one would like to automatically catch all *possibly* hateful tweets to then be manually examined, rather than missing some hateful tweets and being able to automatically classify only a small portion of all the hateful tweets. However, being very accurate on some tweets to clean some out early can also be very useful, as it has the possibility to catch a lot of hateful tweets to be automatically removed instantly. Thus, the choice of model might depend highly on the use-case.

As seen in table (4), the most appropriate dimension size of the `word2vec`-vectors were 100 dimensions. All other dimensional sizes did however perform fairly well; none had an accuracy of less than 7 % below the best accuracy achieved. This indicates that while the choice of dimensions does matter, it is not of utmost importance.

Table (8), showing the F1-scores, show that the logistic regression and the CNN:s with pretrained and trained word vectors are more or less equal in results. XGBoost differs, mainly due to its poor recall for class 0, but also worse precision for classes 1 and 2 (offensive and normal tweets).

7 Conclusion

In this investigation, logistic regression with a TF-IDF approach to process the texts was compared

with a simple version of XGBoost and a gridsearch of the optimal values for a CNN using Word2Vec. The results show that the logistic regression model and CNN model are relatively equal in results when one aims to maximize the accuracy, favouring logistic regression due to its significantly quicker training time. The XGBoost method shows excellent results for specifically achieving a high precision on the hateful tweets, which opens up possibilities for this to be used for automatic removal of hateful tweets. Although its recall can be considered poor, it is by all means a method that should be examined whether it could be optimized to achieve better recall while maintaining its precision.

Although the TF-IDF based approach combined with logistic regression had a significantly shorter training time, the possibility of easily customizing the CNN's results depending on which metric one wishes to optimize indicates that further investigations should be made to see if the recall could be increased to open up for manual investigations of the tweets classified as hateful, to reduce the time spent on finding hateful tweets. However, in the end, the optimal scenario would be to find a model with an excellent F1-score, specifically for the hateful tweets' class. This is however a hard challenge; the simple decision of what is hateful and offensive could often prove ambiguous, as seen in the data. Before a perfect classification model could be achieved, a perfect definition of what is a hateful, offensive and non-offensive tweet is would have to be created, which can be harder to agree upon than building a classification model.

8 References

References

[1] Facebook statistics. `https://zephoria.com/top-15-valuable-facebook-statistics/`. Accessed: 2018-12-28.

[2] Introduction and explanation of. `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`. Accessed: 2018-12-28.

[3] List of the greatest inventions of all time. `http://shortsleeveandtieclub.com/the-top-10-inventions-of-all-time/`. Accessed: 2018-12-28.

[4] Hate speech detection using natural language processing techniques. 2018.

[5] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. *CoRR*, abs/1706.00188, 2017.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[7] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285.

[8] Björn Gambäck and Utpal Kumar Sikdar. Using convolutional neural networks to classify hate-speech. 2017.

[9] Aditya Gaydhani, Vikrant Doma, Shrikant Kendre, and Laxmi Bhagwat. Detecting hate speech and offensive language on twitter using machine learning: An n-gram and TFIDF based approach. *CoRR*, abs/1809.08651, 2018.

[10] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[13] Marco Kuhlmann. Lecture, text mining course tdde16, 2018.

[14] Selena Larsson. Twitter doubles tweet length for some users. `https://money.cnn.com/2017/09/26/technology/business/twitter-tweet-280-characters/index.html`, September 27, 2018. Accessed: 2018-12-28.

[15] Ellen Spertus. Smokey: Automatic recognition of hostile messages. pages 1058–1065, July 1997.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[17] Emma Thomasson. German cabinet agrees to fine social media over hate speech. `https://uk.reuters.com/article/uk-germany-hatecrime-facebook/german-cabinet-agrees-to-fine-social-media-over-hate-speech-idUKKBN1771FK`, April 5, 2017. Accessed: 2018-12-28.

[18] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop*, pages 88–93, San Diego, California, June 2016. Association for Computational Linguistics.