RESEARCH PAPER

Hate Speech Detection Using Natural Language Processing Techniques

Author
Shanita Biere

Supervisor
Prof. dr. Sandjai Bhulai

Master Business Analytics
Department of Mathematics
Faculty of Science

August, 2018

Abstract

Hate speech is currently of broad and current interest in the domain of social media. The anonymity and flexibility afforded by the Internet has made it easy for users to communicate in an aggressive manner. And as the amount of online hate speech is increasing, methods that automatically detect hate speech is very much required. Moreover, these problems have also been attracting the Natural Language Processing and Machine Learning communities a lot. Therefore, the goal of this paper is to look at how Natural Language Processing applies in detecting hate speech. Furthermore, this paper also applies a current technique in this field on a dataset.

As neural network approaches outperforms existing methods for text classification problems, a deep learning model has been introduced, namely the Convolutional Neural Network. This classifier assigns each tweet to one of the categories of a Twitter dataset: *hate, offensive language*, and *neither*. The performance of this model has been tested using the accuracy, as well as looking at the precision, recall and F-score. The final model resulted in an accuracy of 91%, precision of 91%, recall of 90% and a F-measure of 90%. However, when looking at each class separately, it should be noted that a lot of *hate* tweets have been misclassified. Therefore, it is recommended to further analyze the predictions and errors, such that more insight is gained on the misclassification.

Contents

1 Introduction

Research on safety and security in social media has grown substantially in the last decade [24], as people are using more and more social interactions on online social networks. This leads to an increase in number of hateful activities that exploit such infrastructure. The anonymity and mobility given by these social media allows people to protect themselves behind a screen and made the breeding and spread of hate speech effortless.

Moreover, social media companies like Twitter, Facebook and Youtube are criticized for not doing enough to prevent hate speech on their sites and have come under pressure to take action against hate speech. As a matter of fact, the German government has threatened to fine the social networks up to 50 million euros per year if they continue to fail to act on hateful postings (and posters) within a week [10].

Due to the massive scale of the web, the need for scalable, automated methods of hate speech detections has grown substantially. These problems have been attracting the Natural Language Processing (NLP) and Machine Learning (ML) communities quite a lot in the last few years. Despite this large amount of work, it remains difficult to compare their performance, largely due to the use of different datasets by each work and the lack of comparative evaluations [43].

The main aim of this paper is to find out how Natural Language Processing techniques can contribute to the detection of hate speech. This research paper also focuses on exploring and applying a current effective method for this classification task on a Twitter dataset.

In the following, Section 2 first discusses some background and related work of this research that is relevant for applying a model. Section 3 then describes the model that has been chosen for detecting hate speech, including a description of the tasks and the specification of the model. In Section 4 a description of the data is given, followed by the results of the experiments. Finally, the paper ends with a conclusion and discussion in Section 5 and 6, respectively.

2 Background and Related Work

This section discusses the background necessary to conduct the research of this paper as defined by the research objective in Section 1. First, an overview of hate speech and the need of detecting hateful speech is given. Then the different techniques of Natural Language Processing is described, which is followed by a review on existing literature where the two disciplines have been fused together.

2.1 Hate Speech Detection

Identifying if a text has hate speech is not an easy task, even not for humans. That is why it is important to give definitions of hate speech before applying machine learning in order to identify hate speech. Given a proper definition makes it easer to tackle this problem. However, there is not one formal definition of hate speech, but a commonly definition is given by Nockleby (2002), that defines it as 'any communication that disparages a person or a group on the basis of some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic.' Examples are [1]:

1. *Go kill yourself, You're a sad little f*ck.*
2. *Wipe out the Jews.*
3. *Women are like grass, they need to be beaten/cut regularly.*

In the very first work of hate speech the term was referred to as *abusive* and *hostile* messages or flames. But, in defining this phenomenon, the words *hate speech* tends to be used the most. More recently, many authors have shifted to employing the term *cyberbullying* [35]. However, there are also more concept related to hate speech that are used in the NLP community, such as: *discrimination, flaming, abusive language, profanity, toxic language or comment* [9].

Examples of hate speech can include racist cartoons, anti-Semitic symbols, ethnic slurs or other derogatory labels for a group, burning a cross, politically incorrect jokes, sexist statements, anti-gay protests...etc [39]. Even though the amount of hate speech online has grown, hate speech itself is not new nor is the aim of it. According to McElwee (2013) hate speech has

[1] It should be noted that these examples are taken from actual web data and thus not reflect the opinion of the author of this paper.

the intention of achieving at most two goals: an attempt to tell bigots that they are not alone and to intimidate the targeted minority, leading them to question whether their dignity and social status is secure.

While hate speech itself is not new, hate speech detections is a recent area. Detecting hate speech has become an important part for analyzing public sentiment of a group of users towards another group, and for discouraging associated wrongful activities [4]. Manually filtering these hateful text on the web is seen as a lot of work and thus not scalable at all, which is why researcher were determined to find automated ways to detect hateful text.

An important note when detecting hate speech is that it should not be mixed with offensive language. Because, someone using offensive language is not automatically abusive. People tend to use terms that are technically speaking highly offensive, for all sorts of reasons. Some in a playful way, others in qualitatively different ways. For example, people use some terms in everyday language or when quoting rap lyrics or even for fun, such as *n*gga, h*e* and *b*tch*. Such language is prevalent on social media, making this boundary condition crucial for any usable hate speech detection system [7].

2.2 Natural Language Processing

Natural Language Processing or **NLP** (also called Computational Linguistics) can be defined as the automatic processing of human languages. As NLP is a large and multidisciplinary field, but yet comparatively a new area, there are many definitions out there practiced by different people. One definition that would be part of any knowledgeable person's definition is [22]:

Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

Thus, NLP is a field of computer science and linguistics concerned with the interaction between computers and human (natural) languages. Moreover, it is driven by advances in Machine Learning (ML) and also has an integral part of Artificial Intelligence (AI). The techniques of NLP are de-

veloped in such a way that the commands given in natural language can be understood by the computer and also be able to perform according to it. It should be noted that natural language processing can be divided into two parts, namely written and spoken language [2]. Written languages play a less central role than speech in most activities, as the largest part of human linguistic communication occurs as speech. However, written language can be understood easier than spoken language, as spoken languages deal with a lot of noise and ambiguities of the audio signal. Because there is a lot of ambiguity found in language, NLP is seen as a hard problem in computer science.

Research in natural language processing has been going on since the late 1940s. Machine translation (MT) was one of the first computer-based application related to natural language. Cambria and White (2014) comments that NLP research has evolved from the era of punch cards and batch processing, in which the analysis of a sentence could take up to 7 minutes, to the era of Google and the likes of it, in which millions of webpages can be processed in less than a second.

The most explanatory method for presenting what actually happens within a Natural Language Processing system is by means of the 'levels of language' approach [22]. These levels are used by people to extract the meaning from text or spoken languages. This is because language processing mainly relies on formal models or representation of knowledge related to these levels [19]. Moreover, language processing applications distinguish themselves from data processing systems by using the knowledge of language. The analysis of natural language processing have the following levels: Phonology, Morphology, Lexical, Syntactic, Semantic, Discourse and Pragmatic. The meaning of each and every level can be found in the appendix.

2.2.1 Major Tasks in Natural Language Processing

There are a range of applications available that have both theory and implementations in it. As a matter of fact, any application that makes use of text is a candidate for natural language processing. An overview of the most used applications in NLP can be found in the appendix. Most

[2]This paper focuses on the language understanding part, thus speech understanding will be beyond the scope of this paper. Moreover, work on speech processing has evolved into a separate field.

of these problems can be formalized as five major tasks, namely classification, matching, translation, structured prediction, and sequential decision process [21]. Table 2.1 gives an overview of these tasks.

Task	Description	Applications
Classification	Assign a label to a string	Text classification, Sentiment analysis
Matching	Matching two strings	Search, Question answering
Translation	Transform one string to another	Machine translation, Speech recognition
Structured prediction	Map a string to a structure	Named entity recognition, Word segmentation, Semantic parsing
Sequential decision process	Take actions in states in dynamically changing environment	Multi-turn dialogue

Table 2.1: Five Tasks in Natural Language Processing

2.2.2 Approaches in Natural Language Processing

Liddy (2001) describes that Natural Language Processing approaches fall into four categories:

Symbolic approaches perform deep analysis of linguistic phenomena and are based on explicit representation of facts about language through well-understood knowledge representation schemes and associated algorithms [31].

Statistical approaches employ various mathematical techniques and often use large text corpora to develop approximate generalized models of linguistic phenomena based on actual examples of these phenomena provided by the text corpora without adding significant linguistic or world knowledge [22].

Connectionist approaches also develop generalized models of linguistic phenomena, just like the statistical approaches. But what separates connectionism, also known as "parallel distributed processing", "neural networks" or "neuro-computing", from other statistical methods is that connectionist models combine statistical learning with various theories of represen-

tation - thus the connectionist representations allow transformation, inference, and manipulation of logic formulae [22].

An increasing number of researchers combine data-driven and knowledge-driven approaches, which is called the *hybrid approaches* [11].

From the above, it can be seen that there are similarities and differences between the approaches. For example, each approach has different assumptions, philosophical foundations, and source of evidence. Moreover, the existing methods for text classification can be divided into two categories: classical methods and deep learning methods.

Classical methods rely on manual feature engineering and rules in combination with statistical algorithms. The manually designing features of data instances into feature vectors can be done in several ways. Studies have shown that the most effective surface features in hate speech detection are bag of words, word and character n-grams. In terms of classifiers, the most popular algorithm used, is the Support Vector Machine. Algorithms like Naive Bayes, Logistic Regression and Random Forest are also used for classification task.

Support Vector Machines

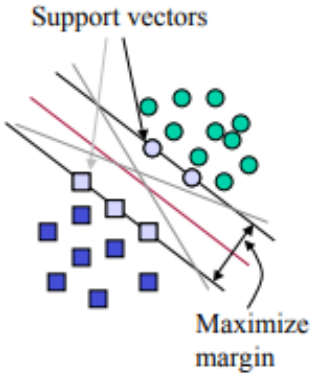


Figure 2.1: Maximum margin hyperplane in SVM [2].

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression Vapnik [40], but mainly target classification problems. This technique does not make any assumptions about the data (non-parametric). Its basic concept is to design a hyperplane that divides all instances into two sets. The points on one of these two hyperplanes are called the support vectors. As not all data can be linearly separated, SVM use a kernel function that maps training vectors to a higher dimensional space where a maximal separating hyperplane is constructed, which makes it non-linear. An advantage of SVMs is that it does not calculate all the points in the new feature space after applying the kernel func-

tion, but performs the calculations in the initial feature space, also called the kernel trick [12].

Deep learning methods make use of neural networks to automatically learn multi-layers of features from the given data. By the early 2000s, improvements in computer hardware and advances in optimization and training techniques made it possible to train even larger and deeper networks, leading to the modern term deep learning [15]. NLP techniques were mostly dominated by machine-learning approaches that use linear models and are trained over high dimensional yet very sparse feature vectors. However, lately non-linear neural-network over dense inputs have been showing success. The most popular used networks are Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), typically Long Short-Term Memory network (LSTM). In the literature, CNN is well known as an effective network to act as 'feature extractors', whereas RNN is good for modeling orderly sequence learning problems [29].

Recurrent Neural Network

Recurrent models have been shown to produce very strong results for language modeling. Recurrent neural networks (RNNs) are so-called feedback neural networks where the connections between neurons can form directed cycles. This network analyzes a text word by word and stores the semantics of all the previous text in a hidden layer. A recurrent neural network is a neural network that consists of a hidden state h and an optional output y that operates on a variable length sequence x = $(x_1, ..., x_T)$. At each time step t, the hidden state $h_{(t)}$ of the RNN is updated by

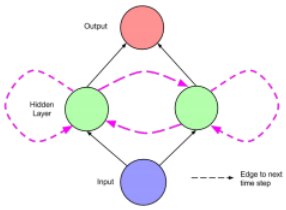


Figure 2.2: A simple recurrent network [41].

$$h_{(t)} = f(h_{(t-1)}, x_t), \tag{2.1}$$

where f is a non-linear activation function [20]. This function could be simple, but also very complex such as the *long short-term memory* (LSTM) unit. LSTM is one of the most popular and efficient methods for reducing the effects of vanishing and exploding gradients [14]. It is thus capable of learn-

ing long-term dependencies.

RNN has its advantage of capturing the contextual information in a better way, which could be helpful when dealing with long text. However, as this model is a biased model, it could also reduce the effectiveness when used for the entire document.

Convolutional Neural Network

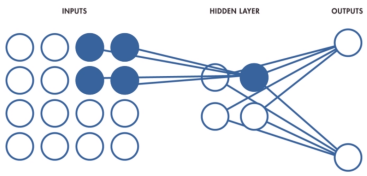


Figure 2.3: Concept of a Convolutional Neural Network, where only a small region of input layer neurons connect to neurons in the hidden layer [1].

To tackle the bias problem of a recurrent neural network, an unbiased model has been introduced, namely the convolutional neural network, referred to as CNN or ConvNet. A CNN is a deep, feed-forward artificial neural network consisting of an input, output and multiple hidden layers. The difference of a CNN is that there is a convolutional multi-layer that automatically identifies features within the input space. The networks are mostly known in applications as image and sound recognition, but have lately been used in more applications, such as text classification.

In the context of hate speech classification, intuitively, CNN extracts word or character combinations (e.g., phrases, n-grams), RNN learns word or character dependencies (orderly information) in Tweets [42].

Ensemble methods. At last, researchers also tend to use ensemble methods to improve the performance of the model. This method is a method that uses a combination of multiple independent models to aggregate the predictions. Several papers have, in fact, shown that using an ensemble method delivers outstanding performance on the training model and also has much success in reducing the testing error. Since this research paper will not be using ensemble methods, it is not needed to go into the depth of this method. However, as these methods are applied in practice quite a lot, naming it was important such that reader is aware of it.

2.3 Natural Language Processing in Hate Speech Detection

So far, both hate speech detection and natural language processing have been discussed independently. In this section, the aim is to look at prior work where hate speech has been detected with natural language processing techniques.

The task of identifying hate speech and abusive language has been in the research community for almost 20 years now, which lead to a lot of studies on computational methods in the last few years. A number of studies that have been done on hate speech detection in terms of classifiers rely on supervised learning approaches (classical methods). One of the most used one are *Support Vector Machines (SVM)* [35]. Examples where SVMs were used include the work by Malmasi and Zampieri (2017), Davidson et al. (2017) and Robinson et al. (2018).

Badjatiya et al. (2017) deploy deep learning to detect hate speech in Tweets. They used three neural network techniques, where the word embeddings were initialize with either random embeddings or GloVe embeddings. The following methods were deployed: (1) *Convolutional Neural Network (CNN)*, (2) *Long short-term memory (LTSM)* and (3) *FastText*. The experiments conducted showed that CNN performed better than LSTM which was better than FastText. Moreover, they concluded that embeddings learned from deep neural network models when combined with gradient boosted decision trees led to best accuracy values, which significantly outperforms the existing methods.

Pitsilis et al. (2018) also conducted a research on detecting hate speech in Tweets using deep learning. They used an ensemble classifier (deep learning architecture) that uses word frequency vectorization for implementing the given features. Their result also outperforms the existing methods. Moreover, they also concluded that no other model has achieved better performance in classifying short messages.

3 Convolutional Neural Network

The literature study in Section 2 showed that deep learning models tend to outperform classical models in text classification. This gives enough reasons to further investigate and implement a deep learning model, such as a convolutional neural network.

3.1 Architecture

There are a lot of variants of the architectures of a convolutional network that are shown effective in classification tasks. Below an example of a simple convolutional architecture is shown.

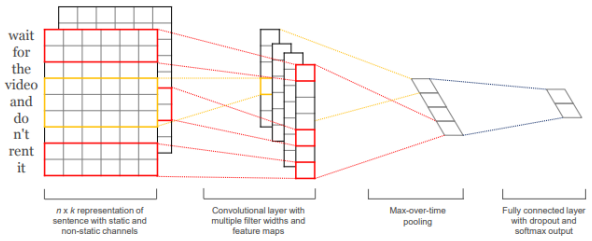


Figure 1: Model architecture with two channels for an example sentence.

Figure 3.1: Model architecture for an example sentence [17].

As mentioned before, a CNN consists of an input and output layer, as well as multiple hidden layers. These hidden layers are connected in a feed-forward manner (thus no cycles in which outputs of the model are fed back into itself).

The input layer. This layer normally consists of an image that has 3D inputs of size width x height x 3. This is because CNNs were originally developed for image data, which is fixed-sized, low-dimensional and dense. This does not mean that CNNs can not be used to categorize other types of data too. In order to use these models for other data types, first transform it to make it look like an image. Thus, applying to text documents, these need to be modified such that they are variable-sized, high-dimensional and sparse if represented by sequences of one-hot vectors [13].

Embedding layer. In the embedding layer, words with similar meaning tend to occur in similar context, which converts words into low-dimensional vectors. As mentioned before, when dealing with text data, the words need to be converted into vectors. However, converting words to vectors means that each of these words have a one-hot vector. Representing words in this way may lead to substantial data sparsity and thus if there is not enough data, models can perform very poorly or may even overfit and lead to the curse of dimensionality. Thus, to overcome these shortcomings, words or characters are transformed into embeddings, where semantically similar words are mapped to a nearby point. The main advantage of these vectors is that they capture similarity between words.The embedding layer is often used as the first data processing layer in deep learning models.

Measuring the similarity between vectors can be done in two ways: count-based methods and predictive methods. The first method is based on the Firth's hypothesis from 1957 that the meaning of a word is defined "by the company it keeps" and leads to a very simple albeit a very high-dimensional word embedding [25]. The second method tries to predict a word from its neighbors in terms of learned small, dense embedding vectors. Commonly used ones are the GloVe and Word2Vec models.

The embeddings have proven to be efficient in capturing context similarity, analogies and due to its smaller dimensionality, are fast and efficient in computing core NLP tasks [37].

Convolutional layer. When working with convolutional networks, the convolutional layer is the core building block that does most of the computational heavy lifting [16]. The function of convolutional layers is to reduce the number of weights by extracting higher level features from the input matrix. As known, each row of a matrix is a vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary [5]. As seen in Fig. 3.2 every node in this layer corresponds to

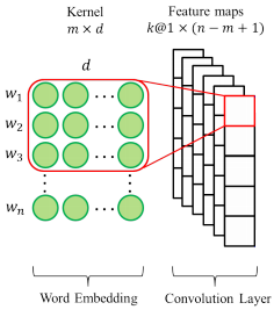


Figure 3.2: An illustration of a convolutional filter from [38].

a filter (or 'kernel') that connects it to a local region. In the case of an image, filters of size F[1] x F x d slide over local patches of the image, while in text data, filters of size m x d slide over the input matrix. Note that the width of the filters is the same as the width of the input matrix, so the connection is only limited in height. The filter is then applied to each possible window of words in the sentence to produce a *feature map*.

Pooling layer. Other than using convolutional layers, CNNs also use pooling layers to reduce the size of the representations, thus reducing its variance. Pooling layers are designed to subsample their input, which is the output of the convolutional layer that is passed to the these layers. The purpose of pooling is to transform the joint feature representation into a more usable one that preserves important information while discarding irrelevant details [8]. Moreover, as classification requires fixed size output matrix, this layer also provides that. This allows the use of variable size sentences, and variable size filters, but always obtaining the same output dimensions to feed into a classifier [23]. Formerly, the most common way to do pooling was to compute averages, but it has been shown that applying the maximum to the result of each filter works better.

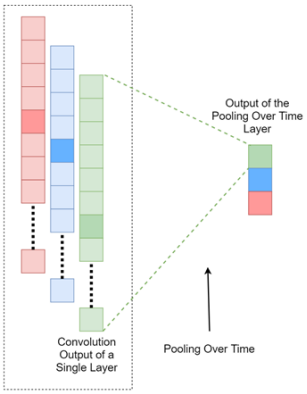


Figure 3.3: The pooling over time operation [32].

Fully-connected layer. After the convolutional and pooling layers, the final layer in the network is called the fully-connected layer. This layer basically gives every value a vote and thus activating the output of each convolutional layer. It flattens the high-level features that are learned by the previous layers and combines them. Afterward, the output of this layer is passed to the output layer where a non-linear function, like a softmax classifier or a sigmoid activation, is used for prediction.

[1] F is the region where the filter slides, which is called the *receptive field*.

3.2 Hyperparameters

While there are many aspects of CNNs that can be learned automatically (such as the backpropagation and gradient descent), there are also aspects that cannot be learned by the computer and needs to be guessed by the user. Building a CNN architecture means that there are many decisions to make, thus many hyperparameters to choose from. While some parameters have been presented in the section above, there are a few more parameters that need to be tuned. A list of these parameters will be described below.

Input representations. Using one-hot encoding or word embeddings, one can consider to train a CNN from scratch, thus not implementing pre-trained word vectors like word2Vec or Glove and directly applying convolutions to one-hot vectors. However, studies have shown that using one-hot encoding performs well on long text. While pre-trained word embedding showed better performances for short text.

Convolution filters. Sliding the filters over the input volume needs to be specified by:

1. the *number* of filters
2. the *size* of the filters
3. the *stride* size, determining the number of shifts at each step
4. the amount of *zero-padding*, which can be used to adjust the size of the input (by symmetrically adding zeroes to the input matrix)

Typical choices are to slide over 2-5 words at a time. When one uses large filters, it is then useful or in some cases even necessary to add zero-padding. Moreover, when looking at the stride size, it was noticeable that a lot of studies used sizes of 1.

Pooling strategies. As mentioned before, there are two conventional pooling methods: max pooling and average pooling. Choosing between these two highly depends on the kind of application one wants. However, in the literature it has been found that max pooling is used more often and works quite well in a lot of experiments. A few results that stand out even concludes that max-pooling always beats average pooling [5].

Activation functions. Every neural network applies activations functions which are used to find out which node needs to be fired. There are many different functions that can be used, the most common onces are: Sigmoid, Tanh, Softmax, and rectified linear unit (ReLU). For the hidden layers, a recent study of Ramachandran et al. (2017) shows that the current most successful activation function is the ReLU function with $f(x) = max(0, x)$. When looking at the output layer (the very last layer), typically softmax or sigmoid functions are used. The choice between these two depends on whether the classes are independent or mutually exclusive. In the latter case, the soft-max activation should be used [3], as it outputs a probability distribution, while the sigmoid function outputs marginal probabilities.

In addition to the above, there are also other architectural decisions that may need some attention. For example, some neural networks can have a lot of layers while other networks do not. This means that it is also needed to look at the number of the layers that needs to be included. Moreover, one can also think of the order in which these layers take place.

3.3 Evaluation

To evaluate the model with certain features and parameters, the data is divided into three separate sets: training, validation and test set.

The training set is used to optimize the parameters of the model, while the validation set is used to tune in the hyperparameters in order to improve the performances. The validation set can be used to set a stopping criteria for learning and thus avoids overfitting [12]. However, the more the models are tuned on the validation set, the more biased it becomes. This is why a test set is used, such that it can obtain an unbiased evaluation.

When dealing with small datasets, it is best to use the limited data in the best possible way. One approach is to apply k-fold cross validation that splits the training data into *k chunks* and then uses *k-1 k chunks* to train on and one *chuck* to test on. However, as this approach repeats the hold-out method k times, it can be computationally quite expensive, especially when using a convolutional network and the use of a large k. But, taking a smaller value for k, might lead to poor performances and less accuracy. This paper however will not focus on determining k in finding a good balance. This is why this approach is beyond the scope of this research paper.

Furthermore, it is also important to evaluate the performance of the different classifiers. This can be done by evaluation measures that can be divided into a *loss* and *metric* functions.

Loss function is used to optimize the model and thus measures how well the algorithm is doing on the dataset. A few popular loss functions that are currently being used are: mean squared error (MSE), likelihood loss and cross entropy loss.

For the loss function the *categorical cross-entropy* loss function is used to obtain the results. This function is preferred rather than the other commonly used function , because from the literature it is know that the cross-entropy tend to be more effective on classification tasks [26].

The categorical cross-entropy loss measures the dissimilarity between the true label distribution y and the predicted label distribution $\hat{y}$, and is defined as cross entropy [18].

Metric function is used to judge the performance of the model and has nothing to do with the optimization process. These are often based on confusion matrix: a table that describes the performance of the classifier as seen in Fig. 3.4. The example of the matrix is used for a binary classification, but can easily be applied to a multiclass classification by adding more rows and columns to the confusion matrix [6].

Data class	Classified as *pos*	Classified as *neg*
pos	true positive (*tp*)	false negative (*fn*)
neg	false positive (*fp*)	true negative (*tn*)

$$\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix}$$

Figure 3.4: Confusion matrix for binary classification from [36].

Based on the values in the table above, a couple of measurements were created. One of the well-know metric and probably the most straightforward one is the *accuracy* that measures the total correct predictions as a percentage of the total instances. It is defined as follows:

$$Accuracy = \frac{tp + tn}{tp + fn + fp + tn}$$

However, looking at the accuracy only to decide whether the model is good, is not enough to make this decision. Another popular metric for classification problems is the F_1-*score*, which is the harmonic mean of *precision* and *recall* measurements. These are defined as:

$$Recall = \frac{tp}{tp + fn}, \quad Precision = \frac{tp}{tp + fp}, \quad and \;\; F_1 = 2 \cdot \frac{recall \cdot precision}{recall + precision}$$

In case of an unbalanced dataset, a F_1-*score* would also give a better unbiased representation of the performance than an accuracy measure.

4 Experiments

4.1 Data

To see how well the proposed method performs, the experiments have been done on the Hate Speech Identification dataset distributed via CrowdFlower [1]. This dataset features 24,783 English tweets that has been classified into three classes:

1. *Hate*: tweet contains hate speech
2. *Offensive*: tweet contains offensive language but no hate speech
3. *Neither*: tweet does not contain hate speech nor offensive language

The distribution of the tweets across the three classes is shown in Table 4.1. These numbers indicate that approximately 5% of the tweets contain hate speech, while the majority of the tweets (77%) contains offensive language. This means that the number of tweets belonging to the three classes are quite skewed, leading to an unbalanced dataset. The size of the dataset is rather small, but will still be used for this research.

Class	**# of Tweets**
Hate	1,430
Offensive	19,190
Neither	4,163
Total	24,783

Table 4.1: Summary classes

4.2 Preprocessing

Given a tweet, the following preprocessing procedure has been adopted to normalize its content. This has been done with the *Keras* package in Python using its *Tokenizer* class.

- removing characters
- lowercase and stemming, to reduce word inflections
- splitting text into tokens

There are also many tools available to clean Twitter datasets, but were not taking into consideration, meaning that it is beyond the scope of this paper.

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

4.3 Experimental setup

As mentioned before, the experiments have been done in Python. Here the neural network and machine learning libraries are utilized. To be more specific, for the training of the model, the Keras library with Tensorflow back-end has been used. This subsection describes the setup of the experiments that assessed the performance of the convolutional neural network.

Data splitting. For this research paper it has been decided to split the data into the three separate sets, instead of using cross-validation. The reason for this is the short time span of this research. Training a CNN is computationally expensive and with the use of cross-validation it would take too much time. Thus, the division has split the dataset in a training, validation, and test set of 50%, 30%, and 20% respectively. The rationale behind this ratio comes from the size of the whole dataset. To evaluate the model and refine the parameters, a larger validation set is needed.

Architecture. The previous section showed that convolutional networks are commonly made up of only three layer types: convolution, pooling and fully-connected. Several studies have shown that it is very uncommon to have resources to succesfully train a full convolutional neural network from scratch. Karpathy (2017) even says that is it more sensible to look at the current architecture that works best for your problem and then download a pre-trained model and fine-tune it on your data. Following this advice, the CNN model that has been used on the dataset is the architecture used by Kim (2014) (see Fig. 3.1), which consists of a non-linear convolution layer, max-pooling layer, and softmax layer. The architecture of Kim is used for a sentence classification, which is why it is expected that a big part of the model is transferable to this hate speech problem.

Input + word embedding. The input of this model is a preprocessed tweet, that is treated as a sequence of words. To set the weight of the embedding layer, this work used the publicly available word2Vec word embedding with 300 dimensions pre-trained on the 3-billion-word from Google News with a skip-gram model.[2]

[2]https://github.com/mmihaltz/word2vec-GoogleNews-vectors

Hyperparameters. As described above, the model parameters will be based on default values or on (empirical) findings that have been reported earlier. However, the batch size and epoches are derived from the training model, which will be explained later. It should be noted that these values may not lead to optimal results, as each setting is dependent of the data. Nonetheless, this work will show that using these parameters leads to obtaining good performances even without tuning the parameters.

CNN. The dimension of the word vector is set to d = 100 at first and thus the *embedding layer* passes an input feature space that has a 3-dimensional tensor of shape `(None, 100, 300)`. The output of this layer is then fed into a 2D *convolutional layer* with filter layers of 3, 4, and 5, each having a 100 feature map. The rectified linear unit function is used for activation. In order to build the convolutional layers followed by max-pooling, it is needed to convert the output of the embedding in a 4-dimensional tensor shape[3]. The layer of the shape then becomes `(None, 100, 300, 1)`. As this network deals with filters of different sizes, each filter has its own layer, which is then merged into one feature vector. The filters are then slided over the sentences without padding the edges. For example, the filter that slides over 3 words gives a tensor of shape `(None, 94`[4]`, 1, 100)`. This input feature space is then further down-sampled by a *max pooling layer* with a stride size of 1, which gives a tensor of shape `(None, 1, 1, 100)`. Once all the output tensor from the pooled layers are created, each filter size is then combines into one long feature vector. Then, the activations are *dropped* randomly with the probability p = 0.5 and the dimension is flatten when possible. At last, the output feature vector is then taken as input in a *softmax layer*. This layer then predicts the probability distribution over all possible classes.

Optimization. To train the model, the Adam algorithm has been used. Furthermore, default parameters given in the paper of Kim (2014) are used and if necessary are adjusted to get a better performance. The batch size is set to *64* and the model will be trained with *10* epochs.

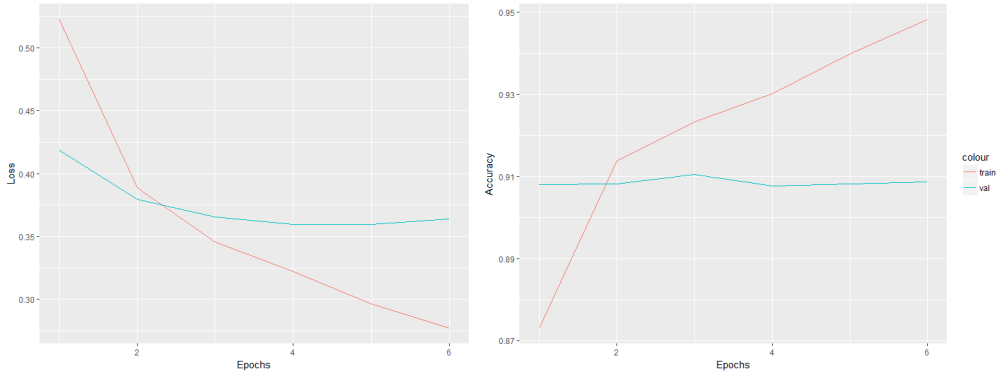
[3]When using a 2D convolution, the Tensorflow in Python only takes a 4-dimensional tensor with dimensions corresponding to batch, width, height and channel.

[4](= sequence length - filter size + 1)

4.4 Results

Before finalizing the model, it is important to see how the model performs by using the validation set. This will give an indication whether the parameters needs to be tuned in for a better performance. Fig. 4.1 gives an overview of how the classifier with the given parameters (as discussed previously) has performed.

Initially, a learning rate of $\alpha = 10^{-3}$, batch size of $B = 64$ and a dropout probability of $p = 0.5$ have been applied. A plot of the loss function is given in Fig 4.1a.



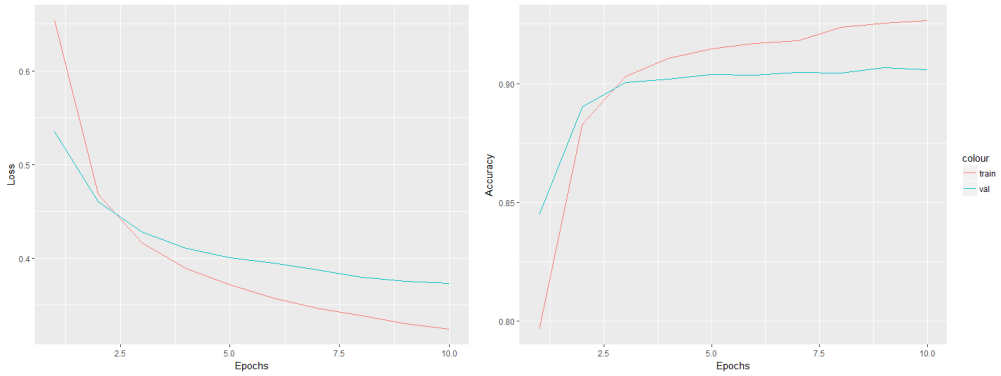
(a) Loss (b) Accuracy

Figure 4.1: The loss and accuracy for the initial model.

Fig. 4.1b shows the accuracy performance of the model. This shows whether the classifier is overfitting the training data. Looking at both plots, it can be seen that the model tends to overfit the training data. Moreover, these plots indicates that the learning rate may be decreased [5].

Thus, based on these observations, the training model has been retrained with the same batch size and probability, but with a lower learning rate of $\alpha = 10^{-4}$. The results of this attempt is given in Fig. 4.2. Here it can be seen that the model did benefit from the lower learning rate. It can also be seen that the model overfits the training data a bit. However, the performance of the accuracy for the training and validation do not differ too much. Therefore, a little overfitting of the training is somewhat affordable.

[5] See Appendix on how to derive these developments



(a) Loss (b) Accuracy

Figure 4.2: The loss and accuracy after the first changes.

It should also be noted again that this model has been tested on a fairly small dataset. Therefore, to reduce overfitting, it is probably desirable to collect more data.

Final results. The parameters described above results in the model predicting each category with 91% accuracy and a loss of 36%. The final model gives an overall precision of 0.91, a recall of 0.90 and a F_1-score of 0.90. Looking at Fig.4.3, it can be observed that the model overall did not identify many tweets as hate-speech tweets: almost 80% of the *hate* class is misclassified. This may be due to insufficient and unbalanced training data, as mentioned in Section 4.1. Which leads the model to be biased towards classifying tweets as offensive. Furthermore, the model also incorrectly identified some *non-hate* speech as *hate* speech. However it did perform better in identifying the *offensive* class. This is because the number of tweets that are categorized as *offensive* are larger than the other two categories.

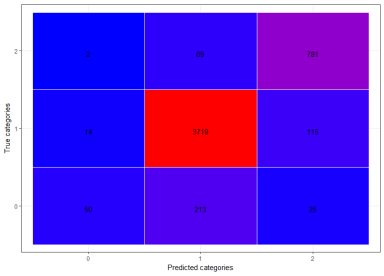


Figure 4.3: Confution matrix

5 | Conclusion

In this paper, the aim was to detect hate speech using a Natural Language Processing technique. To enable successful execution of the research it was first necessary to understand what hate speech is. To accomplishing this, an overview of this topic has been conducted. Here it can be concluded that hate speech has several definition, all coming from different platforms. Hate speech detection is a classification-related tasks, and that is why further literature was reviewed to understand the idea behind Natural Language Processing and the application of various techniques. Previous work showed that deep learning models improve the state-of-art approaches within hate speech classification tasks. Therefore, a deep learning method, namely a Convolutional Neural Network (CNN), has been applied on a Twitter dataset. This data contains tweets annotated with three labels: hate, offensive language and neither.

From the results it can be concluded that the simple CNN architecture used by Kim (2014) obtained good performances (91% accuracy). However, while the overall precision, recall and F-score were also good, the model incorrectly identified some *non-hate* speech as *hate* speech. Moreover, the majority of the *hate* class is misclassified, while the majority of the offensive class is correctly identified. This is because the majority of the classes of the data contains *offensive language*. This leads the model to be biased towards classifying tweets as *offensive*.

Nevertheless, as mentioned in Section 2 identifying hate speech is not an easy task. The fact that hate speech is a difficult phenomenon to define, makes it even harder for users to classify a text as hate. These classification namely tend to reflect the subjective biases. However, if datasets are richer, both in size and quality, CNNs have great potential to give good performances.

6 Future Work

Error analysis. As seen in Fig. 4.3 there are a lot of tweets that have been misclassified, namely in classifying hate speech. An error analysis could therefore help to provide insights about the performance of the model. For example, when examining the wrong predictions of the *hate* class, this could help with clarifying why this class is so hard to predict. It would be interesting to see if and how certain terms are useful for distinguishing between hate speech and offensive language.

Data. There are three ways of showing hate on Twitter: directly to a person or group, in a conversation between people, and randomly to nobody in particular. In future work, when looking at hate speech, it can also be interesting to look at the distinguish between the three ways that people show hatred on Twitter. Future work can also focus on the individual characteristics and motivation of a user for example. And of course, if possible it would be better to collect more data to make a more accurate distinction between the model performances..

Ensemble. In Section 2, there is also a subsection about ensemble methods. Previous work have used ensemble methods and achieved success in classification tasks. Therefore, it is expected that these methods can further improve the results that has been obtained.

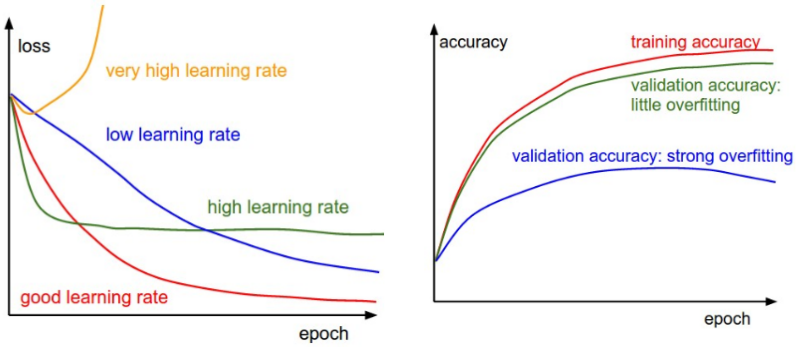
Appendix

A. Levels of Language

- Phonetics and Phonology — knowledge of linguistic sounds
- Morphology — knowledge of the meaningful components of words
- Syntax — knowledge of the structural relationships between words
- Semantics — knowledge of meaning
- Pragmatics — knowledge of the relationship of meaning to the goals and intentions of the speaker

- Discourse — knowledge about linguistic units larger than a single utterance

B.



(a) Development of the loss with different learning rates [16]

(b) Illustration of strong or little overfitting

Figure 1

Bibliography

[1] Introduction to deep learning: What are convolutional neural networks? https://www.mathworks.com/videos/series/introduction-to-deep-learning.html. Accessed: 2018-07-12.

[2] An idiot's guide to support vector machines (svms). http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.

[3] Softmax regression. http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression#Softmax_Regression_vs._k_Binary_Classifiers.

[4] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. 2017.

[5] Denny Britz. Understanding convolutional neural networks for nlp. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[6] Jason Brownlee. What is a confusion matrix in machine learning. https://machinelearningmastery.com/confusion-matrix-machine-learning/, 2016.

[7] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. 2017.

[8] Peiqiu Chen Dingjun Yu, Hanli Wang and Zhihua Wei. Mixed pooling for convolutional neural networks. 2014.

[9] Paula Fortuna. Automatic detection of hate speech in text: an overview of the topic and dataset annotation with hierarchical classes. 2017.

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

[11] Frederik Hogenboom. An overview of approaches to extract information from natural language corporate, n.d. URL https://www.frederikhogenboom.nl/work/posters/dir10-nlp.pdf.

[12] Mark Hoogendorn and Burkhardt Funk. *Machine Learning for the Quantified Self*. Springer International Publishing AG, 2018.

[13] Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. 2015.

[14] Hojjat Salehinejad Sharan Sankar Joseph Barfett, Errol Colak and Shahrokh Valaee. Recent advances in recurrent neural networks. 2018.

[15] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition.* 2017.

[16] Karpathy. Cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/, 2017. Accessed: 2018-07-12.

[17] Yoon Kim. Convolutional neural networks for sentence classification. 2014.

[18] Kevin Koidl. Loss functions in classification tasks. https://www.scss.tcd.ie/~koidlk/cs4062/Loss-Functions.pdf, 2013.

[19] Ela Kumar. *Natural Language Processing*. I.K. International Publishing House Pvt. Ltd., 2010.

[20] Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. 2014.

[21] Hang Li. Deep learning for natural language processing: Advantages and challenges. 2017.

[22] Elizabeth D. Liddy. *Natural Language Processing*. In Encyclopedia of Library and Information Science, 2001.

[23] Marc Moreno Lopez and Jugal Kalita. Deep learning for nlp. 2017.

[24] Shervin Malmasi and Marcos Zampieri. Detecting hate speech in social media. 2017.

[25] Amit Mandelbaum and Adi Shalev. Word embeddings and their use in sentence classification tasks. 2016.

[26] J. D. McCarey. Why you should use cross-entropy error instead of classication error or mean squared error for neural network classier training. https://jamesmccaffrey.wordpress.com/, 2013.

[27] Sean McElwee. The case for censoring hate speech. https://www.alternet.org/civil-liberties/case-censoring-hate-speech, 2013.

[28] John T. Nockleby. *Hate Speech*. In Encyclopedia of the American Constitution (Macmillan), 2nd edition, 2002.

[29] Francisco Javier Ordóñez and Daniel Roggen. *Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition*. Sensors, 16(1), 2016.

[30] Georgios K. Pitsilis, Heri Ramampiaro, and Helge Langseth. Detecting oensive language in tweets using deep learning. 2018.

[31] Preeti and BrahmaleenKaur Sidhu. Natural language processing. 2013.

[32] Packt Publishing. Sentence classification using cnns. https://www.datasciencecentral.com/profiles/blogs/sentence-classification-using-cnns, July 10, 2018.

[33] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. 2017.

[34] David Robinson, Ziqi Zhang, and Jonathan Tepper. Hate speech detection on twitter: Feature engineering v.s. feature selection. 2018.

[35] Anna Schmidt and Michael Wiegand. A survey on hate speech detection using natural language processing. 2017.

[36] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. 2009.

[37] Soujanya Poria Tom Youngy, Devamanyu Hazarikaz and Erik Cambria. Recent trends in deep learning based natural language processing. 2018.

[38] Tzu-Hsuan Tseng Tzu-Hsuan Yang and Chia-Ping Chen. System implementation for semeval-2017 task 4 subtask a based on interpolated deep neural networks. 2017.

[39] Grace Udoh-Oshin. Hate speech on the internet: Crime or free speech? 2017.

[40] V. Vapnik. *The Nature of Statistical Learning Theory*. NY: Springer-Verlag, 1995.

[41] John Berkowitz Zachary C. Lipton and Charles Elkan. A critical review of recurrent neural networks for sequence learning. 2015.

[42] Ziqi Zhang. Hate speech detection: A solved problem? the challenging case of long tail on twitter. 2018.

[43] Ziqi Zhang, David Robinson, and Jonathan Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. 2017.