Birkbeck

University of London

MSc Computer Science

Project Report:

Character-level Convolutional Neural Networks for Hate Speech Detection with Intelligent Adversaries

Author
Joe Worsfold

Supervisor
Dr. Taolue Chen

Tuesday 17th September, 2019

Abstract

As of January 2019 there were 3.48 billion global social media users per month [31]. With the rise of social media users, there has unfortunately also been a rise in the rate of hate speech distributed online. Despite attempts by platforms such as Facebook, Twitter, Reddit, and increasing pressure from Governments, the problem of detection and removal of hate speech has proven difficult to address.

In the past decade, various state-of-the-art detection models have been proposed, including a cutting-edge industry solution - Google Perspective. Nevertheless, recent research has shown these detection models are fragile when adversarial approaches are introduced to intentionally avoid detection. Therefore this paper will demonstrate the capability of Character-level Convolutional Neural Networks (CNNs) for hate speech detection. The CNN will be shown to retain a high level of detection accuracy, even for adversarial content, in part due to improved adversarial training.

Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
BOW	Bag-Of-Words
CNN	Convolutional Neural Network
CNTK	Microsoft Cognitive Toolkit
DRY	Don't Repeat Yourself
GRU	Gated Recurrent Unit Network
KISS	Keep It Simple, Stupid
LR	Logistic Regression
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PEP	Python Enhancement Proposal
POS	Position-Of-Speech
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RT	Retweet
SAGA	Stochastic Average Gradient Descent
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machine
TDD	Test-Driven Development
TF-IDF	Term Frequency–Inverse Document Frequency
WIP	Work-In-Progress

1 Introduction

Social media has irrevocably transformed the way people communicate and access information throughout their daily lives. Despite the tremendous positive impact of instantaneous communication helping to bring the world together, there has been an unmistakable proliferation of hate speech on social media platforms.

Unfortunately this malicious content has contributed to real-life consequences, for example, the Rohingya persecution in Myanmar has been exacerbated by the spread of hate speech and misinformation online. To the extent that an independent assessment into the impact of Facebook in Myanmar held that Facebook "weren't doing enough to help prevent [their] platform from being used to foment division and incite offline violence" [1].

The European Court Of Human Rights classifies 'hate speech' as any form of expression which aims to "spread, incite, promote or justify hatred based on intolerance." [1] Often this intolerance is directed towards the following protected characteristics: race or ethnicity, religion or beliefs, sexual orientation, disability, and/or transgender identity.

In this way, hate speech can be seen as separate from offensive content or content that has been classified as 'toxic'. Nevertheless, the regulation of both forms of content has become a focal point for social media platforms, researchers, and governments; for example in the United Kingdom new legislation has been announced with the aim to make UK citizens safer online [9].

1.1 Background

In order for hate speech to be regulated online it must first be detected, therefore hate speech detection can be specified as a classification problem. Text classification is one of the fundamental areas within the field of Natural Language Processing (NLP). It is widely used to classify or categorise unstructured textual data based on its content, for example, movie reviews could be classified as positive or negative. The goal in text classification problems is to create a classification function γ such that, given a set of labelled documents $\mathbb{D}$, an algorithm or learning method is created which can map a provided document $\mathbb{X}$ into a specified class or category $\mathbb{C} = \{C_1, C_2, ...C_j\}$ [18]. Therefore:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C}$$

Several classification functions which can classify unstructured textual data on social media as either 'hate speech' or 'ordinary speech' have been derived using existing machine learning algorithms. These machine learning models are predominantly trained via supervised learning [25] and in the past

[1] `https://www.echr.coe.int/Documents/FS_Hate_speech_ENG.pdf`

decade, algorithms such as Logistic Regression [7, 19, 29, 30] and Support Vector Machines [3, 7, 19, 28] have shown good results. More recently Deep Neural Networks [2, 8, 12, 22, 38] have also been devised to detect hate speech. In addition to using different machine learning algorithms there are a myriad of features that these algorithms can learn from. Schmidt and Wiegand [25] categorise these prior features into the following eight categories:

i Simple surface features

ii Word generalisation

iii Sentiment analysis

iv Lexical resources

v Linguistic features

vi Knowledge-based features

vii Meta-information

viii Multi-modal information

Despite the apparent complexity inherent in creating a model which accounts for all of the aforementioned features, in the only empirical evaluation comparing existing models it was found that simpler models performed comparably to more complex models. This paper by Gröndahl et al. [13] made a number of important conclusions, the most interesting of which was that existing models are incredibly brittle when confronted by intelligent adversaries.

The goal of an intelligent adversary is to trick a hate speech detection model into classifying hateful content as ordinary content, whilst still retaining the semantic meaning of the content. There are multiple variations of attack that will be discussed in more detail in Section 6; suffice to say, the attacks either severely hindered detection or pushed the accuracy of the models significantly off their baseline. Considering the nature of hateful content, and the desire of platforms and governments to regulate this type of content, it is plausible that users of social media would intentionally avoid detection via adversarial 'attacks'. Gröndahl et al. [13] concluded that future work on hate speech detection should focus on:

- Datasets instead of models: models did not perform well across different datasets due to labelling differences and various linguistic features.
- Character-models over word-based models: models trained on character-level features were more resilient to evasion methods which utilised simple text transformations.
- Adversarial training: data augmentation for example can help to reduce the effect of attacks on classification.

The majority of existing hate speech detection models make use of word-based features, including bag-of-words (BOW), word embeddings such as word2vec, and part-of-speech tagging (POS) [25]. In practise this means that these models focus on the words themselves to learn and distinguish between hate speech and ordinary speech; some words will be more indicative of hate speech than others. Nevertheless, user-generated content is often 'noisy' containing typos and slang, and adversarial approaches take advantage of the weakness of word-based models to handle these problems.

Character-level or character-based models have been implemented instead of or alongside word-based models in past papers. Wulczyn et al. [32] examine the use of Logistic Regression models using n-gram features on both words and characters, and the character-level models outperformed the word-based models. Zhang et al. [38] experiment with a common type of Deep Neural Network: Convolutional Neural Networks (CNN). In their paper, Zhang et al. achieve state-of-the-art competitive results in text classification problems with character-level convolutional neural networks (see Section 5).

1.2 Aims and Objectives

The aim of this project is to implement a character-level CNN capable of detecting hate speech on twitter, including when adversarial approaches are employed to evade detection, whilst retaining a high classification accuracy. In doing so, the project will highlight the importance of considering intelligent adversaries when devising the model architecture for related classification problems. The development process for the project can be summarised in the following stages:

1. Get, explore, and prepare the data
2. Train the models
3. Fine-tune the models
4. Present the solution

The report itself has been structured loosley on these stages and at each stage of the report there will be a critical analysis of the implementation, performance and overall effectiveness of the models. A greater emphasis will be placed on the second and third stage as these are the areas which will provide insight into the direction of future research. These insights will be summarised in the conclusion of the report alongside on-going challenges in the field of hate speech detection.

2 Project Synopsis

During this project a character-level CNN was trained and fine-tuned in order to classify hate speech, even in the presence of intelligent adversaries. In order to corroborate the classification accuracy of the CNN, additional models were trained including a Logistic Regression model. The implementation, performance, and analysis of intelligent adversaries and adversarial training is discussed in detail throughout this report.

To showcase the potential of these models, and to assist with the research, a prediction program was developed which could take a string or .csv file containing text and return a prediction as to whether the text contains hate speech. The program takes in four arguments: 1) -m or –model for the model to use for the predictions; 2) -s or –string used for the prediction; 3) -f –filepath used to provide a file for predictions; and 4) -w or –write used to provide a directory used to store the predictions. Multiple examples of the program running have been provided below:

```
$ predict.py -s 'i hate handicap faggots' -m logreg
  ['[1]', '[0.6968884]', 'i hate handicap faggots']

$ predict.py -s 'i hate handicap faggots' -m cnn
  ['[1]', '[0.7412937]', 'i hate handicap faggots']

$ predict.py -s 'i h4teHand1c4p f4ggots' -m cnn
  ['[0]', '[0.0038761]', 'i h4teHand1c4p f4ggots']

$ predict.py -s 'i h4teHand1c4p f4ggots' -m cnn+
  ['[1]', '[0.9941301]', 'i h4teHand1c4p f4ggots']
```

The program examples above return a list containing the classification: '1' for hate speech, '0' for non-hate speech; a probability score where 0.51 and above is indicative of a hate speech classification; and the original text that was tested.

The above examples not only showcase the prediction program, they also highlight the effect of intelligent adversaries and the performance of the adversarially trained CNN or CNN+ that was implemented for this project. A detailed analysis of all research undertaken during this project is available in this report.

3 Datasets

3.1 Sources

A large proportion of existing research into hate speech and abusive language detection has made use of privately collected datasets. Nevertheless, there is a selection of publicly available Twitter-based datasets. This project will exclusively use these Twitter-based datasets summarised in Table 1 to train and implement hate speech classification models. It is unlikely that these models will be able to generalise with regards to other types of data, however it will allow for a critical analysis of the effects of intelligent adversaries and adversarial training on hate speech classification.

Dataset	**Classes (tweets)**
DT [7]	hate speech (1430), offensive (19,190), ordinary (4163)
WZ-L [29]	racism (1934), sexism (3149), neither (11,010)
WZ-S.exp [30]	racism (85), sexism (777), both (35), neither (5697)
WZ-LS [38]	racism (2012), sexism (3769), both (30), neither (12,810)
WZ-LS*	racism (4), sexism (851), both (0), neither (2510)
DT-WZ	hate: hate speech (1430) ∪ racism (4) ∪ sexism (851) ∪ both (0) non-hate: ordinary (4163) ∪ offensive (19,190) ∪ neither (2510)

Table 1: Project datasets (union ∪ denotes conflation of class elements)

The largest of the publicly available datasets **DT** contains approximately 24,000 tweets. Created by Davidson et al., it has been used in many subsequent works. This corpus was collected using the Twitter API (see Section C) by searching for tweets containing words and phrases identified as hate speech and compiled by hatebase.org. A final sample of 25,000 was taken from a much larger corpus and was manually labeled by CrowdFlower workers. These workers were tasked with labeling each tweet - based on a provided set of instructions - with one of three classes: (1) hate speech, (2) offensive but not hate speech, and (3) ordinary speech. A minimum of three workers labeled every tweet, with a majority vote being used for the final classification. This resulted in a final sample size of 24,783 labeled tweets. Looking at Table 1 however, it can be seen that the dataset is imbalanced with only a very small proportion (5%) being coded as hate speech. See Subsection 3.2 for further analysis.

Similarly, Waseem et al. collected tweets by searching for frequently occurring terms linked to hate speech based on their own analysis. Over the course of multiple papers [29, 30] Waseem et al. collected and annotated 18,621 tweets. Annotations were carried out by a mixture of crowd-sourced amateurs and domain specific experts with the final label being chosen by a majority vote. Unlike Davidson et al. the tweets were labeled based on a specific class of hate speech: (1) racist, (2) sexist, (3) both racist and sexist, or (4) neither. Waseem et al. created a variety of datasets (see Table 1) but this project will utilise a single dataset **WZ-LS** containing all of the collected data compiled by Zhang et al. [38]. The final dataset contained only tweet IDs and class labels, not the tweet itself, which had to be retrieved based on the tweet ID using the Twitter API (see appendix C).

A noticable difference between the two datasets seen in Table 1 is the use of different classes. Davidson et al. [7] use **DT** to highlight the problem of erroneously misclassifying offensive language as hate speech, which could have potential legal ramifications in the real world. In order to combat the issue they trained their models to explicitly differentiate between offensive language and hate speech, but are only partially successful.

In comparison, Waseem et al. [29, 30] classify tweets based on specific characteristics - racism and sexism - which they use alongside meta-information - gender and location - to improve the accuracy of their classification. Nevertheless, they noted only minor improvement when utilising meta-information [25]. Therefore this project will employ a binary classifier in order to concentrate on the issue of intelligent adversaries. **DT-WZ** refers to the final dataset used throughout this project, created by combining the tweets from **DT** and **WZ-LS** as shown in Table 1.

3.2 Analysis

The greatest challenge posed by using **DT-WZ** is the imbalanced classes. This was an existing issue with **DT** but the problem would have been offset by the additional data from **WZ-LS**. However, at the time of accessing this data via the Twitter API a large proportion of the tweets labeled racist or sexist had been deleted or removed. The final version has therefore been classified as **WZ-LS*** and the loss of potential data can be seen in Table 1.

Unfortunately there were no other publicly available datasets to bolster the minority hate class. Despite the challenges this created, there are numerous methods for dealing with imbalanced datasets which will be explored when training the detection models in sections 4 and 5.

To collect the data for **DT** and **WZ-LS** both authors searched for terms which were considered indicative of hate speech. This scoped approach is common when creating datasets but can exclude examples that are more obscure. In addition, **DT** data is sourced mainly in the United States and contains hateful and offensive language from that culture.

In a recent paper [6] Davidson et al. have cautioned that their dataset contains substantial racial bias. **WZ-LS** was also shown to contain similar biases but it was less pronounced due to the influence of expert annotations. There is still a clear and present benefit in using these datasets for academic research but caution was advised if using these datasets for real-world detection and enforcement. For these reasons, any future work should focus on new datasets as highlighted by Gröndahl et al. [13]. Nevertheless, this project's goal is to highlight the importance of considering intelligent adversaries when choosing model architecture and this can be shown using **DT-WZ**.

4 Baseline Models

Many state-of-the-art models have been proposed for hate speech classification. Among them are what could be considered ‘classic methods’ which employ manual feature engineering and utilise algorithms such as Naive Bayes, Logistic Regression (LR) and Support Vector Machines (SVM). By far the most common ‘classic’ algorithms used for hate speech classification are SVMs [3, 7, 19, 28] and LR [7, 19, 29, 30]. In order to showcase the effects of intelligent adversaries and for comparison with CNNs, both a linear SVM[2] and LR[3] model has been implemented using scikit-learn. See appendix B: tools and techniques for further detail.

4.1 Preprocessing

To prepare the raw tweets for feature extraction they must first be preprocessed to remove noise and normalise their content. Noise removal includes:

- removing URLs;
- removing mentions (@username); and
- removing html entities (&entity_name; or &#entity_number;).

Normalisation includes:

- lowercasing;
- removing punctuation and numbers;
- removing stop words including twitter specific terms such as ‘RT’ for retweet;
- removing excessive whitespace characters; and
- stemming words to reduce word inflections using the Porter Stemmer.

Variations on the above mentioned preprocesses were tested and classification performance for both the SVM and LR model improved with their inclusion. The preprocessing was applied for both models via a reusable python class: ‘PreProcessUtils’ (see appendix D) and made use of common packages (see appendix B).

[2]`scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC`

[3]`scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression`

4.2 Features and Parameters

The main features extracted for both the SVM and LR were the unigram, bigram, and trigram features weighted by their Term Frequency–Inverse Document Frequency (TF-IDF). These features were then filtered to remove any which had a document frequency less than 5 as these sparse features are not informative. In addition, any features appearing in more than 75% of the documents were removed for the same reason. Similarly to Gröndahl et al. findings [13], there was no significant improvement in the performance of either model when using additional features outlined in Davidson et al. [7]: Part-of-Speech (POS) tagging, reading ease score, sentiment scores, and counts for hashtags, mentions, retweets, and URLs.

Both the SVM and LR models have parameters that can be tuned to improve their overall classification performance. In simple terms, the goal of these parameters is to further fine-tune the model to improve learning accuracy and efficiency, ensure the model is able to generalise based on the training data, and reduce over-fitting. Over-fitting in particular is a common problem for machine learning algorithms. It is caused when a model attempts to capture noise - data points that don't truly represent the classes - and subsequently reduces a model's ability to generalise as shown in Figure 1.

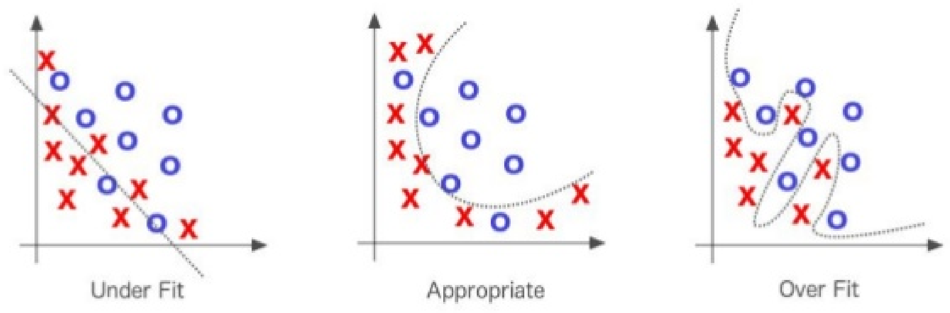


Figure 1: Under-fitting, over-fitting and appropriate-fitting displayed in graph form

Regularisation is used to avoid over-fitting by reducing the variance of a model, without substantially increasing its bias. The parameter C is an inverse regularisation parameter which acts to modify the strength of the regularisation. Although the implementation of the C parameter works differently for an SVM and LR model, its fundamental effects are the same. It helps to moderate the trade off between model simplicity and model accuracy based on the training data as shown in Figure 2.

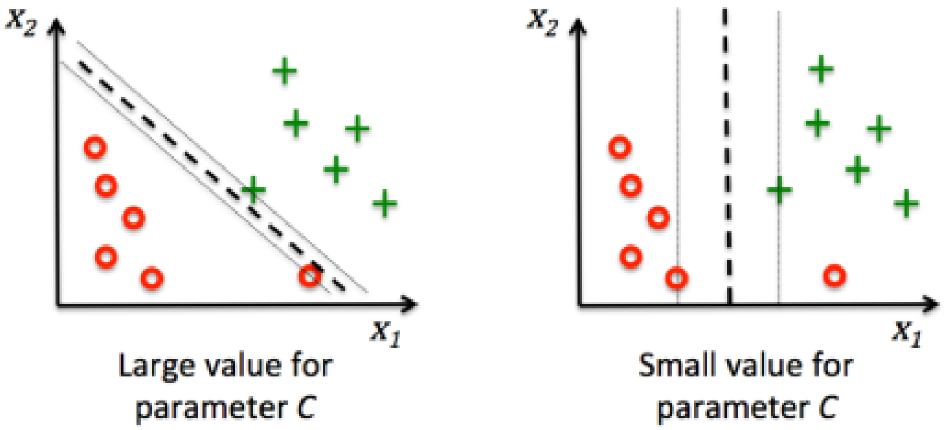


Figure 2: Impact of C on classification

Another way to control the regularisation of the model is to specify a norm used in the penalisation of the loss function. L1 regularisation or Lasso regression and L2 regularisation or Ridge Regression are two examples of the penalty parameter in both a linear SVM and LR model. Linear SVMs can also have a parameter set for its loss function such as 'hinge' or 'square_hinge', whereas the LR model can have a 'solver' algorithm specified for the optimisation problem, for example 'Stochastic Average Gradient Descent' (SAGA).

4.3 Date Resampling

As previously highlighted, the dataset **TD-WZ** has imbalanced classes. This is a common problem in machine learning and there are methods available to manage it. The three most common are oversampling, undersampling or generating synthetic samples. The latter involves utilising certain techniques such as the Synthetic Minority Oversampling Technique (SMOTE) which makes use of a nearest neighbour algorithm to generate new data based on the minority class.

This method was ruled out due to the complexities involved in the classification of hate speech and generated samples would likely suffer from the problem of misclassification as offensive language. The two former methods involve either oversampling the minority class or undersampling the majority classes. Both options were explored and oversampling provided the best results for both the SVM and LR models. See Section 4.5 for implementation details.

4.4 Model Validation

In machine learning, model validation refers to the process where a trained model is evaluated using a set of test data. Commonly when using large datasets the 'train test split' will be made using the hold-out method where the data is simply split, usually 80:20 training and test data respectively. An alternative is to use cross-validation or k-fold cross-validation which randomly splits the data into 'k' groups. Each group is then used to train and test until each group has been used as the test set. For example, Figure 3 shows a 5-fold cross-validation where training and testing takes place 5 separate times. Again, both options were examined and cross-validation was seen to give the best results. See Section 4.5 for implementation details.

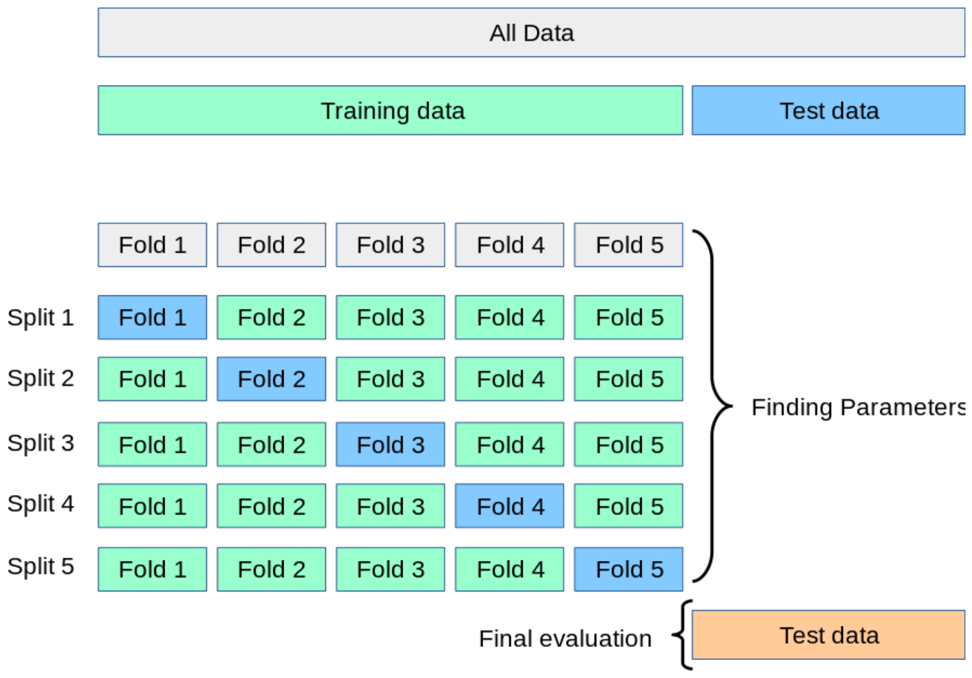


Figure 3: Above: an example of hold-out; below: an example of 5-fold cross-validation (source: scikit-learn cross-validation webpage)

4.5 Model Implementation

In order to evaluate every iteration of the SVM and LR model design and to finalise the implementation, the Precision (P), Recall (R), and F1-measures were used to compare results. Due the imbalanced data the models were fine-tuned to produce the best possible F1 score for the minority class: hate.

The final implementation of the **SVM** made use of the Linear SVM algorithm provided by scikit-learn. In order to find the best value for C a simple for-loop was used to increment through values of C and the best performance was gained by setting C to 0.01. And, although cross-validation was shown to exceed the hold-out method, the data was simply split 90:10 when training and testing this model. The reason for keeping the implementation of the SVM relatively simple will be highlighted in results Section 4.3.

Implementation of the LR model made use of more complex processes to ensure that the final model achieved the best possible F1 measure for the hate class. The data was again split into 90:10 but this time the 90% training data was used in a grid-search to pick the most suitable parameters for the penalty, C, and solver parameters using 5-fold cross-validation. The final parameters were penalty: L2, C: 0.01, and solver: SAGA.

4.6 Results

Overall classification accuracy is very high as seen in Table 2, but the Precision, Recall and F1-scores for the hate class are much lower. This is a common occurrence when dealing with imbalanced data. Nevertheless, the model's performance is slightly better than a similar model implemented in [7] which achieved only 0.44 and 0.61 for its hate classes Precision and Recall respectively.

Metric	Precision		Recall		F1-score	
Model	LR	SVM	LR	SVM	LR	SVM
Non-hate	0.97	0.97	0.95	0.96	0.96	0.96
Hate	0.53	0.61	0.65	0.62	0.58	0.61
Accuracy	–	–	–	–	0.92	0.93
Macro avg	0.75	0.79	0.80	0.79	0.77	0.79
Weighted avg	0.93	0.94	0.92	0.93	0.93	0.94

Table 2: Classification Report for the LR & SVM models based on their predictions over a test dataset

It can be seen in Table 3 that the SVM model is misclassifying hate speech as non-hate speech as often as non-hate speech is being misclassified as hate speech. Whereas, the LR model is misclassifying more non-hate speech as hate speech, as shown in Table 4. This is due to the grid-search selecting the best parameters for the LR model based on the Recall score for the hate class. In real-world environments any recalled hate speech would likely be further classified by a human, therefore recalling the highest amount of possible hate speech is preferable.

Although the overall performance is suitable, any future research will need to handle the inherent difficulty differentiating between hate speech and offensive language. Improving the quality of data or including a multi-class classifier would help to solve this issue in future.

		Predicted		
		Non-hate	Hate	Total
Actual	Non-hate	2488	93	2581
	Hate	90	144	234
	Total	2578	237	2815

Table 3: Confusion matrix for SVM model

		Predicted		
		Non-hate	Hate	Total
Actual	Non-hate	2444	137	2581
	Hate	81	153	234
	Total	2578	237	2815

Table 4: Confusion matrix for LR model

Despite the slightly better performance of the SVM on F1-scores, it is the LR model that will be used to make comparisons with the character-level CNN. This is in part to ensure a greater recall of hate speech but also due to the ability of the LR model to provide a probability prediction for its classification. This will make an empirical exploration into the effects of intelligent adversaries simpler and more rigorous.

5 Convolutional Neural Network

Convolutional Neural Networks (CNN) have achieved incredible results in the field of Computer Vision, but have also been shown to produce excellent results for Natural Language Processing (NLP) tasks. Past implementations of CNNs [2, 12, 38] for text classification have made use of word embeddings like word2vec: a process of mapping words to vectors of real numbers which are easier for a classification model to work with. A more recent paper from Zhang et al. [38] implemented a CNN+GRU (Gated Recurrent Unit Network) which showed promising results for hate speech classification. Nevertheless, the CNN+GRU was shown in [13] to be particularly vulnerable to intelligent adversaries as can be seen in Section 6. The most likely cause of this vulnerability is the model's reliance on word-based features.

Gröndahl et al. [13] summarise that when adversarial approaches are introduced, character-level models have an improved classification accuracy compared to word-based models. Knowing this and with the state-of-the-art performance of CNNs [38] in mind; this project aims to explore the performance of character-level CNNs for hate speech detection. Character-level CNNs have already been shown to produce interesting results in the wider area of text classification. In particular, LeCun. et al. [37] achieve a high degree of classification accuracy with their character-level CNN and it is this model that will act as the foundation of the model implemented by this project for hate speech detection.

5.1 Preprocessing

Initially the data for the CNN is preprocessed in the exact same way as shown in Section 4.1. However the preprocessing utilities (see appendix D) make it possible to adjust these preprocessing steps. These steps will be adjusted in Section 7 when implementing adversarial training to improve the model's accuracy.

5.2 Features and Parameters

Deep Neural Networks are arguably more complex than some of the 'classic methods' of machine learning highlighted in Section 4. In particular, there are many more features, parameters, and other factors to consider when creating a model. Rather than create a model architecture and features from scratch, this paper will replicate the model described by LeCun et al. [37] depicted in Figure 4.

CNNs have very unique features such as convolutions and pooling layers, but they also share common components with all neural networks. These include but are not limited to: an input layer, hidden layers, fully connected layers or output layers, weights and biases, activation functions, loss functions and model optimisers.

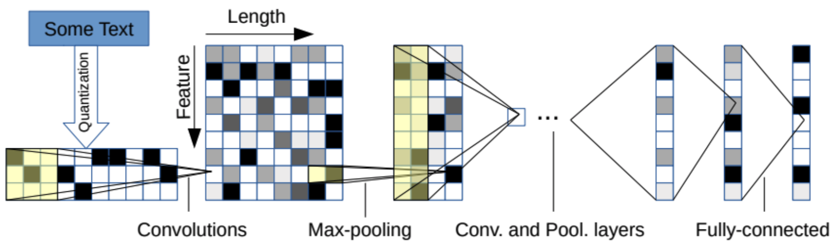


Figure 4: Depiction of LeCun et al. [37] CNN model architecture and implementation

The **input layer** for the CNN must be able to take in a tweet which has been converted into a tensor or 0/1 matrix. Therefore each tweet is broken down into a sequence of individual characters and each character is represented by a one-hot vector. This simple process, known as one-hot encoding, encodes every character in the tweet as an array of zeros with a one in the position of the represented character in a provided alphabet. Assuming the max length of all tweets is 140 characters and we're using the English alphabet of 26 characters, the input will take a tensor of dimension 26 by 140. If any tweets are less than 140 characters they are padded with all zero encodings.

The **hidden layers** will consist of a network of neurons or nodes and connections between the neurons. Consider a neuron:

$$Z : \sum(weight * input) + bias$$

Weight and biases are the trainable parameters that will adjust the value for Z, where the value for Z is used to decide whether a neuron should fire or not. The decision of whether to fire, or in other words activate, the neuron is made by an activation function which takes Z as an input. Two common activation functions used during the implementation of this CNN are the Sigmoid and ReLU, as shown in Figure 5.

In a CNN there are additional weights which make up a **kernel** or filter. These kernels 'slide' over the input data and perform a dot product between the input matrix and the kernel matrix. The sliding size known as the stride can also be parameterised but in most cases the kernel will slide over the entire input matrix. This layer in a CNN is known as a convolution layer. It can often be followed by a **pooling layer** which replaces the output of the network at specified points by instead computing a summary statistic of the nearby outputs. The purpose of pooling is to reduce the total amount of computation and weights that must be tuned when training the model. There are several functions available to perform this pooling step but the most common is max pooling.

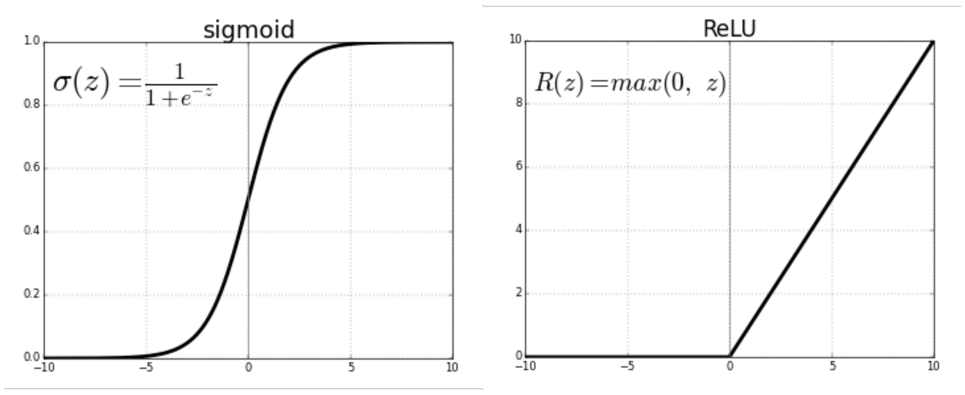


Figure 5: Sigmoid and ReLU activation function and graph

Similarly to the aforementioned 'classic methods', neural networks make use of **loss functions** to calculate the error value based on a comparison between the predicted value and the actual output value. The exact derivation of the error value with respect to the weights and biases is calculated to reduce the error rate. This process in neural networks is known as backpropagation and in simple terms it is how the network learns.

5.3 Data Resampling

Resampling of the imbalanced data was performed and subsequently analysed following the same process outlined in Section 4.3.

5.4 Model Validation

Validation was performed and subsequently analysed following the process outlined in Section 4.4. The only difference is that the data had to be split three ways for the CNN into train, validate, and test sets. First the data was split 90:10 into train and test, then the training data was split again 90:10 into train and validate.

5.5 Model Architecture and Implementation

The model architecture for the character-level CNN was implemented based on the architecture outlined in [37] and was adjusted to better reflect the Twitter dataset used for training. The CNN is 9 layers deep with 6 convolution layers as shown in Table 5 and 3 fully-connected layers as shown in Table 6.

The ReLU activation function is used for all of the CNN layers except the final fully connected layer which uses Sigmoid. This final layer consists of a single node which will contain a probability score between 0 and 1 based on the Sigmoid activation function. Binary cross entropy loss was used as the loss function and the optimiser used was Adam. In total the CNN consisted of 2,609,653 trainable parameters.

Layer	Frame	Kernel	Pool
1	256	7	3
2	256	7	3
3	256	3	N/A
4	256	3	N/A
5	256	3	N/A
6	256	3	3

Table 5: Input and hidden layers or convolutional layers

Layer	Output Units
7	1024
8	1024
9	1

Table 6: Fully connected layers and single node output

In order to deal with the previously highlighted issue of over-fitting there are two dropout models in between the three fully-connected layers with a dropout probability of 0.5. Dropout is a regularisation method which, during training, randomly ignores or ‘drops out’ a layer’s outputs. This has the effect of configuring and updating a layer differently during training to ensure the network is more robust and less reliant on specific nodes. The number of epochs can also have an impact on whether the model is being over-fitted based on the training data. After monitoring the model accuracy and model loss during training and validation it can be seen in Figure 6 that a low number of epochs is suitable to avoid over-fitting. The final epoch size was set at 6 and the batch size was set at 100.

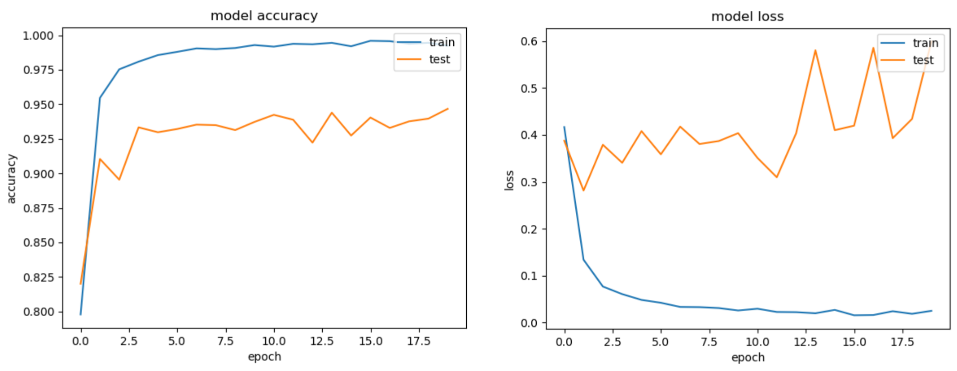


Figure 6: CNN model accuracy and model loss during training and testing

5.6 Results

Similarly to the baseline models the CNN classification accuracy overall was very high, as seen in Table 7. Nevertheless, the Precision, Recall and F1-scores for the hate class are actually lower than the baseline. The difference is relatively minor and is once again caused by the problem of misclassifying offensive language as hate speech.

The Precision is on par with the SVM model as seen in Table 2 but it has a lower Recall than both the LR and SVM model. This would have been improved by utilising a grid-search, like the one used for the LR model, where the best parameters could have been chosen based on their positive effect on the Recall. This and a multi-class classifier would be recommended for future research.

	Precision	Recall	F1-score
Non-hate	0.96	0.97	0.96
Hate	0.61	0.51	0.56
Accuracy	–	–	0.93
Macro avg	0.78	0.74	0.76
Weighted avg	0.93	0.93	0.93

Table 7: Classification Report for the CNN model based on its predictions over a test dataset

		Predicted		
		Non-hate	Hate	Total
Actual	Non-hate	2505	76	2581
	Hate	114	120	234
	Total	2578	237	2815

Table 8: Confusion matrix for CNN model

In lieu of not using the grid-search method a multitude of different parameters and fine-tuning was explored, but it can be seen in Table 8 that misclassification was still an issue. One interesting note is that the classification shown in all three confusion matrices is very similar; the same offensive language and hate speech may be being misclassified. This should be analysed in more detail in future research.

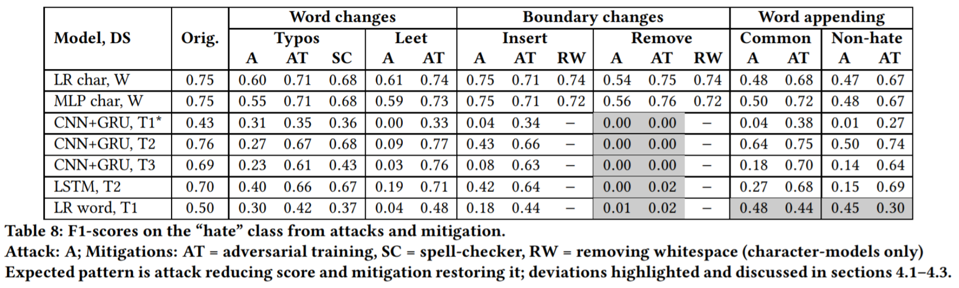
Although the CNN is slightly less accurate than the SVM and LR models, it will be shown to be much more robust when intelligent adversaries are introduced.

6 Intelligent Adversaries

As previously stated, the goal of an intelligent adversary is to trick a detection model into classifying its content as not hateful. Gröndahl et al. in their systematic study [13] looked at how susceptible existing models were to six common and easily implementable 'attacks':

- Word-character changes
  1. Inserting typos
  2. Inserting leetspeek
- Word-boundary changes
  1. Inserting whitespace
  2. Removing whitespace
- Word appending
  1. Appending common words
  2. Appending non-hateful words

These six attacks were performed on a total of seven different model-dataset combinations based on studies discussed throughout this report. The word-based models' performance was almost completely compromised and character-based models were pushed from their baseline prediction precision. The effects of these six kinds of attack can be seen in Figure 7 which shows the decreased detection accuracy of models, as well as their improved performance following adversarial training.



Model, DS	Orig.	Word changes					Boundary changes						Word appending			
		Typos			Leet		Insert			Remove			Common		Non-hate	
		A	AT	SC	A	AT	A	AT	RW	A	AT	RW	A	AT	A	AT
LR char, W	0.75	0.60	0.71	0.68	0.61	0.74	0.75	0.71	0.74	0.54	0.75	0.74	0.48	0.68	0.47	0.67
MLP char, W	0.75	0.55	0.71	0.68	0.59	0.73	0.75	0.71	0.72	0.56	0.76	0.72	0.50	0.72	0.48	0.67
CNN+GRU, T1*	0.43	0.31	0.35	0.36	0.00	0.33	0.04	0.34	–	0.00	0.00	–	0.04	0.38	0.01	0.27
CNN+GRU, T2	0.76	0.27	0.67	0.68	0.09	0.77	0.43	0.66	–	0.00	0.00	–	0.64	0.75	0.50	0.74
CNN+GRU, T3	0.69	0.23	0.61	0.43	0.03	0.76	0.08	0.63	–	0.00	0.00	–	0.18	0.70	0.14	0.64
LSTM, T2	0.70	0.40	0.66	0.67	0.19	0.71	0.42	0.64	–	0.00	0.02	–	0.27	0.68	0.15	0.69
LR word, T1	0.50	0.30	0.42	0.37	0.04	0.48	0.18	0.44	–	0.01	0.02	–	0.48	0.44	0.45	0.30

Table 8: F1-scores on the "hate" class from attacks and mitigation.
Attack: A; Mitigations: AT = adversarial training, SC = spell-checker, RW = removing whitespace (character-models only)
Expected pattern is attack reducing score and mitigation restoring it; deviations highlighted and discussed in sections 4.1–4.3.

Figure 7: Table 8 from Gröndahl et al. [13] showing the effectiveness of adversarial approaches on seven existing model-data-set combinations

The difference in performance between both word and character-based models is indicative of the importance of implementing new character-based models like the one proposed by this paper. To further analyse the impact of these 'attacks' the three most harmful: inserting leetspeak, removing whitespace, and appending non-hate terms, will be used to test the prediction accuracy of both the baseline LR model and the newly created CNN.

6.1 Attacks

In order to test the 'attacks' impact on prediction accuracy a sample dataset of hateful tweets was created. In July 2019 it was widely reported that actor Jessica Alba's Twitter account had been hacked and there was widespread condemnation of the racist, homophobic and sexist content that had been published. This method of sourcing existing samples is preferable to creating any data from scratch which would likely suffer from classification bias. The tweets which will be used for the construction of these attacks are outlined below (offensive words have been censored).

T1: I Hate Handicap Faggots
T2: ugh, police sirens in the distance again. When will niggers stop committing crimes so I can get some fucking sleep?
T3: Man I hate pakis. Fuck those magic flying carpet riders
T4: Nazi Germany Did Nothing Wrong And That's On God Nigga
T5: Listen up women, there is a reason why god made men stronger than you. It's because rape is a force of good, that should be celebrated. IF a man wants to fill you with his big cock, you should learn to : relax, orgasm and thank him!!! Heil Hitler #bussdown
T6: To all the little girls watching...never doubt that you are valuable and powerful & deserving of every chance & opportunity in the world.

Every tweet contains some elements of hateful speech except for **T6** written by Hilary Clinton which has been included as a control.

6.1.1 Leetspeak

Leetspeak, also known as leet or 1337, is a system of modifying the spelling of a word by replacing certain characters with similar looking numbers or symbols [23]. Commonly used on the internet, leetspeak would have an adverse effect on existing classification models that rely on word-based features or do not include numbers in character-based features. As shown below **T2** has been modified to intentionally obfuscate it using leetspeak and common hateful words have been specifically targeted.

T2: ugh, police sirens in the distance again. When will n1gg3rs stop committing cr1m3s so I can get some sleep?

6.1.2 Whitespace Removal

A simple and well-known form of writing without the use of whitespace is often utilised by programmers: Camel Case and Pascal Case. Similarly to leetspeak, removing the separators from a string of text makes it very difficult for a word-based model to understand the input. This has been showcased by removing whitespace around the most hateful terms in **T1** shown below.

T1: I HateHandicapFaggots

6.1.3 Word Appending

Finally, word appending was shown in [13] to have a significant effect on classification accuracy. They propose using a single word ‘love’ for its obvious counter meaning to ‘hate’ and as a single word will affect classification without changing the semantic meaning of the sentence. We experiment with adding ‘love’ to all tweets in the above mentioned sample.

6.2 Results

To showcase the effectiveness of the three specified attacks, and an amalgamation of all three attacks, they were performed on three classification models as shown in Table 9. The LR and CNN model implemented in this paper were included, as well as a state-of-the-art industry solution implemented by Google - Perspective. By including the Perspective API it can be seen that the attacks are effective against more well established models. Using Perspective isn’t a perfect comparison as it is designed to classify ’toxic’ content, but the effectiveness of the attacks is still noteworthy.

It must be noted that the LR model returns a probability for both non-hate and hate so it very inflexible and doesn’t go outside a range of about 0.40 to 0.61. Whereas, the CNN and the Perspective API give a single probability score that has a much more varied range. In all cases, anything above 0.50 can be considered a classification of hate speech.

A	Original			Leetspeak			Whitespace			Append			All attacks		
M	GP	LR	CNN	GP	LR	CNN	GP	LR	CNN	GP	LR	CNN	GP	LR	CNN
T1	0.92	0.61	0.74	0.38	0.45	0.01	0.14	0.45	0.85	0.90	0.59	0.70	0.08	0.43	0.01
T2	0.93	0.56	0.99	0.22	0.45	0.01	0.23	0.45	0.97	0.93	0.54	0.99	0.15	0.46	0.03
T3	0.97	0.53	0.89	0.18	0.45	0.00	0.09	0.46	0.00	0.97	0.51	0.66	0.08	0.44	0.00
T4	0.92	0.49	0.20	0.36	0.46	0.04	0.23	0.46	0.01	0.88	0.47	0.01	0.19	0.44	0.01
T5	0.95	0.55	0.01	0.47	0.45	0.01	0.24	0.48	0.00	0.95	0.54	0.01	0.18	0.46	0.00
T6	0.27	0.49	0.00	0.11	0.47	0.00	0.11	0.47	0.00	0.23	0.49	0.00	0.17	0.44	0.00

Table 9: Predicted probability scores per sample tweet (A: attack; M: model; Whitespace: whitespace removal; Append: appending non-hate terms; GP: Google Perspective)

There is a large amount of data to analyse in Table 9, but there are three main areas worth focusing on:

1. All three models were adversely affected by the adversarial examples, especially an amalgamation of all three.

2. The character-level CNN performed better than the other two models when it came to both whitespace removal and appending non-hateful terms.

3. Appending non-hateful terms was the least impactful adversarial attack despite reports to the contrary in [13].

Until this paper there had been no corroborating research to that undertaken by Gröndahl et al. in [13]. It is clear that adversarial attacks can have an adverse effect on word-based models, but the leetspeak insertion was also shown to affect the character-based CNN to the same degree as the other models. Unlike the conclusions in [13] the word appending had the least impact on classification across the models; further research will be required to understand why.

The character-level CNN did struggle, as did the LR model, to classify some of the sample tweets. In particular, **T4** and **T5** were misclassified by the CNN as the racism and sexism displayed was less common in the training dataset. This is why using scoped searches to find examples of hate speech in tweets can be unreliable. Nevertheless, the CNNs performance is promising and further research should be undertaken to find the best possible model architecture and parameter hyper-tuning for this problem area. In the meantime, adversarial training can be employed to improve the classification accuracy of the CNN when adversarial examples are in use.

7 Adversarial Training

Adversarial training can apply to any training methods used to resist adversarial examples in a given problem area. The importance of this kind of training has increased alongside the greater prevalence of intelligent adversaries. The most common form of adversarial training injects adversarial examples into the training data in an attempt to improve a model's ability to handle adversarial examples.

7.1 Data Augmentation

The training and testing data was supplemented with additional examples consisting of stochastically transformed versions of the existing data. This was implemented with the help of two additional preprocessing functions: one was responsible for creating leetspeak adjusted tweets and the other would append specific non-hate terms. These could both be adapted to work with different forms of leetspeak and could append words from a set of non-hateful terms. The initial implementation is relatively simple to showcase the possible benefits of augmenting the data in this way.

7.2 Model Improvements

Improvements were also made to the aforementioned CNN's implementation. Specifically, the preprocessing of the tweets was handled very differently. Instead of removing numbers, punctuation and stop words, only the whitespace characters were removed. This has the effect of improving the model's classification of tweets where whitespace has been removed and provides additional features by which to make the classifications. Given that adversarial approaches often create additional 'noise', it is beneficial to utilise a model that can interpret it and thus remain unaffected. The only other change to the model architecture and implementation is the use of an enlarged alphabet which includes numbers, punctuation and additional special characters.

7.3 Results

The improved model was retrained on the augmented data and the aforementioned attacks were performed again on this new model. The results of these attacks can be seen in Table 10 alongside the results from the attacks on the original CNN for comparison.

Unfortunately, like the previously implemented CNN, the advanced CNN had difficulty classifying some of the sample tweets. Therefore, Table 10 is looking at the most promising results from the adversarial training and testing.

A	Original		Leetspeak		Whitespace		Append		All attacks	
M	CNN	CNN+	CNN	CNN+	CNN	CNN+	CNN	CNN+	CNN	CNN+
T1	0.74	0.80	0.01	0.99	0.85	0.81	0.70	0.91	0.01	0.99

Table 10: Predicted probability scores for sample tweet 1 (A: attack; M: model; Whitespace: whitespace removal; Append: appending non-hate terms; CNN: original CNN implementation; CNN+: adversarially trained, advanced CNN implementation)

It can be seen that, by introducing adversarial training and adjusting the model implementation, the advanced CNN performed measurably better at classifying hate speech even in the presence of adversarial examples. In fact, the adversarially altered tweets received an even higher hate speech probability score than the original tweet. This is indicative of over-engineering when it comes to data augmentation, but the results are still promising.

More research is needed to improve the accuracy of both the original and advanced CNN, and more recent research into adversarial training should be examined. However the results make it clear that character-level CNNs are a powerful tool in the field of hate speech classification.

8 Summary and Conclusion

This project set out to showcase the potential benefits of using a character-level CNN for hate speech detection, whilst simultaneously highlighting the importance of making informed choices regarding model architecture in the early stages of a project. It has been shown just how problematic intelligent adversaries can be in the field of hate speech classification and it is recommended that any future research considers their impact from the outset. Adversarial training is a particularly powerful tool to ensure that adversarial 'attacks' are less effective, but the model architecture and features are just as important. This project has helped to confirm that character-based models are a suitable alternative to word-based models and perform much better in the presence of adversarial examples. The final model implemented in this project, the adversarially trained CNN, showed a great deal of potential but ultimately was not able to accurately classify more varied forms of hate speech outside of the training dataset.

Ultimately the biggest challenge faced throughout this project was the lack of available data for the hate class. This would have been less pronounced had the Waseem et al. datasets been available and the tweets themselves still been accessible via the Twitter API. Even if this data had been available, the models implemented during this project would have still struggled to classify more varied, nuanced forms of hate speech. The dataset authors made use of a scoped search, looking for tweets containing terms indicative of hate speech. This will undoubtedly cause certain types of hate speech to be easier to classify, whilst others are missed.

To avoid this issue, and the problem of classifying hate speech alongside offensive language, this project could have also focused on the detection of 'toxic' content. It has been seen that the Perspective API is just as susceptible to intelligent adversaries and there is likely more data available to assist with this task. In addition, LeCun et al. [37] highlight in their paper that text classifying CNNs are likely to require millions of data points rather than the smaller amount used for this project.

Despite the aforementioned challenges, the implementation of the project progressed smoothly. All models were trained and fine-tuned as outlined in the project proposal and a rigorous empirical evaluation of intelligent adversaries was carried out. The data was sourced and carefully analysed to understand the potential challenges, and multiple resampling techniques were attempted to solve the data imbalance. Throughout the project, there were multiple barriers, for example: bugs, deprecated code, missing data. Nonetheless, this project was a successful exploration of the possibilities in the field of machine learning and natural language processing.

If work were to continue on this project then the following updates are recommended:

- Source more hate speech data for the initial dataset
- Grid search and cross-validation should be used to improve the implementation of the CNN
- Fine-tune and explore more variety in the model architecture and hyper parameters
- Create a web API for easy access to the prediction models.

In summary, any future research in this field should focus on the following:

- Creating new datasets which are region specific to account for cultural variations
- Avoiding racial bias which has been shown in [6] to be a huge problem, for example simply putting 'black gay woman' into the Perspective API returns a toxicity score of 0.87.
- Implementing and comparing more character-level models including a simple character-based Logistic Regression model and Decision Tree model.

In doing so, models can make use of meta-information and knowledge-based features to ensure detection and classification is safe to employ in a real-world enterprise.

Appendices

A Development Process

As highlighted in the Project Proposal the development of this project followed Agile methodologies. In particular, all elements of the development were recorded in a product backlog and progressed through a set of stages from ‘Ready to Develop’ to ‘Complete’. A Kanban board was used to track the progress of each backlog item through the stages. Following Work-In-Progress (WIP) methods, work was done one item at a time to ensure focus and consistency. Rather than work to a specific timeframe, each backlog item’s complexity was estimated in a process similar to Planning Poker on a Fibonacci scale. Each week the Kanban board would then be reviewed and time dedicated to backlog items based on their estimates. This way the development was done on time and work was prioritised to avoid scope creep.

Development on the project was also carried out following popular development principles. Most importantly code was written, where possible, using Test-Driven Development (TDD). Tests were written in advance for a selection of functions and classes to ensure that they would fulfill their exact requirement. When the code was refactored and modified the tests ensured that no new bugs were introduced to the code. The Twitter API (see appendix C) and Preprocess Utilities class (see appendix D) had the largest number of tests written for them due to their complexity. In addition, code was written to follow two more popular principles: DRY (Don’t Repeat Yourself) and KISS (Keep It Simple, Stupid). Finally, the code was formatted based on the current best practise guidelines for Python: PEP8.

B Tools and Techniques

Python (3.6.9) was chosen as the programming language to develop this project based on a number of important factors.

First and foremost, the machine language libraries available in Python are some of the most professional and celebrated. Second, the standard library and other common libraries make scripting in Python powerful and robust. These include: RE for regular expressions, Pytest for unit testing, Numpy for scientific computing, Matplotlib for plotting graphs, and Pandas for data manipulation and analysis.

Thirdly, Python’s virtual environments (venv) make it possible to keep this project’s dependencies and installation separate from other projects. Fourthly, Python is adept at working with .csv files which were used to store the data. Due to the simple nature of the data, consisting of just two columns for the tweets and labels, it was not necessary to use a more complex database such as MySQL. Finally, PyCharm was especially good at managing all of the above mentioned libraries, packages and environments. Whilst there are plenty of other languages suitable for machine learning, for example R and

Matlab, Python was the better choice for this project.

The availability of machine learning libraries in Python make the choice of language easier, but the choice between the libraries was more challenging. One of the most versatile ML libraries in Python is scikit-learn or sklearn. Throughout the project this library was used for splitting data into training, validation and test sets, resampling data, utilising k-fold cross-validation, searching for the best model parameters, and evaluating the final performance of the models over the test data. It was also the library used to build the linear SVM and LR models respectively. The sheer utility and ease of use was unmatched by other similar libraries. The Natural Language Toolkit or NLTK library was also useful for certain preprocessing tasks such as stemming but was otherwise less versatile than sklearn.

Despite this, sklearn does not provide any libraries for Deep Neural Networks. Therefore the CNN implemented for this project was built utilising a high-level API called Keras. An advantage when using Keras is that it runs on top of state-of-the-art deep learning libraries such as TensorFlow and Theano. As suggested by the Keras documentation, this project made use of a TensorFlow backend. Keras made it possible to build the CNN at a higher level of abstraction, leaving more time to focus on hyper-tuning the model parameters. Other popular frameworks for deep neural networks include Pytorch and Microsoft's Cognitive Toolkit (CNTK).

C Twitter API

Twitter maintains a robust developer platform which enables developers to harness APIs, tools, and resources to build applications. Access to the APIs is provided via a developer account which can be created free upon request - assuming Twitter approves the specific use-case. Once access is granted, it is possible to generate the required credentials and take advantage of their REST APIs. The simplest way to leverage the Twitter API is via an existing Python library known as Tweepy. The code written exclusively for this project showcases how to use the Twitter API to access and store tweet statuses based on Twitter IDs. It was developed by applying TDD and all of the code and test cases are available upon request.

D Preprocessing Utilities

Data preprocessing is a very important step in any data science project when working with raw data such as text. Section 4.1 and 5.1 highlight some of the methods used to preprocess data prior to their use in their respective models. To avoid repeating code and ensure consistency between models all preprocessing steps were implemented in a single Python class. Having all of the methods in a single class also made it easier to develop using TDD. The code for the class and all of the test cases are available upon request.

References

[1] Alex Warofka, Product Policy Manager. An Independent Assessment of the Human Rights Impact of Facebook in Myanmar. `https://newsroom.fb.com/news/2018/11/myanmar-hria/`, (2018). [Online; accessed 15/09/2019].

[2] Badjatiya, P., Gupta, S., Gupta, M., and Varma, V. Deep learning for hate speech detection in tweets. *CoRR abs/1706.00188* (2017).

[3] Burnap, P., and Williams, M. L. Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy & Internet 7*, 2 (2015), 223–242.

[4] Chen, Y., Zhou, Y., Zhu, S., and Xu, H. Detecting offensive language in social media to protect adolescent online safety. In *Proceedings - 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust and 2012 ASE/IEEE International Conference on Social Computing, SocialCom/PASSAT 2012* (12 2012), Proceedings - 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust and 2012 ASE/IEEE International Conference on Social Computing, SocialCom/PASSAT 2012, pp. 71–80.

[5] Dadvar, M., Trieschnigg, D., Ordelman, R., and de Jong, F. Improving cyberbullying detection with user context. In *Proceedings of the 35th European Conference on Advances in Information Retrieval* (Berlin, Heidelberg, 2013), ECIR'13, Springer-Verlag, pp. 693–696.

[6] Davidson, T., Bhattacharya, D., and Weber, I. Racial bias in hate speech and abusive language detection datasets. *CoRR abs/1905.12516* (2019).

[7] Davidson, T., Warmsley, D., Macy, M., and Weber, I. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media* (2017), ICWSM '17, pp. 512–515.

[8] Del Vigna12, F., Cimino23, A., Dell'Orletta, F., Petrocchi, M., and Tesconi, M. Hate me, hate me not: Hate speech detection on facebook.

[9] Department for Digital, Culture, Media & Sport. New laws to make social media safer. `https://www.gov.uk/government/news/new-laws-to-make-social-media-safer`, (2018). [Online; accessed 16/03/2019].

[10] Dinakar, K., Jones, B., Havasi, C., Lieberman, H., and Picard, R. Common sense reasoning for detection, prevention, and mitigation of cyberbullying. *ACM Trans. Interact. Intell. Syst. 2*, 3 (Sept. 2012), 18:1–18:30.

[11] Erraqabi, A., Baratin, A., Bengio, Y., and Lacoste-Julien, S. A3T: adversarially augmented adversarial training. *CoRR abs/1801.04055* (2018).

[12] Gambäck, B., and Sikdar, U. K. Using convolutional neural networks to classify hate-speech. In *Proceedings of the first workshop on abusive language online* (2017), pp. 85–90.

[13] Gröndahl, T., Pajola, L., Juuti, M., Conti, M., and Asokan, N. All you need is "love": Evading hate-speech detection. *CoRR abs/1808.09115* (2018).

[14] Hosseini, H., Kannan, S., Zhang, B., and Poovendran, R. Deceiving google's perspective API built for detecting toxic comments. *CoRR abs/1702.08138* (2017).

[15] Johnson, R., and Zhang, T. Effective use of word order for text categorization with convolutional neural networks. *CoRR abs/1412.1058* (2014).

[16] Kim, Y. Convolutional neural networks for sentence classification. *CoRR abs/1408.5882* (2014).

[17] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[18] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, (2008).

[19] Mehdad, Y., and Tetreault, J. Do characters abuse more than words? In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue* (2016), Association for Computational Linguistics, pp. 299–303.

[20] Nguyen, T. H., and Grishman, R. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing* (2015), pp. 39–48.

[21] Noever, D. Machine learning suites for online toxicity detection. *CoRR abs/1810.01869* (2018).

[22] Park, J. H., and Fung, P. One-step and two-step classification for abusive language detection on twitter. *arXiv preprint arXiv:1706.01206* (2017).

[23] Perea, M., Duñabeitia, J. A., and Carreiras, M. R34d1ng w0rd5 w1th numb3r5. *Journal of Experimental Psychology: Human Perception and Performance 34*, 1 (2008), 237.

[24] Santos, C. D., and Zadrozny, B. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)* (2014), pp. 1818–1826.

[25] Schmidt, A., and Wiegand, M. A survey on hate speech detection using natural language processing. In *10.18653/v1/W17-1101* (2017).

[26] Sun, Y., Lin, L., Tang, D., Yang, N., Ji, Z., and Wang, X. Modeling mention, context and entity with neural networks for entity disambiguation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[27] Wang, P., Xu, J., Xu, B., Liu, C., Zhang, H., Wang, F., and Hao, H. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)* (2015), vol. 2, pp. 352–357.

[28] Warner, W., and Hirschberg, J. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media* (Stroudsburg, PA, USA, 2012), LSM '12, Association for Computational Linguistics, pp. 19–26.

[29] Waseem, Z. Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In *Proceedings of the first workshop on NLP and computational social science* (2016), pp. 138–142.

[30] Waseem, Z., and Hovy, D. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *SRW@HLT-NAACL* (2016).

[31] We Are Social. Kemp, S. DIGITAL 2019: GLOBAL INTERNET USE ACCELERATES. `https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates`, (2019). [Online; accessed 16/03/2019].

[32] Wulczyn, E., Thain, N., and Dixon, L. Ex machina: Personal attacks seen at scale. *CoRR abs/1610.08914* (2016).

[33] Xiang, G., Fan, B., Wang, L., Hong, J., and Rose, C. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2012), CIKM '12, ACM, pp. 1980–1984.

[34] Yin, D., Xue, Z., Hong, L., Davison, B., Edwards, A., and Edwards, L. Detection of harassment on web 2.0. *Proceedings of the Content Analysis in the WEB 2.0 (CAW2.0) Workshop at WWW2009* (01 2009).

[35] Zeng, D., Liu, K., Lai, S., Zhou, G., and Zhao, J. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers* (Dublin, Ireland, Aug. 2014), Dublin City University and Association for Computational Linguistics, pp. 2335–2344.

[36] Zhang, X., and LeCun, Y. Text understanding from scratch. *arXiv preprint arXiv:1502.01710* (2015).

[37] Zhang, X., Zhao, J. J., and LeCun, Y. Character-level convolutional networks for text classification. *CoRR abs/1509.01626* (2015).

[38] Zhang, Z., Robinson, D. C., and Tepper, J. A. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *ESWC* (2018).