Detection of Cyberbullying Through BERT and Weighted Ensemble of Classifiers

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

30-12-2021 / 05-01-2022

CITATION

Graney-Ward, Christopher; Issac, Biju; KETSBAIA, LIDA; Jacob, Seibu Mary (2022): Detection of Cyberbullying Through BERT and Weighted Ensemble of Classifiers. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.17705009.v1>

DOI

[10.36227/techrxiv.17705009.v1](https://doi.org/10.36227/techrxiv.17705009.v1)

Detection of Cyberbullying Through BERT and Weighted Ensemble of Classifiers

Christopher Graney-Ward, Biju Issac, Lida Ketsbaia and Seibu Mary Jacob

Abstract—Due to the recent popularity and growth of social media platforms such as Facebook and Twitter, cyberbullying is becoming more and more prevalent. The current research on cyberbullying and the NLP techniques being used to classify this kind of online behaviour was initially studied. This paper discusses the experimentation with combined Twitter datasets by Maryland and Cornell universities using different classification approaches like classical machine learning, RNN, CNN, and pre-trained transformer-based classifiers. A state of the art (SOTA) solution was achieved by optimising BERTweet on a Onecycle policy with a Decoupled weight decay optimiser (AdamW), improving the previous F1-score by up to 8.4%, resulting in 64.8% macro F1. Particle Swarm Optimisation was later used to optimise the ensemble model. The ensemble developed from the optimised BERTweet model and a collection of models with varying data representations, outperformed the standalone BERTweet model by 0.53% resulting in 65.33% macro F1 for TweetEval dataset and by 0.55% for combined datasets, resulting in 68.1% macro F1.

Index Terms—Hate speech, NLP, Transformers, Machine learning, Deep Learning

I. INTRODUCTION

Due to the advancement of social media platforms, large quantities of data are now present for users to acquire, and thus social media has become a platform for ideas to be shared, exchanged and analyzed in order to extract valuable information about a variety of subjects. One of the most popular social mediums that can be found today is Twitter, a micro-blogging site which enables users to write up to 280 characters of text commonly referred to as tweets. Twitter messages contain basic terminology such as the use of emoticons that acts as a pictorial representation of facial expressions through the use of the punctuation, users use the “@” symbol to refer to other users on the platform thus automatically being alerted if they have been mentioned in such a fashion, special symbols such as “RT” is used in order to repeat and further comment on someone else’s earlier tweet. Lastly hashtags play a significant role within the microblogging site as users use “#” to mark topics making their tweets visible to a greater audience [1].

Cyberbullying is the use of the Internet, particularly social media to exclude, stalk, trick, humiliate, impersonate, harass, intimidate, or otherwise cause some form of psychological harm [14]. Smith et al, [27] defined cyberbullying as ‘an aggressive, intentional act carried out by a group or individual, using electronic forms of contact, repeatedly and over time against a victim who cannot easily defend him or herself’. Throughout the evolution of the Web, as discussed before, cyberbullying has immensely increased yearly. It is an increasingly important and serious social problem, which can

negatively affect individuals. Studies conducted on cyberbullying have concluded it is one of the top reasons teens fall into depression as it affects their everyday lives, it affects people’s perception of life and has led to people being twice as likely to self-harm or attempt suicide. In contrast to traditional bullying, cyberbullying is not limited to a place and time, thus making cyberbullying a more prevailing issue than traditional bullying. A concern that many often have is that victims do not necessarily identify their experiences as bullying thus leading to many not reporting the instances or seek the required help for their emotional distress. Various studies have supported these statements.

Cyberbullying detection is widely regarded to be a binary classification problem, however there is large variation in classification methods. Some researchers have opted for a bag-of-words or lexicon-based approach, but it should be noted that this method lends itself strongly to false positives. It should be considered that aggression is often misclassified as bullying or hate speech, notably in the case of profanities being used as terms of affection [33]. This is dependent on the dataset, however, including non-bullying, aggressive samples labelled as false in training can negate this but equally may lead to overbearing specificity. Sequential analysis of the text corpus is a good option since it takes the order of words/characters into account, therefore producing a form of contextual understanding, this is shown well in bi-LSTM solutions. Transformers are also a promising prospect for a complex task that will most likely require bidirectionality such as this one [25].

This paper is organised as follows. Section 2 includes the literature review of current methods for cyberbullying problems and some potentially novel approaches to the problem. Section 3 is on the datasets used and the preprocessing techniques. Section 4 is on the methodology that covers the details of the choice of metrics, model and the novel AI approaches applied. Section 5 is the design and implementation of the software. Section 6 is the results section with the settings/hyperparameters and their subsequent results for classical machine learning, CNN, RNN and transformers models. Particle Swarm Optimisation (PSO) was applied on the ensemble model to better the result. The dataset wordcloud and a link to a working product are introduced. Section 7 is the conclusion and future work.

II. LITERATURE REVIEW

Most studies have identified that cyberbullying detection is not just a simple bag-of-words text classification problem and

in some cases researchers have approached the problem using semantic based solutions.

A. Classical Machine Learning

Zhao et al. [43] used SVM along with word embeddings and bag of words + semantic + bullying features combination referred to as EBoW. This method claims to have garnered an f1 score of 78%. Unfortunately, the dataset used consists only of tweet id's, many of which are now inaccessible, meaning comparison is not possible. Zhao and Mao have reported the use of an embedding-enhanced bag-of-words approach for improving textual cyberbullying detection [10] and have suggested the use of participant-vocabulary consistency for detecting cyberbullying. Other efforts have focused on the use of complementary information to enhance text-based cyberbullying detection [42]. Raisi et al. proposed a model that simultaneously discovers instigators and victims of bullying as well as new bullying vocabulary by starting with a corpus of social interactions and a seed dictionary of bullying indicators. They formulate an objective function based on participant-vocabulary consistency [32].

Menger, Scheepers and Spruit [36] presented an improved model using user-based features, i.e., the history of the user's activities and demographic features. Huang, Singh and Atrey [29] focused on social network features for cyberbullying content and provided improved performance in cyberbullying detection by considering online relationships. Another significant research conducted on sentiment analysis using Twitter data is by Barbosa and Feng [4] who used a two-phased approach to Twitter sentiment analysis. The two phases were 1) classifying the dataset into objective and subjective classes, identified as subjectivity classification, and 2) classifying subjective sentences into negative and positive classes which is described as polarity detection. Gebremeskel described that Barbosa and Feng [4] were "suspecting that the use of n-grams for Twitter sentiment analysis might not be a good strategy since Twitter messages are short, they use two other features of tweets: meta information about tweets and syntax of tweets. For meta-info, they use POS tags (some tags are likely to show sentiment, eg. adjectives and interjections) and mapping words to prior subjectivity (strong and weak), and prior polarity (negative, positive and neutral). The prior polarity is reversed when a negative expression precedes the word. For tweet syntax features, they use (hashtag, @reply), RT(retweet), link, punctuations, emoticons, capitalised words, etc." They went on to create a feature set from both the features and used machine learning techniques which are available on WEKA. In conclusion they found that SVM outperforms all classifiers. For the test data, 1000 tweets were manually annotated as positive, negative, and neutral [29].

B. Neural Networks

It has been found that using a CNN with character embedding yielded the best results in a 2016 study [8]. Similarly, using the same dataset, a study by Wikipedia found that word or character level n-grams yielded better results than previously explored methods [40].

Character embedding treats each character as a vector instead of each word. This provides improvements in some cases as the number of potential vectors is decreased from potentially limitless to a maximum of 3560 (including special characters, capitals and lower case and all Unicode emojis), this can go as low as 26 vectors if the text is made into lower case and special characters are ignored. There are 171,476 header words (vectors) in the English dictionary and this does not include misspelled words which are likely to be a common occurrence on social media. Character level embedding also handles misspelling and slang better than word-level embedding as it does not create a new numeric value for a misspelled/out of vocabulary (OOV) word, this will improve the robustness of models outside controlled conditions [18].

Character embedding yielded better results than word embedding on one study [40], but recent advances in contextual embedding techniques using multiple bi-directional LSTM layers look to be a promising way to deal with highly nuanced language. A study into sarcasm detection using contextual embedding outperformed other state-of-the-art methods [24] using Bi-directional LSTMs. A 2017 study also found that LSTM to be effective in sarcasm detection [6]. The proven success of this method in sarcasm detection is promising for bullying detection.

The choice of loss function and optimiser play an important role in NN development. Binary crossentropy loss with Adam optimiser is the logical choice for a binary classification problem like this. The possibilities of using AdamW [20] should be explored. AdamW improves the generalizability of Adam by adding a decoupled weight decay. Implementing a onecycle training policy [34] is another fairly new technique that can seek some extra performance out of a given model by throttling learning rate to prevent the fitting process from finding a local optimum rather than the global optimum.

C. Transformers (Pretrained General Language Models)

As with many text classification problems these days, building a model from scratch is often not the best option, especially for smaller datasets. State of the art (SOTA) pretrained models could potentially set a solid basis for solving this problem as they already have high quality architectures which have already been trained on huge amounts of high quality, unlabelled data. The transformers python library [39] includes many SOTA pretrained models such as Google's BERT [12], XLNet [41], RoBERTa [19], GPT2 [31], FastText [17] et al. These pretrained models are excellent in the fact that they are very generalizable as they can be trained on downstream text tasks from categorisation to generation to summarization. XLNet for example held SOTA status in 18 different benchmark NLP problems at time of release. A fun analogy for applying a pretrained model on a downstream task compared to training a new model would be that the pretrained model is an adult and the from-scratch model is a child. Children learn new skills faster than adults, however adults have a much richer base understanding which can provide a contextual background for future learning.

Fig. 1. The BERT input representation

BERT [12] is a generalizable-pretrained NLP model developed by Google in response to OpenAi’s (OAI) ground-breaking GPT 1.0 [30]. At the time of release, BERT [12] held SOTA status on no less than 11 common NLP tasks. It improved on OAI’s GPT by employing bi-directionality through the use of masked language model (LM) for backwards learning and Next Sentence Prediction for forwards learning. It differentiated from other models in that all parameters are sent down to initialise end-task model parameters. As a factor of BERT’s [12] age and significance in the 2018 NLP landscape, there are many easy to use tutorials, implementations and wrappers available which make using it especially easy for research and development. It also benefits from being a GPU friendly size, so can be fine-tuned on relatively cheap consumer hardware as of 2021. The BERT input representation is shown in figure-1 [13]. The input embeddings are the sum of the token embeddings, i.e., the segmentation embeddings and the position embeddings.

XLNet is the successor to BERT, it outperforms BERT considerably in over 20 different NLP tasks from classification to text generation [41]. These performance gains were achieved by scrapping the masked LM backwards language modelling objective for backwards learning with an improved permutation LM inspired by advances in AR. The reason for masked LM’s comparative shortfall came down to an independence assumption of the masked vectors. The other major improvement was the implementation of the Transformer-XL language modeller as well as other minor changes. Unfortunately, XLNet is highly resource intensive, and training on a GPU can be taxing as it is generally designed with TPU use in mind.

BERT [12] also has a few variants including RoBERTa [19] by Facebook which improved on some design choices the original BERT team decided to overlook and achieved SOTA results in GLUE, RACE and SQuAD, outperforming or matching both BERT and XLNet. Another BERT variant is BERTweet [26], trained entirely on twitter data, BERTweet is designed to excel in twitter specific NLP tasks and was inspired by earlier domain specific equivalents such as SciBERT and BioBERT, to name a few. BERTweet is the first public large-scale language model pre-trained for English Tweets. BERTweet is trained based on the RoBERTa pre-training procedure, using the same model configuration as BERT-base. The corpus used to pre-train BERTweet consists of 850M English Tweets (16B word tokens $\sim$ 80 GB), containing 845M Tweets streamed from 01/2012 to 08/2019 and 5M Tweets related to the COVID-19 pandemic. BERTweet does better than its competitors RoBERTa-base and XLM-R-base and out-

performs previous state-of-the-art models on three downstream Tweet NLP tasks of Part-of-speech tagging, Named entity recognition and Text classification [25]. BERTweet has the advantage of being trained on everyday language found on social media rather than Wikipedia pages (like BERT, XLNet and RoBERTa) which in theory should make any kind of Tweet classification task perform better. BERTweet is also based on RoBERTa which on paper is the most performant of the three base architectures mentioned.

D. Ensemble approach

Combining different high performing classifiers into an ensemble proved to be an effective performance boost in a recent study [35]. The researchers found that an ensemble of models using different data representations balanced the strengths and weaknesses of each data representation. For example, they had ML models using TF and TFIDF as well as a BERT nomination for the Transformers representation, and ULMFIT representing multi-layer LSTM architectures.

III. DATASETS

There are a variety of publicly available and academic access-on-request datasets for text aggression, harassment, bullying, hate speech, trolling and so forth. The aim of this model is to combine these datasets to allow for the best generalizability in the hopes of real world application. Two of the first results to come up in a regular search are the DataTurks and MeanBirds datasets (<https://www.kaggle.com/daturks/dataset-for-detection-of-cybertrolls> and <https://zenodo.org/record/1184178>). The dataturks dataset is a dump with no model or paper officially associated, though it looked very promising. However upon manual review, it was clear that many of the entries were mislabelled. For example, the tweet “*man that rly sucks. I for 1 am positive that all will work out. You’re a bright dude... pretty cool 2 if I don’t say so myself.*” had been mislabelled as aggressive, likely due to the presence of the word “sucks” – others attempting to solve this problem should be weary of the poor quality of this dataset as it is one of the first google results when searching for cyberbullying datasets. There is also a dataset available through chat coder (<https://www.chatcoder.com/drupal/DataDownload>), which follows threads of chats, making it unsuitable for binary classification. It was also relatively small and required lots of pre-processing due to being in a chained XML format made up of hundreds of files. It should be noted that conversation history would greatly improve the contextual understanding of a model, however this greatly increases the complexity of the model, reduces the number of datasets that can be used and makes the required data format for a classification much more limited in scope.

A dataset by University of Maryland [15] is ideal to achieve the aims of this paper with accurate labels, substantial data, and no obvious duplicates. This dataset was selected for research and development in this paper. This is the only dataset that is not publicly available and access is granted on a case-by-case basis. The problem with swear words and offensive text being manually misclassified as hate-speech or bullying is

Fig. 2. Class frequency of the MMHS150K dataset after applying mode.

talked about in length in a study by Cornell University [11]. The researchers asked annotators to separate tweets into “*hate speech*”, “*offensive*” and “*neither*” to subvert the problems introduced by manual annotation and over reliance on semantic lexicons such as HateBase. The researchers suggested that future work should aim to correct some of the biases in the dataset itself. The dataset is of high quality and was selected for use in this research work. The other dataset evaluated by this paper is also by Cornell – named MMHS150K [16]. It comprises of 150,000 tweets, each manually labelled three times by independent reviewers into one of 6 categories: Racial, Homophobic, Sexist, Religious, Other and NonHate. Upon review, the data seems properly labelled – no instances of profanities being labelled as hate speech.

Potentially offensive but otherwise innocuous tweets such as those containing the slang word *ni*guh/ni*ga* used as a term of affection for example are also correctly labelled as non-hate. This dataset can easily be processed into a binary class dataset by replacing all hate labels with True and ‘non hate’ with False, then taking the mode of the three labels. Despite being large, is still not enough on its own to differentiate the different types of hate text, likely due to noisy data and the general required background understanding of race, sex, religion and politics that backs opinion and identity. This being said, the MMHS150K is three times larger than Maryland and Cornell datasets combined. The chosen three datasets (one Maryland and two Cornell datasets) were combined to form one dataset [15], [11], [16].

A. Sanitisation (one Maryland and two Cornell datasets)

Since the research focuses on Twitter messages the requirement of pre-processing became largely necessary as tweets were not formatted in the required way needed as it contains vast amounts of spelling errors. Through the research [21], it was found that ‘text based communication in English seems to frequently ignore the rules of grammar and spelling’. This makes it necessary for pre-processing to be done to produce a cleaner dataset, thus increasing the performance of classification that are later used significantly. Whilst researching all literature regarding twitter analysis the pre-processing technique was largely integrated by altering words, removing symbols and punctuation as well as correcting simple spelling errors. One study suggests that tweets containing links should be excluded from the dataset as it can be assumed that part of the bullying lies within an image [5].

An option which can be explored in development is the standardisation of @usernames with all the datasets being twitter based. The benefit of replacing @ExampleUsername with <USER> in the pre-processing stage would be the reduction of noise, vectors and complexity. Most models will benefit from this as an individual username is almost always noisy data, but knowing that the tweet is referring to another user is highly useful information, especially in this domain. The same thinking can be applied to URLs. However this will likely have a much better performance impact on word vectorised solutions as character vectorised solutions are likely to learn what hate-speech associated domains look like and detect hate-related strings in the route. It is then important that the same regex (regular expression) replacements are applied to testing, validation and prediction data. In many cases, converting all train and test strings to lower case can help to reduce the number of potential vectors in a model. This avoids model confusion caused by misspellings and differences in capitalisation.

As proposed by Thelwall et al. [21], it may help to correct common spelling and grammatical mistakes. Finally, improvements in performance should be tested for when removing URLs from the data or entirely ignoring tweets which contain URLs under the assumption that the context lies within the image [5]. Of the selected datasets, some are binary classification, and others are multi class. To rectify this, all datasets will have to be formatted into binary classification. For example, the MMHS150K dataset contains the individual labels for each kind of hate (racism, sexism, etc.), to make this dataset binary, all the hate labels will be combined.

B. Benchmarking (TweetEval dataset)

In the context of hate speech detection, to get a good picture of how well the final model performs against other models, a standardised benchmark test comparable to the GLUE or SentEval task sets (for hate speech detection) is a good option. General Language Understanding Evaluation (GLUE) benchmark, is a collection of tools for evaluating the performance of models across a diverse set of existing natural language understanding (NLU) tasks [37]. SentEval is a toolkit for evaluating the quality of universal sentence representations and encompasses a variety of tasks, including binary and multi-class classification, natural language inference and sentence similarity [9].

There are various well known benchmarks in the NLP space, however for the domain of hate speech as a classification problem, the tests are less well known. TweetEval consists of seven heterogeneous tasks in Twitter, all framed as multi-class tweet classification. All tasks have been unified into the same benchmark, with each dataset presented in the same format and with fixed training, validation and test splits. The TweetEval standard [3] provides a variety of datasets for the evaluation of LMs against a variety of classification problems specific to social media (twitter). Of particular interest to this paper is the ‘hate’ dataset which has a test set of 2790 items, a validation set of 1000 items and a training set of 9000. Various transformers models have been trained downstream on the data in this paper and the results can be seen in table I.

TABLE I
TWEETVAL LEADERBOARD

Model	Macro F1
SVM	36.7%
FastText	50.6%
Bi-LSTM	52.6%
RoBERTa base	46.6%
RoBERTa with downstream Twitter training	49.9%
RoBERTa trained entirely on Twitter data	52.3%
BERTweet	56.4%
BERTweet (optimised in this paper)	64.8%

IV. METHODOLOGY

In order to achieve the best real-world results possible, the AI/ML model combines the data from [15], [11] and [16], i.e., one Maryland and two Cornell datasets. The combined dataset has a 3:1 imbalance of non-hateful to hateful tweets respectively as some datasets were augmented from multiclass to binary. The dataset from [11] is one such example and when deciding how to make this dataset binary, the choices were either to not count offensive language, count offensive language as hate speech or count offensive language as normal. The main issue that many research papers point to is the false positives that models return when text is aggressive or offensive but not bullying or hate speech. It was concluded that this should be avoided so that the decision was made to count offensive language as normal language. This will have the effect of overall reduced accuracy on the test data as the most obvious way to detect bullying and hate speech is by detecting swear n-grams. Any models trained on this data will suffer if they use simple ngrams and BoW (Bag of Words) techniques, but better performance should be achieved in real world situations. For this reason, accuracy and AUC were not used as metrics in testing. On the other hand, class weighted accuracy was used where possible and macro F1 score was used as the main metric, along with recall and precision as in equations (1) to (3), for model comparison.

$$precision_{macro} = \sum_{classes} \frac{precision\ of\ class}{number\ of\ classes} \quad (1)$$

$$recall_{macro} = \sum_{classes} \frac{recall\ of\ class}{number\ of\ classes} \quad (2)$$

$$F1_{macro} = 2 \cdot \frac{recall_{macro} \cdot precision_{macro}}{recall_{macro} + precision_{macro}} \quad (3)$$

Username in the combined dataset were all replaced with "@user" and URLs with "http". The ML models were created using Sklearn [28] for Python and optimised with a combination of grid search and manual tuning. The neural network solutions were developed using Tensorflow [2]. The transformers solutions were developed using ktrain [22], a wrapper for Keras [7] and huggingface transformers [39] that allow for easy implementation of the onecycle training policy [34] and weight decay optimization via AdamW [20]. Ktrain also has features to allow models to explain their predictions as can be seen in Figure 3. All the tools or libraries were tweaked manually to get a high local optimum.

Fig. 3. A transformers model making a prediction and explaining its decision making process.

A. Ensemble

When the BERTweet model was perfected, it was included in a roster with a variety of classical machine learning models, each with a count vectorised representation and a TFIDF vectorised representation. Predictions were made for all models. Next, an objective function is defined which takes a list of numerical weights between 0 and 1 for each model as a parameter. The objective function then executes the operations represented in the design stage for combining the weights in order to generate an array of predictions. These predictions are then used to generate a macro F1 score. The score is inverted so that the PSO (Particle Swarm Optimisation) algorithm will store the best configuration.

B. RoBERTa / BERTweet

BERTweet is the transformer model that is used in this paper. It has the same design and architecture as RoBERTa, however it is pre-trained entirely on twitter data as opposed to Wikipedia pages. RoBERTa improves on BERT with a plethora of minor improvements, all of which are overlooked by the original BERT team and these minor improvements add up quickly to create a model that is much better trained. BERTweet is trained on 850 million tweets, totalling 80 GB of uncompressed text and is the first general language model for twitter data. BERTweet is still new having only been published in the latter half of 2020 [25].

C. Onecycle Policy

The onecycle policy [34] is a method that helps models to find better minima (local optimum) and avoid overfitting by throttling the learning rate until the loss explodes, then returning to the base learning rate. This is done on a cycle by cycle basis, hence the name. This achieves the following. If a model has saddled in a local minima, having a fixed learning rate means that minima will be the final resting place of the model. Since transformers are trained with very small learning rates, being able to escape a potentially poor local minima and find a better one, or even find the global minima is an important step that should not be missed. The onecycle policy could be viewed as dynamic, automatic hyperparameter optimisation.

D. Decoupled Weight Decay

The application of a decoupled weight decay, like the one proposed in [20] prevents overfitting, and avoids the exploding gradient problem. De-regularisation means the regularisation

Fig. 4. Loss graphs comparing a standard Adam optimiser with a decoupled weight decay modified version of Adam, referred to as AdamW [20]

is independent from the optimisation steps taken by the loss function, leading to better overall performance than a weight decay coupled with L2 regularisation or no weight decay at all. This performance boost can be seen in figure 4 where the error and loss of AdamW is consistently lower than the loss of regular Adam. AdamW is also very easy to implement with ktrain, requiring only one line of code.

E. PSO (Particle Swarm Optimisation)

The Particle Swarm Optimisation (PSO) algorithm was used in this work to search for the optimal weights used to combine classifiers, which is a stochastic population-based algorithm. A form of swarm intelligence that uses a high quantity of particles to iterate several times exploring a space is used to compute the most optimal pathway. PSO makes use of additional parameters such as velocity rates of particles in addition to social aspects of particles. The particles in PSO are categorised in two groups of global best (gbest) and personal best (pbest).

The global best is the value which is retrieved from a particle in the search space which is then returned as the most optimal value for a given fitness function. Then there is the personal best which is closely related to the fitness function by tracking the best fitness that it has achieved by itself. Both values work closely with velocity and particles in order to iterate a given number of times and return an optimal value.

The study of PSO takes different variables of particles into account which then determine the shortest path to travel to reach a specific destination. This task is achieved by the particles iterating in each work-space and co-operating to plan out the shortest possible path. PSO is available in Python through the PySwarms library [23]. The social aspects of particles consist of total information being shared and level of co-operation between particles. The PSO algorithm is shown in figure 5.

For a binary PSO optimization technique, the position of the particles is expressed in two terms: 1 or 0 (or on and off).

Fig. 5. The PSO algorithm

If we have a particle x on d -dimensions, then its position can be defined as follows [25]:

$$x = [x_1, x_2, x_3, \dots, x_d] \quad \text{where } x_i \in 0, 1 \quad (4)$$

where we are given a data set with d features. We can assign each feature as a dimension of a particle. Hence, once we have implemented binary PSO and obtained the best position, we can then interpret the binary array simply as turning a feature on and off [25].

As in [35], the main concept of PSO is a particle involving two components: a position vector which refers to a potential solution (candidate) for the optimisation problem and a velocity vector. A set of positions were initialised as follows: $w_i^{(0)} (i = 1, \dots, nPoP)$ where $nPoP$ is the number of candidates in each generation and velocities $v_i^{(0)}$ randomly distributed throughout the design space bounded by specified limits.

Normally, the $w_i^{(0)}$ and $v_i^{(0)}$ are generated from the uniform distribution as $w_i^{(0)} \sim U[b_l, b_u]$, $v_i^{(0)} \sim U[-|b_u - b_l|, |b_u - b_l|]$.

The best known position $p_i^{(0)}$ is initialised by $w_i^{(0)}$ and the swarm's best-known position $p_g^{(0)}$ was updated based on the comparison between the value of the objective functions $f(p_i^{(0)})$ and $f(p_g^{(0)})$.

At the t^{th} iteration ($1 \leq t \leq maxT$), we update the position of each particle using its previous position and its updated velocity vector as in equations below:

$$w_i^{(t)} = w_i^{(t-1)} + v_i^{(t)} \quad (5)$$

$$v_i^{(t)} = av_i^{(t)} + C_1 r_1 (p_g^{(t-1)} - w_i^{(t-1)}) + C_2 r_2 (p_g^{(t-1)} - w_i^{(t-1)}) \quad (6)$$

Fig. 6. A saddle point between two maxima [38]

in which a is inertia weight used to set up the balance between the abilities of global and local search in PSO, C_1 and C_2 are the social and cognitive attraction indicating how much confidence it has in the swarm or in itself respectively, and r_1 and r_2 are two random number drawn from a uniform distribution over $[0, 1]$.

As in [35], the equations 7 and 8 will be used in the PSO objective function to generate a macro F1 score from the collective predictions vs the true predictions. Since PSO is only set up to find minima and not maxima, the F1 score will have to be inverted ($1/\text{F1 score}$) before the value is returned. The PSO algorithm was initially unsuccessful in procuring an improvement in score. However, by increasing the number of iterations and number of particles both to 10000 from 100, and reducing the minstep from $1e-8$ to $1e-4$, an absolute score increase of 0.53% was achieved.

Once all models have been tested and fine tuned, the best performing models will be selected to be included in a custom ensemble method based on the one described in Uphadhyay et al. (2020) [35] where Particle Swarm Optimisation (PSO) will be used to find optimal weightings to be applied to the prediction probabilities for each classifier. The hope here is that different data representations will iron out the weaknesses of the best model when it is unsure on a prediction. For that reason, the top ML classifiers will be represented with both count vectorised and Term Frequency Inverse Document Frequency (TF-IDF) vectorised nominations. The script will have to either train or load models that have already been trained, so it is important to ensure that all models are using the same train test split. The mathematical representation for combining the weights is shown in equations (7) and (8).

$$\text{combined_weights} = \sum \{ \text{probabilities}_i \times \text{weights}_i : i \in Z \text{ and } 0 \leq i < \text{num_models} \} \quad (7)$$

$$\text{collective_predictions} = \{ \text{classes}_{\text{argmax}(i)} : i \in \text{combined_weights} \} \quad (8)$$

F. Convergence and Complexity

While the Transformers based models performed the best, machine learning solutions were not far behind. The linear SVM most notably performed very well and had by far the

TABLE II
CONVERGENCE TIMES VS. PERFORMANCE

Model	Time to converge	Performance
XLNet	120 minutes	Medium
Linear SVM	1 minute	Medium
SVM	Did not converge	Low (on smaller dataset)
BERTweet	30 minutes	Best
RoBERTa	30 minutes	High

shortest training time, only taking a few seconds to a few minutes. BERTweet and RoBERTa did outperform the linear SVM by large margins, however they took 30 minutes to train on a GTX 1080. XLNet-large took the longest to train taking 120 minutes on the same hardware, and did not perform as well as RoBERTa and BERTweet, but matched the Linear SVM. The regular SVM failed to converge on the combined datasets ([11], [15], [16]) having being left to train for 2 days with no result. Please refer to Table II for details.

If it is assumed that the data is composed of M samples and N features, the general complexity of the PSO algorithm is $O(N_{\text{param}} \cdot N_{\text{pop}})$ where N_{param} is the number of parameters to be optimized N_{pop} is the population size, i.e. the number of swarm particles or potential solutions that we start with. The complexity of the potential ML classification model can be estimated as $O(M \cdot N)$. In the discussion below, d is the representation dimension or embedding dimension of a word (range: 128–512), n is the sequence length (range: 40–70) and k is the kernel size of the convolution. The transformer uses the self-attention mechanism where attention weights are calculated using all the words in the input sequence at once, hence it facilitates parallelization. In addition to that, since the per-layer operations in the Transformer are among words of the same sequence, the complexity does not exceed $O(n^2 d)$. For CNN it would be $O(k \cdot n \cdot d^2)$ and for RNN it would be $O(n \cdot d^2)$. So the total complexity for the PSO-based Transformer and ML models can be: $O(N_{\text{param}} \cdot N_{\text{pop}}) + n * O(M \cdot N) + O(n \cdot d^2)$, where n is the number of ML models.

V. DESIGN AND IMPLEMENTATION

To properly demonstrate the capabilities of the software developed, an API and accompanying website will allow users to interface directly with the AI, without having to download the entire model (0.5 GB) locally. In some cases where users have small amounts of RAM, the model might simply be too big to load into the memory. The API will be a Flask-RESTful based HTTP server, managed by Gunicorn and reverse proxied by an NGINX webserver. Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. Gunicorn 'Green Unicorn' is a Python WSGI (Web Server Gateway Interface) HTTP Server for UNIX.

A. Microservices

The entire system is encapsulated in a docker-compose environment which manages everything from deployment to package management, environment variables and cross platform compatibility. Docker was chosen because it is an industry best

Fig. 7. Class diagram for the API.

practice for cross platform tech stack standardisation via the deployment of microservices. Docker achieves this by running services within mini virtual machines - henceforth referred to as images. Docker is often used in full stack solutions when a webserver, backend, frontend and DB are required.

B. Webserver

Nginx was chosen as the webserver because it is the industry standard, beating Apache in terms of adoption and performance. Nginx also has a docker image which allows it to be set up and ran automatically as a microservice. Nginx is often used in industry as a reverse proxy or load balancer as it can accept a high number of simultaneous connections which it then passes to the slower upstream server (in this case, gunicorn + flask) in a manageable way.

C. Backend

Flask was chosen because it is incredibly lightweight and meets the requirements of the project. It also has extensions that allow Flask to become RESTful. FastAPI was also looked into and it does appear to be a technically superior solution, supporting asynchronous requests by default. It is however, newer and therefore has fewer plugins and educational resources. The API follows a class based structure, where each class represents an endpoint and methods are represented by functions. This means that endpoints can share code via a parent class as seen in the class diagram in figure 7. `HateTwitterHashtagPrediction` and `HateTwitterAccountPrediction` both inherit the abstract class `HateTwitterPrediction` which includes the logic for making Twitter API requests and in turn inherits the `HatePrediction` class which includes the logic for making AI predictions.

D. User Interface

The User Interface (UI) is a React web application. It is a single page application that communicates with the API using http(s) requests. React was chosen because of its ease of use and readability. React's state system also means that dynamic changes to the site such as automatic text classification when the user is typing, can be made into succinct code modules. React was chosen because it is an industry standard, developed by Facebook and adopted by various tech giants all over the world. The React code will be compiled into static HTML, CSS, and JS which can then be served from either the NGINX webserver used in the API, or a Linux, Apache, MySQL, PHP (LAMP) hosting provider.

From a human computer interaction (HCI) standpoint, the site is responsive, meaning it will dynamically adapt layout based on screen size so that the web application is not unusable on any device. This was made easy through the utilisation of Bootstrap for React, which provides a variety of pre-made React components and accompanying CSS styles. The React application contains three views in total. The home-page contains the basic prediction, wordcloud and explained prediction operation. The API page documents the individual endpoints including the request and response formats, as well as examples. Finally, the twitter demo page allows users to request predictions based on Tweets from either a hashtag or user. In terms of additional components, there is a header applied to every page for navigation purposes and a footer applied to every page for contact details and site information.

E. The Hate Detection Model

The model will be either a classical machine learning model, a CNN, an RNN, transformers model, or ensemble of model, depending on test results. To train the transformers models, it will be important to keep test logs and model checkpoints as training a transformers model on large amounts of data will take a substantial amount of time. After each epoch, the metrics for the entire test set should be taken and stored in a log file with a parent directory structure representing settings used to create the model. This could be achieved by writing a custom Tensorflow callback class called `validate` which is shown in figure 8. When a model is finished training, there should be a way to easily compare it to previous models. Equally, if the freshly trained model does turn out to be the new best, it is important that the model is automatically saved at the end of training so that it does not have to be re-trained, wasting valuable compute time.

In figure 8, the structure of the various classes and scripts demonstrates how they interact. The `CbadAi` class is a super class for the various types of AI model such as RNN, machine learning and transformers as there would be a lot of repeated code otherwise. It loads the chosen data using the instantiated `StandardiseDataSets` object, then formats it properly depending on the type of model and the specific expected input format. In figure 8, only the transformer variant was included as it is the only one to use the `validate` class. The transformer variant is a script rather than a class so unlike the CNN version for example, it imports and instantiates `CbadAi` rather than extending

Fig. 8. Class diagram for the model trainer.

it. The CbadAiTransformer script produces a predictor object which is saved to a local disk. The test_TweetEval script loads the saved predictor and instantiates a StandardiseDataSets object to get the tweeteval test set. It then spits out the results for that model against the test set. The StandardiseDataSets class performs the same preprocessing on all data it returns. It can return individual datasets or all of them combined, which helps to assist in peer reviewing results.

VI. RESULTS

After exhaustive experimentation with as many methods as described earlier, the results show a clear advantage to using transformers over more traditional methods. Results can be seen for machine learning in table III, for neural networks in table IV and for transformers in table V. Both word and character vectorizations were attempted, however, in all cases, word vectorization performed significantly better. The best performance was achieved by BERTweet trained on a onecycle policy with the settings found in Table VI. To test the learning power of this model, it was trained and tested on TweetEval data alone. Surprisingly, the modifications made to the model when using a combination of datasets, transferred well to the TweetEval dataset. As mentioned before, TweetEval consists of seven heterogenous tasks in Twitter, all framed as multi-class tweet classification. All tasks have been unified into the same benchmark, with each dataset presented in the same format and with fixed training, validation and test splits.

The model achieved a macro-F1 of as much as 64.8%, putting it up to 8.4% higher than the previous SOTA (which was also BERTweet). The variance in results suggests that the TweetEval dataset should be larger to improve reliability. The full results can be found in table VII. This demonstrates the power of weight decay and the onecycle policy. It should also be noted that the 99th percentile of tokenised string length in the TweetEval train set is 49, the 95th percentile is 42 and the mean is 21.

The ensemble developed from the optimised BERTweet model and a collection of models with varying data repre-

TABLE III
TOP PERFORMING BASE ML CLASSIFIERS

Model	Parameters	Macro-F1
Linear SVM	C=0.1 loss=hinge use_idf=True class_weight=balanced	65.3%
Decision Tree	splitter=random class_weight=balanced	59.5%
Rocchio	KNeighboursTransformer mode=connectivity leaf_size=0.0001 metric=city_block	64.1%
Passive Aggressive	use_idf=True C=0.1 average=True class_weight=balanced shuffle=False warm_start=True	63.2%

TABLE IV
TOP PERFORMING NEURAL NETWORK CLASSIFIERS

Model	Parameters	Macro-F1
CNN (word embed)	epoch=10 embedding_dim=10 vocab_size=30000 sequence_length=26 weight_decay=0.01 learning_rate=1e-4 Structure: Input () Embedding () Dropout (0.5) Conv1D (6, 2, relu) MaxPooling1D (2) Conv1D (3, 2, relu) MaxPooling1D (11) Dense (1, sigmoid)	61.1%
RNN (Bi-LSTM)	Epochs=10 vocab_size=30000 max_sequence_length=26 embedding_dim=10 weight_decay=0.01 learning_rate=1e-4 Structure: Embedding () Dropout (0.2) Bidirectional-LSTM() Dense (sigmoid)	63.2%

sentations, outperformed the standalone BERTweet model on TweetEval dataset by 0.53% macro F1 (i.e., 65.33%). On the three combined dataset, the increase was by 0.55%, resulting in 68.1% in macro F1. The weighting configuration for the various models can be seen in table VIII. Interestingly, despite the linear SVM being the best performing ML classifier on its own, the PSO optimisation gave it a weight of 0 (among other models), which effectively selected those models out of the final ensemble model.

When adapting our model for TWEETEVAL, changes were attempted to improve performance by increasing the max_length from 26 to 45, and this resulted in a serious drop in performance, however it can be assumed that a number near to the mean rather than the max is optimal on this specific dataset. Another thing to take note, is that preprocessing input/test labels (note: not training labels) so that they are lowercase, surprisingly, improved F1 score and in-practice performance.

TABLE V
TOP PERFORMING TRANSFORMERS

Model	Parameters	Macro-F1
BERT-base	epochs=2 learning_rate=5e-5	64%
DistilBERT-base	same as above	65%
roBERTa-base	learning_rate=3e-5	67%
DistilroBERTa-base	learning_rate=5e-5	65%
XLNet-base	learning_rate=1e-5	65%
BERTweet-base	weight_decay=0.1 learning_rate=3e-5	67.6%
BERTweet-base (WEC using PSO)	learning_rate=3e-5	68.1%
All of the above	batch_size=100	

TABLE VI
BERTWEET SETTINGS

lr	wd	batch size	max len	epochs	momentum
3e-5	0.1	100	26	2	0.85-0.95

When the BERTweet model was perfected, it was included in a roster with a variety of classical machine learning models, each with a count vectorised representation and a TFIDF vectorised representation. Predictions were made for all models. Next, an objective function is defined which takes a list of numerical weights between 0 and 1 for each model as a parameter. The objective function then executes the operations represented in the design stage by equations 7 and 8 shown earlier, in order to generate an array of predictions. These predictions are then used to generate a macro F1 score. The score is inverted so that the PSO algorithm will store the best configuration. The PSO algorithm was initially unsuccessful in procuring an improvement in score. However, by increasing the number of iterations and number of particles both to 10000 from 100, and reducing the minstep from $1e-8$ to $1e-4$ an absolute F1 score increase of 0.53% was achieved resulting in 65.33% macro F1 for TweetEval dataset and by 0.55% for combined datasets, resulting in 68.1% macro F1.

A. Word Cloud

A word cloud is a clear visual representation of words. It is used to show popular words and phrases based on their frequency and relevance, which can provide us with quick and simple visual insights. The word cloud generated from the combined datasets used in this paper can be seen in figure 9. Please be aware that some readers may find the language offensive (though the image is swirled). The word cloud generator is also available for public use via the API. The image is generated using the wordcloud Python library, with a maximum of 1000 words, a width of 3000px, a height of 2000px and standard English stop words.

TABLE VII
RESULTS ON THE TWEETEVAL BENCHMARK

	F1	Recall	Precision
Macro	64.8%	67.8%	69.2%
Weighted	64.4%	65.0%	71.3%
Binary	67.4%	85.9%	55.5%

Fig. 9. The hate word distribution within the datasets (swirled version)

B. Live Website demonstration

The API was created using Flask-restful, an extension of Flask that enforces best practices when designing a RESTful API. Flask was initially chosen because it is lightweight and is a Python package, making it easy to integrate with existing AI model and code. Code is reused efficiently thanks to the class based structure of Flask-restful. The Flask API is managed by Gunicorn, a production ready Web Server Gateway Interface (WSGI) server, which promotes effective scaling in a production environment. NGINX was selected as the web server for the system because it is currently the industry standard. All these elements were combined through the use of docker compose, this allows any user on any operating system to run the code as intended, without any setup troubles.

The interface was created using React, a client side, single-page application framework. React is highly modular which makes the code highly readable and intuitive by default. The interface features three pages. One page is a static API documentation page which explains how to use the API via http requests. On the homepage there is a typewriter graphic in the text field which gives helpful tips for interacting with the AI. This effect was achieved by starting an asynchronous function after the homepage component mounts. The function is an endless while loop which cycles through an array of hard-coded strings, further iterating each character in the string before adding the string to a React state variable which updates the text entry placeholder, resulting in the typewriter effect. Text inputted in the field is predicted on every second while the user is typing. This was implemented by creating an auto-updater function, which contains an infinite 'while' loop which checks for changes in the text corpus every two seconds. If a change is found, a new classification request is sent. The user may also generate explained predictions and word clouds from the text corpus, this calls the relevant endpoint which returns either an SVG image or raw HTML which is then injected into a HTML div. The final page is a Twitter demonstration page where one can enter a hashtag or username, the server will collect the latest posts from the said user or hashtag and make hate speech predictions on each of them.

The built UI is hosted on a LAMP server separate to the rest of the system, this decision was made simply because the LAMP server was already being used and paid for with a different site and had a significant amount of reserve capacity

Fig. 10. The hate detection website screenshot

TABLE VIII
ENSEMBLE WEIGHTING CONFIGURATION

Vectorisation	Model	Weighting
FTIDF	Passive Aggressive	0.00150191
TFIDF	Bagging	0.00648598
Count	Decision Tree	0.14094584
Count	Extra Tree	0.00203921
Count	Naive Bayes compliment	0.69857189
Sentencepiece	BERTweet	1

and free SSL certificates. The API is privately hosted on a dedicated server with 64GB RAM, Ryzen 5 3600 CPU, 1TB NVMe SSD, 1GB/s bandwidth.

A working online software interface of the final hate speech detection model is shown in figure 10 and can be accessed at the weblink: <https://cbad.xyz>. For example, if the words 'I hate myself' or 'I hate Jewish food' is entered into the web-based text box, it will be shown as 'non-hate'. But if it is replaced with 'I hate Jews', it will be shown as 'hate', showing that the software understands the context of hate speech. The figure 11 shows the user interface to search Tweets for hate speech based on username or hashtag.

VII. CONCLUSION AND FUTURE WORK

This paper discusses cyberbullying detection with Twitter datasets using different classification and NLP approaches. For

Fig. 11. The tweet hate detection website screenshot on search using user name or hashtag

TweetEval, it demonstrated to beat the previous SOTA by up to 8.6% as seen in Table I. It was found that base versions of machine learning and neural networks are not as efficient as transformers on the problem of cyberbullying or hate speech detection for results, but can still compete for complexity and size. A working prototype based on BERTweet was produced by combining the multiple datasets with plans of being used for real world application. The ensemble developed from the optimised BERTweet model and a collection of models with PSO with varying data representations, outperformed the standalone BERTweet model on multiple datasets. In future, we want to work to improve the detection accuracy even better through other forms of optimisations to the transformer models.

APPENDIX

GitHub implementation codes are available for this work. GitHub for AI code and data, GitHub for the API and GitHub for the UI, will all be made available in the final version of the paper, so that others can evaluate our work.

REFERENCES

- [1] A. Agarwal, B. Xie, I.V.O.R., Passonneau, R.: Sentiment analysis of twitter data. ACL Anthology pp. 30–38 (2011)
- [2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), <https://www.tensorflow.org/>, software available from tensorflow.org
- [3] Barbieri, F., Camacho-Collados, J., Neves, L., Espinosa-Anke, L.: Tweeteval: Unified benchmark and comparative evaluation for tweet classification, arXiv preprint arXiv:2010.12421 (2020)
- [4] Barbosa, L., Feng, J.: Robust sentiment detection on twitter from biased and noisy data. In: Proceedings of the 23rd International Conference on Computational Linguistics: Posters. p. 36–44. COLING '10, Association for Computational Linguistics, USA (2010)
- [5] Bouazizi, M., Otsuki Ohtsuki, T.: A pattern-based approach for sarcasm detection on twitter. IEEE Access **4**, 5477–5488 (2016)
- [6] Bravo-Márquez, F.: Acquiring And Exploiting Lexical Knowledge For Twitter Sentiment Analysis. University of Waikato, Hamilton (2017)
- [7] Chollet, F., et al.: Keras. <https://keras.io> (2015)
- [8] Chu, T., Jue, K., Wang, M.: Comment abuse classification with deep learning. Von <https://web.stanford.edu/class/cs224n/reports/2762092> Pdf abgerufen (2016)
- [9] Conneau, A., Kiela, D.: Senteval: An evaluation toolkit for universal sentence representations. arXiv preprint arXiv:1803.05449 (2018)
- [10] Das, B., Chakraborty, S.: An improved text sentiment classification model using tf-idf and next word negation. <https://arxiv.org/abs/1806.06407> (2018)
- [11] Davidson, T., Warmesley, D., Macy, M., Weber, I.: Automated hate speech detection and the problem of offensive language, arXiv preprint arXiv:1703.04009 (2017)
- [12] Devlin, J., Chang, M., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018)
- [13] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2019)
- [14] Feinberg, T., Robey, N.: Cyberbullying. The education digest **74**(7), 26 (2009)
- [15] Golbeck, J., Ashktorab, Z., Banjo, R., Berlinger, A., Bhagwan, S., Buntain, C., Cheekalos, P., Geller, A., Gnanasekaran, R., Gunasekaran, R., Hoffman, K.: A large labeled corpus for online harassment research. In: Proceedings of the 2017 ACM on web science conference. p. 229–233 (2017)

- [16] Gomez, R., Gibert, J., Gomez, L., Karatzas, D.: Exploring hate speech detection in multimodal publications. In: The IEEE Winter Conference on Applications of Computer Vision. p. 1470–1478 (2020)
- [17] Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
- [18] Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., Wu, Y.: Exploring the limits of language modeling. arXiv preprint arXiv:1602.02410 (2016)
- [19] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
- [20] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
- [21] M. Thelwall, K. Buckley, G.P.D.C., Kappas, A.: Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology* **61**(12), 2544–2558 (2010)
- [22] Maiya, A.S.: Ktrain: A low-code library for augmented machine learning. arXiv preprint arXiv:2004.10703 (2020)
- [23] Miranda, L.J.V.: Pyswarms (2017), <https://pyswarms.readthedocs.io/en/latest/>, [Online: Accessed October 1, 2021]
- [24] Naseem, U., Razzak, I., Eklund, P., Musial, K.: Towards improved deep contextual embedding for the identification of irony and sarcasm. In: 2020 International Joint Conference on Neural Networks (IJCNN). p. 1–7. IEEE (2020)
- [25] Nguyen, D.Q., Vu, T., Tuan Nguyen, A.: BERTweet: A pre-trained language model for English tweets. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 9–14. Association for Computational Linguistics, Online (Oct 2020). <https://doi.org/10.18653/v1/2020.emnlp-demos.2>, <https://www.aclweb.org/anthology/2020.emnlp-demos.2>
- [26] Nguyen, D., Vu, T., Nguyen, A.: Bertweet: A pre-trained language model for english tweets. arXiv preprint arXiv:2005.10200 (2020)
- [27] P. Smith, J. Mahdavi, M.C.S.F.S.R., Tippet, N.: Cyberbullying: Its nature and impact in secondary school pupils. *Journal of Child Psychology and Psychiatry* **49**(4), 376–385 (2008)
- [28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
- [29] Q. Huang, V.S., Atrey, P.: Cyber bullying detection using social and textual analysis. In: Proceedings of the 3rd International Workshop on Socially-Aware Multimedia - SAM '14 (2014)
- [30] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018)
- [31] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. *OpenAI blog* **1**(8), 9 (2019)
- [32] Raisi, E., Huang, B.: Cyberbullying identification using participant-vocabulary consistency. <https://arxiv.org/abs/1606.08084> (2016)
- [33] Salawu, S., He, Y., Lumsden, J.: Approaches to automated detection of cyberbullying: A survey. *IEEE Transactions on Affective Computing* (2017)
- [34] Smith, L.: A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820 (2018)
- [35] Upadhyay, A., Nguyen, T.T., Massie, S., McCall, J.: WEC: Weighted ensemble of text classifiers. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2020)
- [36] V. Menger, F.S., Spruit, M.: Comparing deep learning and classical machine learning approaches for predicting inpatient violence incidents from clinical text. *Applied Sciences* **8**(6), 981 (2018)
- [37] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.: GLUE: A multi-task benchmark and analysis platform for natural language understanding. In: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. pp. 353–355. Association for Computational Linguistics, Brussels, Belgium (Nov 2018). <https://doi.org/10.18653/v1/W18-5446>, <https://aclanthology.org/W18-5446>
- [38] Wikimedia Commons: Saddle point between maxima (2016), https://commons.wikimedia.org/wiki/File:Saddle_Point_between_maxima.svg, [Online: Accessed November 1, 2021]
- [39] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020), <https://www.aclweb.org/anthology/2020.emnlp-demos.6>
- [40] Wulczyn, E., Thain, N., Dixon, L.: Ex machina: Personal attacks seen at scale. In: Proceedings of the 26th International Conference on World Wide Web. p. 1391–1399 (2017)
- [41] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.: XLnet: Generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems, p. 5753–5763 (2019)
- [42] Zhao, R., Mao, K.: Fuzzy bag-of-words model for document representation. *IEEE Transactions on Fuzzy Systems* **26**(2), 794–804 (2018)
- [43] Zhao, R., Zhou, A., Mao, K.: Automatic detection of cyberbullying on social networks based on bullying features. In: Proceedings of the 17th international conference on distributed computing and networking. p. 1–6 (2016)

BIOGRAPHY SECTION

Christopher Graney-Ward finished BSc (Computer Science with AI) at Northumbria University, UK in 2021. His interest is in the application of AI/machine learning in the context of social media. His research work is centered around hate speech detection in social media through Machine learning and Deep learning approaches.

Biju Issac received BE degree in Electronics and Communications Engineering, MCA (Master of Computer Applications) degree, and PhD in Networking and Mobile Communications. He is an academic staff at Northumbria University, UK, since 2018. He has authored more than 100 refereed conference papers, journal papers, and book chapters. His research interests are in networks, cybersecurity, machine learning (text mining/image processing) and technology in education.

Lida Ketsbaia finished BSc (Computer and Digital Forensics) at Northumbria University and is doing PhD in AI-based Digital Forensics since 2019. Her PhD research work is centered around hate speech detection using Machine learning, Deep learning and Natural Language Processing (NLP).

Seibu Mary Jacob is an academic staff in Mathematics at Teesside University, UK. She took her Bachelor degree in Mathematics (BSc) and Master degree in Mathematics (MSc), along with Post Graduate Diploma in Computer Applications (PGDCA), Bachelor degree in Mathematics Education (BED) and PhD in Mathematics Education. She has authored more than 30 research publications as book chapters, journal papers and conference papers.