IRE Major Project Report
HatEval - Group 24

Team members
Vaibhav Bajaj
Sriven Reddy
Kripa Anne
Karthika Ramineni

Mentor
Vivek Anand

Link to the code: *https://github.com/vaibhavb26/hatEval*

Link to webpage of the project: *HatEval Website*

1. Abstract:

The aim of this project is two-fold:

- We initially classify hateful tweets from a corpus where hate speech against women or immigrants has been identified, as aggressive or not aggressive.
- Additionally, we classify these hateful tweets to identify if the target being harassed is a generic group of people or a specific individual.

We employ machine learning methods like logistic regression and support vector classification with various hyperparameters to initially perform the above binary classification tasks. We have also used LSTM on word embeddings and CNN on character level.

2. Related Work:

Recent years have seen an increasing number of research on hate speech detection as well as other related areas. As a result, the term ‘hate speech’ is often seen to co-exist or become confused with other terms such as ‘offensive’, ’profane’, and ‘abusive’ languages, and ‘cyber bullying’. By exploring some of the existing literature and research done on the classification of hate speech, we can proceed to improve on the results provided.

SemEval-2019 Task 5: Multilingual Detection of Hate Speech

The above paper contains the description of the task of the HatEval at SemEval 2019. It provides an in depth description of the task details and sub tasks and also provides a description of the data used along with its methods of collection and validation. It also provides a summary of the overall competition and gives an analysis and discussion about the various submissions and their results.

INF-HatEval at SemEval-2019 Task 5: Convolutional Neural Networksfor Hate Speech Detection

The above paper provides the results of using CNNs for hate speech evaluation. It was done on the task and data provided by HatEval in SemEval 2019. The paper explains and summarizes the existing architectures present for classifying hate speech and proceeds to propose a different method of using CNNs for classifying hate speech in tweets.

Hate Speech Detection Using a Convolution-LSTM Based DeepNeural Network

This paper introduces a new method based on a deep neural network combining convolutional and long short term memory networks, and conducts an extensive evaluation of the method against several baselines and state of the art on the largest collection of publicly available datasets to date. It also shows that the proposed method outperforms state of the art on 6 out of 7 datasets and provide an analysis of their method.

Deep Learning for Hate Speech Detection in Tweets

This paper proposes a deep learning method to improve on existing state of the art on hate speech detection in tweets which has been defined as being able to classify a tweet as racist, sexist or neither. The paper highlights extensive experiments done with multiple deep learning architectures to learn semantic word embeddings to handle the complexity of the task.

Character-level Convolutional Networks for Text Classification

This article offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. We constructed several large scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional models such as a bag of words, n-grams and their TF-IDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks.

3. Baseline methodologies

- General preprocessing and cleaning of data through pattern matching that includes:
    - Converting to lowercase
    - Removing special characters that won't contribute positively to the accuracy (like the '#'s used in hashtags and punctuation)
    - Removing stop words
    - Removing URLs
    - Removing mentions
    - Stemming words
- Tokenization using bag of words (sklearn vectorization).
- Logistic Regression model
- Support vector machines model
- Word level embedding : 1-gram through 3-gram
- Character level embedding : 2-gram through 6-gram
- LSTM model on word embeddings
- CNN model on char level embeddings

4. Architecture

1) Preprocessing

The first big challenge was to clean and process the tweets to remove the noise and other unnecessary words, which would be quite useful for training any model and would be helpful in producing better results. Hopefully, we got to know about **TweetTokenizer** which is part of NLTK library, we used it to remove Twitter username handles and replace repeated character sequences of length 3 or greater, with sequences of length 3 which generally present as noise in tweets. Then there

we thought of experimenting our models by modifying the tweets in a few different ways and hence we wrote functions to try out the following:

- Remove_URL: process tweets with and without removal of URL
- Remove_Hashtags: process tweets with and without removal of Hashtag
- Remove_num: process tweets with and without removal of Numbers
- Remove_Swords: process tweets with and without removal of Stopwords
- Stem_tweet: process tweets with and without stemming of words

We found that the accuracy of various models change when we use the above functions to modify the tweets, in general on a positive side. Preprocessing procedure depends upon the model being used, we used all of the above functions for training our model.

2) Word Embedding Model with LSTM

A word embedding is a class of approaches for representing words and documents using a dense vector representation. Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from the text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding.

- First, data is prepared using the Tokenizer API also provided with Keras where the tweets are converted into a list of sequences (one per text input).
  These sequences are padded to make all of them of the same size. The list of sequences is transformed into a 2D Numpy array of shape (number_of_tweets, MAXLEN). MAXLEN if not provided, is the length of the longest sequence otherwise. Sequences that are shorter than MAXLEN are padded with value at the end and sequences longer than MAXLEN are truncated so that they fit the desired length.
  The vector of expected outcome is converted into a binary matrix representation of it.

- Embedding layer is used as the first layer in the model. It requires that the input data be integer encoded so that each word is represented by a unique integer.
  The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. The output of the Embedding layer is a 2D vector with one embedding for each word in the input sequence of words.

- Next, a Bidirectional LSTM layer is added. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step. This

structure allows the networks to have both backward and forward information about the sequence at every time step.

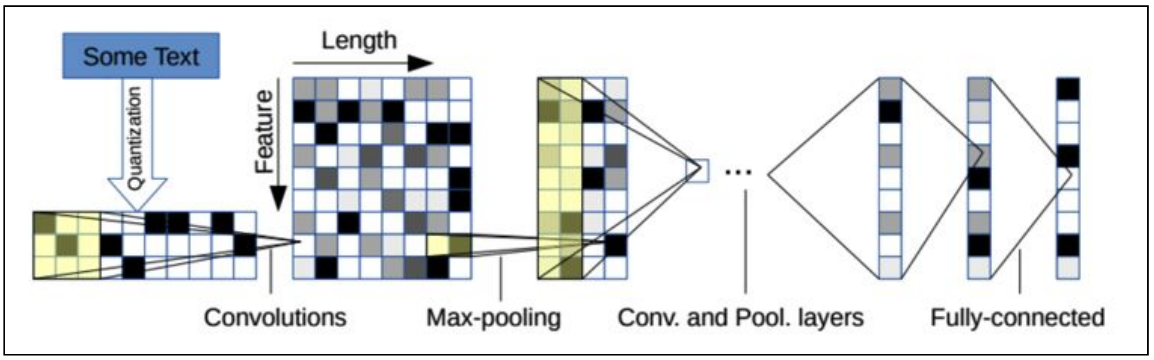
- Lastly, a Dense layer is introduced where we have used softmax as an activation function. A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 140)               0
_________________________________________________________________
embedding_4 (Embedding)      (None, 140, 128)          812416
_________________________________________________________________
bidirectional_4 (Bidirection (None, 128)               197376
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,010,050
Trainable params: 1,010,050
Non-trainable params: 0
```

3) Character level CNN

The next model we tried was character based CNN since texts in tweet corpus is a bit different than any other regular text corpus. In tweets, word shorthands e.g. ‘FYKI’ instead of ‘for your kind information’, repeating characters e.g. ‘yaaayyyyyy’ to express feelings, emoticons using symbols etc are used. These are not general English words and these usages may differ from person to person. As for example, ‘yaaaayyyyy’ and ‘yaayyy’ both are the same word. So character level model would work better than word level model.
In this approach, we came up with the idea of embedding the characters of the words and then pass it through the CNN network.

a) The model accepts a sequence of encoded characters as input. The encoding is done by prescribing an alphabet of size m for the input language, and then quantize each character using 1-of-m encoding (or
“one-hot” encoding). Then, the sequence of characters is transformed into a sequence of such m sized vectors with fixed length l0. Any character exceeding length l0 is ignored, and any characters that are not in the alphabet including blank characters are quantized as all-zero vectors. The character quantization order is backward so that the latest reading on characters is always placed near the beginning of the output, making it easy for fully connected layers to associate weights with the latest reading.



b) The alphabet used in all of our models consists of 70 characters, including 26 english letters, 10 digits, 33 other characters and the new line character. The non-space characters are: abcdefghijklmnopqrstuvwxyz0123456789 -,;.!?:’’’/\|_@#$%ˆ&*˜‘+-=<>()[]{}

c) The embeddings are then passed to the CNN network whose architecture is described below. We also insert 2 dropout modules in between the 3 fully-connected layers to regularize. They have dropout probability of 0.5.

d) The architecture of the CNN used is as follows:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 512)	0
lambda_1 (Lambda)	(None, 512, 69)	0
Conv1 (Conv1D)	(None, 506, 256)	123904
MaxPool1 (MaxPooling1D)	(None, 168, 256)	0
Conv2 (Conv1D)	(None, 162, 256)	459008
MaxPool2 (MaxPooling1D)	(None, 54, 256)	0
Conv3 (Conv1D)	(None, 52, 256)	196864
Conv4 (Conv1D)	(None, 50, 256)	196864
Conv5 (Conv1D)	(None, 48, 256)	196864
Conv6 (Conv1D)	(None, 46, 256)	196864
MaxPool3 (MaxPooling1D)	(None, 15, 256)	0
flatten_1 (Flatten)	(None, 3840)	0
dense_1 (Dense)	(None, 1024)	3933184
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
output (Dense)	(None, 2)	2050

Total params: 6,355,202
Trainable params: 6,355,202
Non-trainable params: 0

5. Evaluation Mechanisms

- **Precision:** the ability of a classification model to identify only the relevant data points.

$$Precision = \frac{\#\ of\ True\ Positive}{\#\ of\ True\ Positive + \#\ of\ False\ Positive}$$

- **Recall:** the ability of a model to find all the relevant cases within a dataset.

$$Recall = \frac{\#\ of\ True\ Positive}{\#\ of\ True\ Positive + \#\ of\ False\ Negative}$$

- **F1-score:** F1-score is the harmonic mean of precision and recall taking both metrics into account in the following equation:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

- **ROC_AUC score:** a metric which falls between 0 and 1 with a higher number indicating better classification performance.

6. Data Analytics:

Average length of all tweets in the dataset: 125.588 characters
Average length of hateful tweets in the dataset: 127.136 characters
Average length of all non-hateful tweets in the dataset: 124.636 characters

Maximum tweet length: 63 words
Average tweet length: 22.12 words
Minimum tweet length: 1 word

Total number of hateful tweets : 3783
Number of aggressive tweets : 1559 (40% of hateful samples)
Number of aggressive *and* non-hateful tweets : 0

Number of individual-targeted hateful tweets : 1341 (35% of hateful samples)
Number of group-targeted hateful tweets : 2442 (65% of hateful samples)

Number of features used in word level embedding : 89709
Number of features used in character level embedding : 253035

7. Results:

SVM and Logistic Regression
We found that both the logistic regression and the SVM method performed equally when detecting hate speech. Word level or character level recognition did not change the results by noticeable margins. Both methods performed much better at target classification than at

aggression detection. We tried the same methods after including the mentions (i.e. not removing them during data preparation). Adding mentions did not positively affect the classification accuracy in any way, except in the case of character level embeddings with SVM for both tasks.

a) Performance on Target Classification
   WordLevel + Logistic Regression : 0.906
   WordLevel + Logistic Regression (with mentions) : 0.903
   CharLevel + Logistic Regression : 0.908
   CharLevel + Logistic Regression (with mentions) : 0.903
   WordLevel + SVM : 0.911
   WordLevel + SVM (with mentions) : 0.911
   CharLevel + SVM : 0.882
   CharLevel + SVM (with mentions) : 0.906

b) Performance on Aggression Detection
   WordLevel + Logistic Regression : 0.680
   WordLevel + Logistic Regression (with mentions) : 0.666
   CharLevel + Logistic Regression : 0.673
   CharLevel + Logistic Regression (with mentions) : 0.667
   WordLevel + SVM : 0.704
   WordLevel + SVM (with mentions) : 0.687
   CharLevel + SVM : 0.665
   CharLevel + SVM (with mentions) : 0.669

While using SVM and Logistic Regression on the hateful tweets (Phase 1),
A few samples did not fit the model and were wrongly classified.

In the target classification task, examples of false positives were tweets with abusive language that wasn't necessarily hateful to a person or group.
For example,
seeing ur close girlfriends in public: HEY U STUPID FUCKING SLUT BITCH I HATE U seeing girls u dont really like: hey girl omgggg how u beeeennn its been foreverrrrrr

Examples of false negatives had phrases like 'hysterical women' commonly occurring, that didn't contain any abuse but rather, veiled insults.
For example,

A. *You girls were working it today @CoryBooker @SenKamalaHarris. Its too bad you both come across as hysterical women.*
B. *@thebettinaarndt Hysterical women and she men by the looks of it*

LSTM and CNN

The accuracies for both the tasks are as follows:

	CNN	**LSTM**
Target classification	0.798	0.815
Aggression detection	0.791	0.604

On char level CNN we went one level deeper to understand the effect of stemming, stop word removal on the overall performance of the model. The results are as follows:

Target Classification with CNN:

	Precision	**Recall**	**F1 Score**
With Stopword removal and stemming	[0.7611 0.8889]	[0.9038 0.7355]	[0.8263 0.8020]
Without Stopword removal	[0.7654 0.8804]	[0.8942 0.7397]	[0.8248 0.8039]
Without Stemming	[0.8088 0.8712]	[0.875 0.8036]	[0.8406 0.8361]
Without Stemming and Stopword removal	[0.8292 0.8288]	[0.8173 0.8401]	[0.8232 0.8344]

Aggression Detection with CNN:

	Precision	**Recall**	**F1 Score**
With Stopword removal and stemming	[0.6523 0.7711]	[0.7811 0.6530]	[0.7030 0.6918]
Without Stopword removal	[0.6362 .7018]	[0.7712 0.6590]	[0.7388 0.7010]
Without Stemming	[0.7010 0.7523]	[0.7582 0.6988]	[0.7570 0.7426]
Without Stemming and Stopword removal	[0.6812 0.7045]	[0.6918 0.7136]	[0.6812 0.7010]

Target Classification with LSTM:

	Precision	**Recall**	**F1 Score**
With Stopword removal and stemming	[0.7651 0.8944]	[0.9086 0.7351]	[0.8307 0.8070]
Without Stopword removal	[0.7256 0.9068]	[0.9279 0.6667]	[0.8143 0.7684]
Without Stemming	[0.6735 0.9248]	[0.9519 0.5616]	[0.7888 0.6989]
Without Stemming and Stopword removal	[0.6667 0.937]	[0.9615 0.5434]	[0.7874 0.6879]

Aggression Detection with LSTM:

	Precision	**Recall**	**F1 Score**
With Stopword removal and stemming	[0.5645 0.5897]	[0.7847 0.33823]	[0.6566 0.4299]
Without Stopword removal	[0.5851 0.6000]	[0.7399 0.4265]	[0.6535 0.4986]
Without Stemming	[0.5865 0.5839]	[0.6996 0.4608]	[0.6380 0.5151]
Without Stemming and Stopword removal	[0.5823 0.6486]	[0.8251 0.3529]	[0.6827 0.4571]

8. Analysis

While using LSTMs and CNNs on the hateful tweets (Phase 2), a few samples did not fit the trained model and were wrongly classified. We found 645 correctly classified tweets, 188 false negatives and 167 false positives.

(Correctly classified as NOT hateful towards any target)
eg: *'We must reject the "globalization of indifference" toward refugees, says the president of the International Catholic Migration Commission. https://t.co/RtZRiHxw2v'*
Processed tweet - *'must reject global indiffer toward refuge say presid intern cathol migrat commiss'*

(Correctly classified as hateful to a targeted group)
eg: *"@potus @realDonaldTrump Why have you done NOTHING to stop #INDIA stealing AMERICAN JOBS?We are watching, won't be conned by 5 BILLION WALL funding. #BuildThatWall NOW, tired of useless talk.It won't stop #INDIA flying in, stealing American jobs,STOP #INDI",*
Processed tweet - *'done noth stop steal american job watch con billion wall fund tire useless talk.it stop fli steal american job stop'*

(Wrongly classified/false positives)
eg: *"@lilmaggie186 @Jim_Jordan Illegals too? #BuildThatWall #AmericaFirst God Bless America and @POTUS',*
Processed tweet - *'illeg god bless america'*
In this example, after processing and cleaning the data by removing hashtags etc, the model is run on the remaining data. In short tweets such as this, the message of the tweet cannot be conveyed through a few words. So by just looking at ‘illeg’ or ‘america’, the model understandably classified the tweet as hateful to a group.

(Wrongly classified/false negatives)
eg: *'@ABCPolitics They illegally invaded our nation. They should get NOTHING. Period. #AmericaFirst #BuildThatWall #MakeItEvenHigher',*
Processed tweet - *'illeg invad nation get noth period'*

The absence of mention of a particular group might have contributed to the wrong classification.

9. Difference in Original timeline and implementation:

We have achieved hate speech identification using models and have used simple logistic regression and SVM as well as LSTM and CNN models to identify target and aggressive tweets as per the original scope document. The methodology used and the implementation of project was as specified in the scope document. We have implemented the code as per the original timeline provided.