Machine Learning In Natural Language Processing: A "Hate Speech Detection" Comparative Study & Application

Project Repository reference [1]

Md Saroar Jahan
ITEE department
University Of Oulu
Oulu, Finland
saroarjahan.bd@gmail.com

***Abstract*—In the internet at large, there leftovers an area of discourse largely deemed too marginal to analyze at any length: openly racist or hate speech. It remains unexamined, in part, because much attention has been given to real life social racism, not much for internet. Recently, technology has allowed openly racist groups to shift strategies for creating and maintaining their own identity. Many documents exist in the Internet that must be examined. Technology is advanced enough now to identify hate speech by using natural language processing. In this project we have created a web application that is capable of identifying hate speech from online and free documents. This report, we will describe all details about the project methodology including dataset preparation, feature engineering, model training and compare each classifier accuracy that have been used in the project and empirical result of each classifier in the thought of better understanding of natural language processing.**

Keywords—npl, machine learning, hate speech

I. INTRODUCTION

There is always a serious risk that he or she may be the target of harassment through online like message board forums, commentaries or social network. For a bad user experience, words and sentences may be great impact for what they've done are awkwardly not rare in internet and can require a profound by the civility of a community or a user's experience. For overcome this issue, many internet companies have standard and guidelines that users must adhere to and employ human editors through the blacklisted systems for catching bad language and remove a post. Now a day's people are increasingly communicating online, the need for high quality automated abusive language classifiers. For example, Facebook came under fire for hosting pages which were using hateful and violent words against woman like raping your friend just for laughs and also showed that kicking your girlfriend in the fanny due to not making a sandwich in 2013. Several major companies either pulled or threatened to remove their ads from the social site called Facebook because a petition was started which amassed over 200000 supporters just within 1 days. This will create impact on several companies which hosts user generated content will have a moderation issue. For detecting the abusive language in online automatically is great concern about the topic and task but the main problem is that the prior art not been very unified. This makes slowing the progress. Different research has proposed of Natural Language Processing to web sciences and also artificial intelligence. Within the research, several methods have published in the last three years for detecting the abusive language online. Moreover, abusive language can be a bit of catchall term. There are some studies related to this topic which we have covered that shows different approaches of identifying hate speech using natural language processing and text classification.

1. MOTIVATION

Ziqi Zhang proposed about a novel Deep Neural Network structure which is serving as effective feature extractors [6]. And also, usage the background of information in the form of different word embeddings pre-trained from unlabeled corpora. In that paper, the author also did an empirically evaluate their methods on the largest collection of hate speech datasets based on Twitter. Their works actually is to understand the extremely unbalanced nature of the typical datasets and then investigating methods on the principles of effective feature extractors by DNN architecture. This architecture is incorporated with background information from large unlabeled corpora in the form of pre-trained word embeddings models. After that they were tried to empirically evaluate, compare and analyses the performance of several proposed methods against state of the art on the task of hate speech detection. From that paper we got an idea about how they process their whole research for hate speech analysis. At first, they showed chalL.Luo / Hate Speech Detection - the Difficult Long-tail Case of Twitter 19 which lenging nature of identifying hate speech from the short text. Additionally, in that paper they also showed that micro-averaging over both hate and non-hate classes in a dataset can be adopted. Furthermore, it will toward to the evaluation of the dominant non-hate class in a dataset through masking a method's ability to identify the real hate speech. Secondly, they tried to build a CNN architecture where they wanted to discover the implicit features that can be useful for detecting hate speech from the short text. Thirdly, they have some drawbacks like they didn't draw conclusion whether one-word embeddings option is consistently better than others.

From Anna Schmidt and Michael Wiegand, they have showed a survey on hate speech detection. Their survey also described about the key areas for recognizing the utterances using natural language processing and also, they tried to emphasize the limits of those approaches [7]. In that paper, they also gave a short comprehensive and structured overview of automatic hate speech detection. Additionally, they have also focused on feature extraction for the existing research. Their task actually framed as a supervised learning problem. In that paper, they were using a generic feature which contains the bag of words or embeddings systematically. Their paper also showed that character level approaches is work better than token level approaches. Lexical resources help out the classification but various complex features using linguistic knowledge. They have mentioned that information derived from text may not be suggesting the presence of hate speech but also complemented by meta-information or information from other modalities. For better findings features and methods, they set up a benchmark for hate speech detection.

Accordingly, to Anat and Ardina, they have studied about the overt hate speech and covert discriminatory practices circulate on Facebook [8]. They have argued with that hate speech and user's motivation and actions are not creating discrimination, but are also formed by a network of ties between the platform's policy, its technological affordances, and the communicative acts of its users. In that paper, their argument is supported with longitudinal multimodal content and network analysis of the data extracted from Facebook pages which consisted of seven extreme right political parties in Spain between 2009 and 2013.From that paper, we got that Spanish extreme-right political parties primarily implicate discrimination, which is then taken up by their followers who use overt hate speech in the comment space. Their study actually concerns about the Facebook corporate logic and also technological affordances.

From Shervin Malmasi and Marcos Zampieri, they have discussed about examine methods to detect hate speech in social media. At first, they have established lexical baselines for this task through the supervised classification method. For a feature, they used word n-grams and word skip-grams. They have obtained obtain results of 78% accuracy in identifying posts across three classes. From their results, the noticed a main challenging discriminatory profanity and hate speech each other. In that paper, they have applied text classification methods for distinguishing hate speech, profanity and other texts. For baselines they have used SVM classifier.

From Hitesh Kumar Sharma, T.P. Singh, K Kshitiz, Harsimran Singh and Prince Kukreja, they were involved for determining ways to identify bullying in text by analyzing and experimenting with different methods for classifying bullying comments [13]. They have proposed an efficient algorithm to identify the bullying test and aggressive comments and later they also want to analyse these comments for checking the validity. They used NLP and machine learning for analyzing the social comment and identified the aggressive effect of an individual or a group. They also try to notice their audiences that best performing classifier acts as the core component in a final prototype system that can detect cyberbullying on social media.

According to Thomas Davidson, Dana Warmsley, Michael Macy and Ingmar Weber, automatic hate-speech detection on social media is the key challenges for separation of hate speech from other instances of offensive language [14]. For classifying all messages containing particular terms as hate speech they used lexical analysis for its low precision. Also, they try to mention that their previous work used supervised learning but it has failed for distinguishing between two categories. They also used crowd-sourced hate speech lexicon to collect tweets containing hate speech keywords. Their Close analysis of the predictions and the errors showed that they can separate the hate speech from others offensive language. Within their research they also noticed that their racist and homophobic tweets are classified as hate speech. They also tried to mention that tweets without explicit hate keywords are also more difficult to classify.

The work we intended to accomplish here in this project is quite fascinating, unique in many ways, and probably the best among all work that has been done so far. As an example, no work has compared so many classifier like we have done in this project. Not only this, we have worked with three (3) type of dataset simultaneously. In addition, we have successfully developed a fully functional web application at the end of the project.

II. Project Description

This project aims to put forward an efficient approach for identifying hate speech in online forum and free documents.

This project is divided into two parts. First part we have worked with wikipedia dataset corpus and second part we have worked with Zi. Zhang Dataset. Wikipedia dataset described in section IV and Zi. Zhang dataset preparation described in section VIII.

Since this project purpose to analyse aggression, toxicity and attack; therefore, its required three types of datasets and a neutral dataset as well. Prior to this goal we have to construct a Neutral Dataset consists of non-aggressive, non-personal and non-toxicity datasets using algorithm written in python code, and another randomly generated sentences from the Wikipedia website using another python program.

After preparing the dataset its fundamental to construct a classifier that classifies a text or sentence into either a personal attack or non-attack. Since, we have to use various types of classifiers (Naive Bayes, Linear Classifier, SVM Model etc); therefore, we have to discuss the various combinations of features of classifiers, accuracy rate, and F1 score. After creating functionality for a single type (example 'Attack') we have to do the same for other two types namely 'aggression' and 'toxicity' cases.

Now we would like a classifier to learn the three types of hate speech simultaneously. We have to use the structure of

the dataset and design an ensemble of classifiers for classifying a sentence into neutral, attack, aggression and toxicity with various combinations of feature sets in order to test the performance of the various classifiers. We must need a comparative report analysis with metric F1 score and accuracy of classification.

After classifying post by classifier, this project required to use the LIWC features alone or combined with other features investigated in order to test the performance of the suggested classifiers.In this case, we would prefer majority voting rule to test the combination of the various classifiers.

Finally, the final step is to design a GUI Graphical User Interface for user where they can input the URL of the online forum and it must output the identification of various categories of the hate speech.

III. Wiki Dataset description

All data we have used and generated for the Analysis of Hate Speech (AHS) project is available under free licenses on the Wikipedia Talk Corpus on Figshare [2]. This dataset is open source. There are currently two distinct types of data included. A corpus of all 95 million user and article talk diffs made between 2001–2015 which can be scored by our personal attacks model. An annotated dataset of 1m crowd-sourced annotations that cover 100k talk page diffs (with 10 judgements per diff) for personal attacks, aggression, and toxicity. HSA used two types of dataset namely Wikipedia Comments Corpus and Annotations Corpora.

Wikipedia Comments Corpus is comments from English Wikipedia talk pages. Comments are grouped into different files by year, and by the user or article talk-namespace. Comments are generated by computing diffs over the full revision history and extracting the content added for each revision.

We have used annotated selected fragments of the wikipedia comments corpus for personal attacks, aggression, and toxicity.

Personal Attacks, 100k labeled comments from English Wikipedia by approximately 10 annotators via Crowdflower on whether it contains a personal attack. We also include some demographic data for each crowd-worker.

Aggression, 100k labeled comments from English Wikipedia by approximately 10 annotators via Crowdflower on how aggressive the comment was perceived to be. We also include some demographic data for each crowd-worker.

Toxicity, 160k labeled comments from English Wikipedia by approximately 10 annotators via Crowdflower on a spectrum of how toxic the comment is (perceived as likely to make people want to leave the discussion) to how healthy to conversation the contribution is.

We have also used other two type of dataset that has been programmatically developed. One is a negation of the existing hate dataset and another is randomly collected sentence from wikipedia.

IV. General methodology

One of the generally utilized common linguistic processing assignments in various business issues is "Text Classification". The objective of text classification is to consequently order the text archives into various characterized classes. Text classification could be used in various purposes, as example, thoughtful gathering of people decision from online life, identify of junk emails, auto labeling of client questions, arrangement of news articles into characterized subjects etc.

Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for train a classifier. An end-to-end text classification pipeline is composed of three main components namely dataset preparation, feature engineering and model training. Below a brief description presented towards the project general methodology.

A. Dataset Preparation

The first step is the dataset preparation step which includes the process of loading a dataset and performing basic pre-processing. Since we have to use three types of classification, therefore, we will use three types of dataset namely attack, aggression and toxicity. The dataset is then labelled as a comment attack if the majority of annotators did so, after then join labels and comments. One of the important parts of the dataset preparation is split dataset into training and validation sets. This train and validation data would be used in the classification phase.

B. Feature Engineering

The next step is the Feature Engineering in which the raw dataset is transformed into flat features which can be used in a machine learning model. This step also includes the process of creating new features from the existing data. For feature engineering purpose, we have used some popular feature engineering method namely count vector, TF-IDF (word level, character level,) and word embedding as features.

C. Model Training

The final step is the Model Building step in which a machine learning model is trained on a labelled dataset. There are many popular machine learning model that is suitable to train a final model. For this particular project we would like to work with Naive Bayes Classifier, Linear Classifier, Support Vector Machine (SVM), Shallow Neural Networks and some important Deep Neural Networks classifier.

D. Linguistic Inquiry and Word Count (LIWC)

Text or sentences will be identified when it passes through classifier. Since we have used various type of classifier then its important to check which classifier can

detect more efficiently. To analyze each classifier detected sentences relevaness , we have used empath [12].

E. Graphical User Interface (GUI)

To make the working code more attractive to interact, we decide to convert our whole project into a web application. For web application we would use Django which is a python build web framework. We would integrate our developed code in Django backend, in front end we would use html5 and css3.

F. Text Classifier Performance improvement

In this project, we had tried different techniques to improve the accuracy of the text classifiers.

.

V. DETAILED METHODOLOGY

A. 1.Dataset Preparation

For the purpose of this project, we are using dataset of Wikipedia Talk Corpus on Figshare [2]. The dataset consists of more than 300MB of text comments and their annotations. To prepare the dataset, first we have downloaded data then read the data as comments and annotations.

A. 2.Neutral Dataset Building

In Analysing and detecting hate speech online, we are faced in most cases by non-hate speech or neutral sentences. The Neutral sentences are those which does not imply or include hate speech or words. In order to build classifiers which have the ability to point out the hate speech from neutral speech, the classifiers have to learn what is hate and what is neutral. For the purpose of that, we have constructed the neutral datasets by two main methods: one is to build a python code which to do the negation of the already provided attack, toxicity and aggression datasets. And two, is to randomly extract sentences from wikipedia using another python code. The generated neutral datasets are to be used later on in the process of building classifier and rising performance of prediction. The two methods are explained in the next lines.

A. 2.a. Negation of the datasets

One way to generate a non-hate speech is to basically do the negation of the datasets. building the negation of a sentence is not that simple, the difficulty rises when it is a long sentence, and yet it is harder when it comes to non-formal speech. In order to have a negation of a sentence we have built a python code to do the job. The algorithm of it is as following:

1. Load a sentence and initiate a new list sen_neg.
2. Perform pos_tagging on the word_tokenizing of the sentence
3. For word in pos_tags do:

every time check if the word belongs to one of the verb forms or adjective forms, then perform either adding antonym instead of it, or add negation before it (with different forms of negations and stemming if it is a verb), or pass the word doing nothing, or remove the negation from the word.

The code of the algorithm is better seen in the provided github link in the negationClass[1]. After having the code ready to start, we take the prepared and clean datasets and push them into the code. This results in new datasets of the csv form containing over 100k of well negated sentences, this goes for the three datasets. we take an example of sentence before and after negation performed: easy and more complex sentences

Sentence	**Negation of sentence**
'This page will need disambiguation. '	'This page will not need disambiguation. '
I removed from scratch . in addition to your reasons it just looks better without it.	I did not remove from scratch . In addition to your reasons it just does not look better without it .

Table1: Example of sentence and negation of sentence

one can observe the quality of negation in the second sentence, how the built code could do the negation for the past simple, adding to it 'did not' before, and yet stemming the verb. And the same goes for the present tense verb. This makes sense that the output sentence is well understood and one may not detect that this is a computer generated negation.

A. 2.b Random sentences Dataset

The second way to generate a neutral dataset is to extract a random sentence from the Wikipedia webpage. Wikipedia provides a huge source of information and texts which can be used in education and research. In our case, for getting a random sentence we have used the random article generating link[11] that changes the article every reload of the page.
using this link, we have written a python code which generates random sentences from each article and saves the dataset into a csv file.

The code of generating the random sentences dataset is provided within the github link along with the dataset generated which contains over 30k sentences or paragraphs.

Now, for the provided datasets, we perform some manipulations, below all example code snippet for analysing of Attack [3].

Afterwards, the dataset is cleaned using the function **clean_text()** from the DataPrep class. The cleaning task would remove any no-sense words or numbers and characters, thus making the text cleaner which has the impact on the trained model.

Then the dataset labelled, comments identify as attack if the majority of annotators represent attack, after then join labels and comments.

B. Feature Engineering

The next following phase is the feature engineering. In this phase, raw text data will be modified into feature vectors and new features will be fashioned using the existing dataset. We have implemented the accompanying distinctive thoughts so as to get significant features from the dataset. We have used Count Vectors as features, TF-IDF Vectors as features, and Word Embeddings as features in our project. Below we will briefly discuss the implementation of each of these features.

a) Count Vectors as features: The notion of count vector is documentation of the dataset in which each line appointed to a record from the corpus and each column appointed to a term from the corpus, and each cell appointed to the identification of a specific term a specific document. The most straightforward count vector feature is that the vectorizer counts the number of times a token shows up in the document and uses this value as its weight, and this is what are we going to use in the paper.

b)TF-IDF Vectors as features: TF-IDF score speaks to the comparative significance of a term in the document and the whole corpus. TF-IDF score is made by two terms: the first calculate the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), figured as the logarithm of the quantity of the documents in the corpus ratio by the quantity of documents where the explicit term shows up. This project has used three type TF-IDF vector features namely Word level, N-Gram level, Character level.

Word level TF-IDF represent a score of every term in different documents, while N-grams are the combination of N terms together, and Character Level TF-IDF is the matrix representation of tf-idf scores of character level n-grams in the corpus.

c) Word Embedding as features : A word embedding is a procedure of demonstrating words and documents with a dense vector representation. The place of a word inside the vector space is learned from text and is constructed on the words that surround the word once it is used. We can train word embedding using the input corpus itself; however, for our project we have used pre-trained word embedding namely FastText [4]. There are other pre-trained word embedding like Glove, Word2Ve etc, these are open source.

Below snippet displays the procedure of using pre-trained word embeddings in the model. We have followed four fundamental steps during the process, (i)Loading the pre trained word embeddings , (ii)Forming a tokenizer object , (iii)Changing text documents to series of tokens and pad them and (iv) Create a mapping of token and their individual embeddings.

C. Model Training

There are a wide range of decisions of machine learning models which can be utilized to prepare the model training. The model training is the final step of text classification framework. The fundamental concept is to train a classifier with the features created in feature engineering step. For model training we have used Naive Bayes Classifier, Linear Classifier, Support Vector Machine, Bagging Models, Boosting Models Shallow Neural Networks and Deep Neural Networks. Deep Neural Network itself has some model namely Convolutional Neural Network (CNN), Long Short Term Modelr(LSTM), Gated Recurrent Unit (GRU), Bidirectional RNN, Recurrent Convolutional Neural Network (RCNN) and Other Variants of Deep Neural Networks. To understand text classification it's prerequisite to understand each of the model training deeply. In this report we will discuss each of the models briefly with our project code spapiit so that we have a better understanding of overall classification and project work.

1)Naive Bayes: In machine learning, Naive Bayes classifiers are a group of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) with an assumption of independence among predictors. A Naive Bayes classifier undertakes that the existence of a specific feature in a class is distinct to the existence of any other feature.[5].

2)Linear Classifier: In the field of machine learning, the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use Linear Classifier (Logistic Regression) Logistic measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function. One can read more about logistic regression.

3)SVM Model: Another important model training we have used in our project namely Support Vector Machine (SVM). We have used it because it's mainly a controlled machine learning procedure which can be used for together taxonomy and regression challenges. The model extracts a

finest probable hyper-plane / line that isolate the two classes. In short, SVM specified a set of training samples, each manifest as belonging to one or the other of two categories, an SVM training algorithm constructs a model that allocates new samples to one category or the other, making it a non-probabilistic binary linear classifier. Moreover, SVM model is a demonstration of the samples as points in space, plotted so that the samples of the distinct categories are divided by a clear gap that is as inclusive as possible. New samples are then plotted into that same space and predicted to belong to a category constructed on which side of the gap they belong.

4)Deep Neural Networks: Deep Neural Networks are additional multifarious neural networks in which the concealed layers perform greatly more compound processes than simple sigmoid or relu activations. Diverse kinds of deep learning models can be useful in text classification problems.

D. Concatenation

One of the goals of this project was to improve the classifier result. To improve classifier feature accuracy and f1 score, one approach was to contact different feature of the classifier. In this research we have concat Count and LIWC, Count and TFIDF, Count and TFIDF, Count and Ngram, TFIDF and LIWC, TFIDF and LIWC, Ngram and Ngram Char, TFIDF and Ngram Char. This concatenation carried out for Linear classifier, Naive Bayes, SVM and Neural Network. Concatenation result has been added to result section.

E. Graphical User Interface (GUI)

For every application its very important to have a visualize graphical interface for user. GUI can be developed by creating a desktop application, web application, mobile application or any existing related light frame work. Since our project is analysing hate speech from online or free document; therefore, we have chosen web UI. We have used Django web framework for developing our web based UI [9]. Django is fast therefore there will be no delay between backend and frontend. For frontend we have used html and css [10]. Because our developed web application is a complete example of working product that represent how this kind of application help in real life scenario; therefore, we have only connected best accuracy gained classifier into the GUI. In GUI we have shown classifier name, classifier accuracy, identified hate sentence and LIWC result of positive and negative emotion of detected text. Below figure depicts the developed GUI of the project, project more images are available in github inside GUI directory [1].

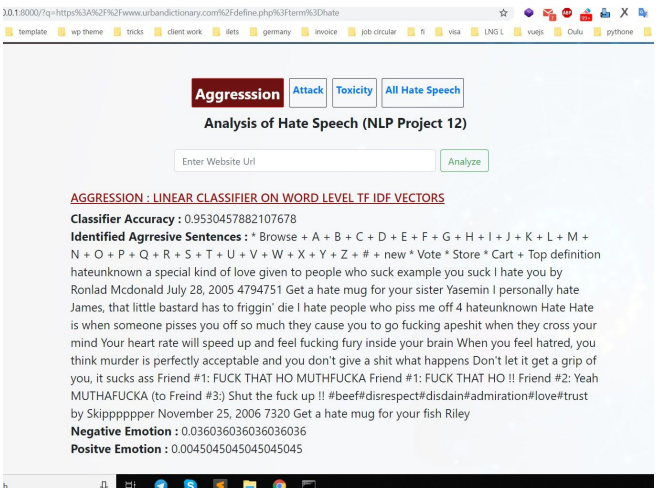

Figure 1: project GUI example for Aggression analysis

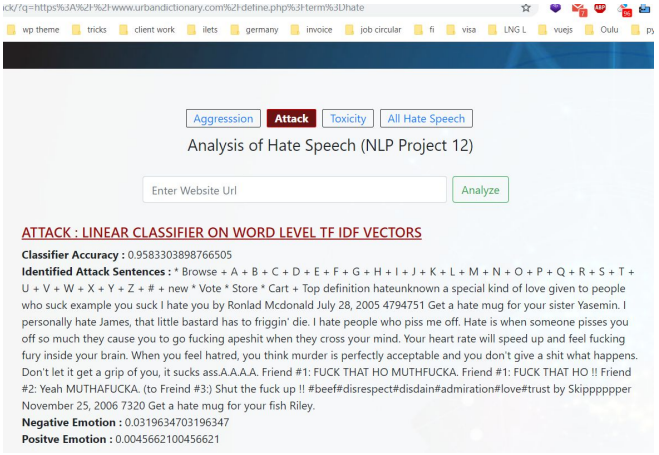

Figure 4: Project GUI example of Attack analysis

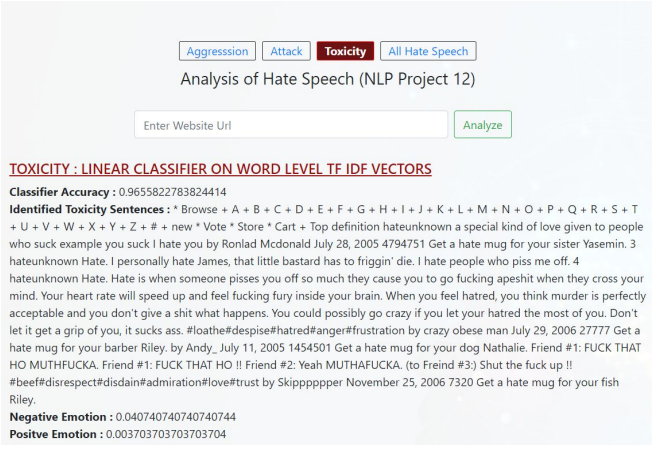

Figure 4: Project GUI example of Toxicity

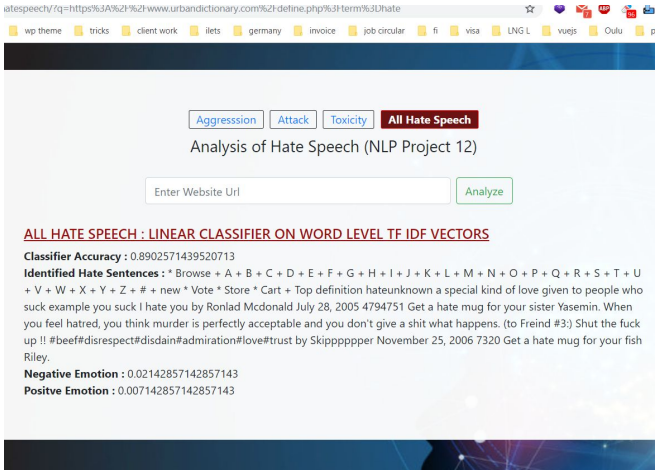

Figure 4: Project GUI example of three types of classifier simultaneously

F. Processing User Input

In GUI we have allowed user to input website url and analyze the result. To analyze the text from web pages, first we had to extract plain text from the web page. Next, we had split each sentence by using python split function. After then we have analyze each sentence with classifier and if the sentences contain hate speech then we have save those sentences in a separate variable and prepare for the LIWC test. Getting fresh text from website url was fairly simple; however, it was a very important step to accomplish.

G. Text Classifier Performance improvement

During the project we tried to achieve a decent accuracy by doing some developments in the overall framework. Below, the subsequent techniques has done during the improvement of classification model.

1. Text Cleaning : The text cleaning which was advantages to lessen the noise existing in text data in the form of stopwords, punctuation marks, suffix variations etc.

2. H-stacking Text / NLP features with text feature vectors: In the feature engineering section, we engendered a number of different feature vectors, joining them together, it helped to advance the correctness of the classifier.

3. Hyper-parameter Tuning in modelling : Tuning the factors is a significant phase, a number of parameters such as tree length, leafs, network parameters etc. can be fine-tuned to acquire a paramount result.

4. Concatenation : Two different classifier features (ex. count and liwc, count and tfidf etc.) has been added to observe the new improved result.

VI. Result (First Part, Wikipedia Dataset)

The outcome of the project analysis was very satisfactory. We have analyzed each classifier Accuracy and F1 score for attack, aggression, toxicity, combination of all three types together and result of two concatenation of feature. Below we will discuss each of the analysis and findings.

Table 2, shows the classifier accuracy result during the 'Attack' analysis. The best accuracy result .95 obtain by the **CNN Word Embedding** and the lowest accuracy result obtain .88 **Naive Bayes Count Vectors**. The best F1 score obtain from **CNN Word Embedding** .948. After analysis of all classifier feature for Attack, we come to conclusion that **CNN Word Embedding** performance was better.

Classifier Name	**Accuray Score**	**F1 Score**
Naive Bayes Count Vectors	**0.880**	**0.874**
Naive Bayes Word Level TF-IDF	0.935	0.927
Naive Bayes N-Gram Vectors	0.898	0.871
Naive Bayes CharLevel Vector	0.926	0.914
Linear Classifier Count Vectors	0.888	0.841
Linear Classifier Word Level TF-IDF	0.945	0.941
Linear Classifier N-Gram Vectors	0.899	0.870
Linear Classifier Char Level Vector	0.945	0.940
CNN Word Embedding	**0.950**	**0.948**
NN, Count Vector	0.902	0.874
NN, Word Level	0.947	0.945
NN, N-gram	0.900	0.877
NN, CharLevel Vectors	0.946	0.943
LSTM, Word Embeddings	0.946	0.942
SVM Count Vectors	0.899	0.868
SVM Word-Level TF-IDF	0.884	0.8305
SVM N-Gram Vectors	0.884	0.8305
SVM CharLevel Vectors	0.884	0.8305

Table 2: Classier f1, and accuracy score for Attack

Below, Table 3 shows the classifier accuracy and f1 score after concatenation of two features during the 'Attack' analysis. After concatenation, we have observed, some result of accuracy and f1 score has improved very little (approximate 1%) ; however, some results have decreased as well.

	Naive Bayes		Linear Classifier		SVM classifier	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count and LIWC	0.880	0.874	0.888	0.841	0.892	0.8496
Count and TF-IDF	0.906	0.907	0.888	0.841	0.885	0.8319
Count and Ngram	0.8896	0.885	0.888	0.841	0.885	0.8319
Count and Ngram Chars	0.886	0.890	0.888	0.840	0.892	0.8496
TFIDF and LIWC	0.935	0.927	0.945	0.941	0.884	0.8305
Ngram and LIWC	0.8985	0.871	0.899	0.870		
Ngram Chars and LIWC	0.9266	0.914	0.9456	0.940		
TFIDF and Ngram	0.933790	0.929	0.945	0.940		
tfidf_ngram_char	0.93780	0.936	0.950	0.947		
ngram_ngram_char	0.92537	0.919	0.945	0.941		

	CNNs		LSTMs	
Features	Accuracy	F1 score	Accuracy	F1 score
liwc embedding	0.947	0.946	0.949	0.946
count Embedding	0.891	0.851	0.892	0.849
TFIDF Embedding	0.884	0.830	0.884	0.830
TFIDF ngram Embedding	0.884	0.830	0.884	0.830
TFIDF ngram chars Embedding	0.884	0.830	0.884	0.830

Table 3: Classier f1, and accuracy score after concatenation of two features for Attack

Table 4, shows the classifier accuracy result during the 'Aggression' analysis. The best accuracy result 0.939 obtain by the **LSTM, Word Embeddings** and the lowest accuracy result obtain .86 **Naive Bayes Count Vectors.** The best F1 score obtained from **CNN Word Embedding and NN, Word Level** 0.935 and **LSTM** perform almost similar 0.933 . After analysis of all classifier for Aggression, we come to the conclusion that **LSTM, Word Embeddings** accuracy performance was better.

Classifier Name	**Accuray Score**	**F1 Score**
Naive Bayes Count Vectors	**0.864**	0.859
Naive Bayes Word Level TF-IDF	0.927	0.917
Naive Bayes N-Gram Vectors	0.889	0.861
Naive Bayes CharLevel Vector	0.918	0.905
Linear Classifier Count Vectors	0.877	0.8261
Linear Classifier Word Level TF-IDF	0.937	0.932
Linear Classifier N-Gram Vectors	0.889	0.858
Linear Classifier Char Level Vector	.935	0.929
CNN Word Embedding	0.938	**0.935**
NN, Count Vector	0.885	0.866
NN, Word Level	0.938	**0.935**
NN, N-gram	0.889	0.863
NN, CharLevel Vectors	0.936	.933
LSTM, Word Embeddings	**0.939**	**0.933**
SVM Count Vectors	0.880	0.815
SVM Word-Level TF-IDF	0.873	0.815
SVM N-Gram Vectors	0.873	0.815

Table 4: accuracy and f1 for Aggression

Below Table 5, shows the classifier accuracy and f1 score after concatenation of two features during the 'Aggression' analysis. After some concatenation, we have observed, some result of accuracy and f1 score has improved very little (approximate 1%) ; however, some result has decreased as well.

	Naive Bayes		Linear Classifier		SVM classifier	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count and LIWC	0.885	0.8817	0.905	0.864	0.880	0.831
Count and TF-IDF	0.913	0.9161	0.905	0.865	0.873	0.863
Count and Ngram	0.894	0.893	0.905	0.865	0.873	0.862
Count and Ngram Chars	0.883	0.89	0.95	0.952	0.87	0.852
TFIDF and LIWC	0.946	0.939	0.912	0.884	0.912	0.814
Ngram and LIWC	0.912	0.888	0.953	0.949	0.882	0.892
Ngram Chars and LIWC	0.936	0.92	0.956	0.952	0.919	0.889
TFIDF and Ngram	0.944	0.940	0.960	0.957	0.90	0.880
tfidf_ngram_char	0.948	0.945	0.953	0.949	0.873	0.814

ngram_ngram_char	0.936	0.930	0.905	0.864	0.873	0.814

Features	Accuracy	F1 score	Accuracy	F1 score
liwc embedding	0.938	0.933	0.945	0.919
count Embedding	0.880	0.839	0.945	0.919
TFIDF Embedding	0.873	0.814	0.945	0.919
TFIDF ngram Embedding	0.873	0.814	0.945	0.919
TFIDF ngram chars Embedding	0.873	0.814	0.945	0.919

Table 5: Classier f1, and accuracy score after concatenation of two feature for Aggression

Below Table 6, shows the classifier accuracy result during the 'Toxicity' analysis. The best accuracy result **0.959** obtain by the **LSTM, Word Embeddings** and the lowest accuracy result obtain 0.88 by **Naive Bayes Count Vectors.** The best F1 score obtained from **CNN Word Embedding and NN, Word Level** 0.956. After analysis of all classifier feature for Toxicity, we come to conclusion that **LSTM, Word Embeddings** features accuracy performance was better.

Classifier Name	Accuray Score	F1 Score
Naive Bayes Count Vectors	**0.885**	0.881
Naive Bayes Word Level TF-IDF	0.946	0.939
Naive Bayes N-Gram Vectors	0.912	0.888
Naive Bayes CharLevel Vector	0.936	0.924
Linear Classifier Count Vectors	0.905	0.864
Linear Classifier Word Level TF-IDF	0.956	0.952
Linear Classifier N-Gram Vectors	0.913	0.885
Linear Classifier Char Level Vector	0.954	0.949
CNN Word Embedding	**0.958**	**0.956**
NN, Count Vector	0.913	0.883
NN, Word Level	0.956	0.953
NN, N-gram	0.912	0.886
NN, CharLevel Vectors	0.954	0.950
LSTM, Word Embeddings	**0.959**	0.942
SVM Count Vectors	0.913	0.884
SVM Word-Level TF-IDF	0.902	0.856
SVM N-Gram Vectors	0.902	0.856
SVM CharLevel Vectors	0.902	0.856

Table 6: Classier f1 and accuracy for Toxicity

Below Table 7, shows the classifier accuracy and f1 score after concatenation of two features during the 'Toxicity' analysis. After some concatenation, we have observed, some result of accuracy and f1 score has improved very little (approximate **1%**) ; however, some results have decreased as well.

	Naive Bayes		Linear Classifier		SVM classifier	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count and LIWC	0.864	0.859	0.878	0.826	0.880	0.831
Count and TF-IDF	0.892	0.894	0.826	0.826	0.885	0.831
Count and Ngram	0.875	0.873	0.877	0.826	0.885	0.831
Count and Ngram Chars	0.869	0.876	0.877	0.824	0.884	0.830
TFIDF and LIWC	0.927	0.917	0.937	0.932		
Ngram and LIWC	0.889	0.861	0.889	0.858		
Ngram Chars and LIWC	0.918	0.905	0.935	0.929		
TFIDF and Ngram	0.926	0.921	0.937	0.932		
tfidf_ngram_char	0.929	0.926	0.941	0.937		
ngram_ngram_char	0.918	0.912	0.936	0.930		

	CNNs		LSTMs	
Features	Accuracy	F1 score	Accuracy	F1 score
liwc embedding	0.9455	0.945	0.919	0.919
count Embedding	0.945	0.945	0.919	0.919
TFIDF Embedding	0.945	0.945	0.919	0.919
TFIDF ngram Embedding	0.945	0.945	0.919	0.919
TFIDF ngram chars Embedding	0.945	0.945	0.919	0.919

Table 7: Classier f1, and accuracy score after concatenation of two feature for Toxicity

Below Table 8 shows the classifier accuracy result during all three (Attack , Aggression and Toxicity) analysis. The best accuracy result .90 obtained by CNN, NN and LSTM. Here, Linear and NB Word Level TF-IDF

perform almost the same as CNN, NN and LSTM. However, CNN, LSTM and NN were very slow compared to Linear and NB model.

Classifier Name	**F1 Score**	**Accuray Score**
Naive Bayes Count Vectors	.84	0.86
Naive Bayes Word Level TF-IDF	.86	0.89
Naive Bayes N-Gram Vectors	.84	0.88
Naive Bayes CharLevel Vector	.86	0.89
Linear Classifier Count Vectors	.83	0.88
Linear Classifier Word Level TF-IDF	.87	0.897
Linear Classifier N-Gram Vectors	.83	0.89
Linear Classifier Char Level Vector	.78	0.896
CNN Word Embedding	.88	0.90
NN, Count Vector	.88	0.90
NN, Word Level	0.88	0.90
NN, N-gram	0.88	0.90
LSTM, Word Embeddings	0.88	0.90

Table 8: Classier f1 and accuracy for simultaneous analysis of Attack, Aggression and Toxicity

After analysis of result for Attack, Aggression, Toxicity and all three together , we have experienced the best output from Linear Classifier with Word Level TF-IDF features. The best result was true for classifier Accuracy , F1 and LIWC score. Additionally, Linear Classifier Word Level TF-IDF has performed well compared to the time its takes to perform the calculations. In Our GUI we have implement Linear Classifier with Word Level TF-IDF as a selected classifier.

Effect of Adding Random sentences and Negation of Dataset

At the first stage in the preparation of the dataset we have created a dataset consisting of a random sentences generated from wikipedia website, and a negation of each of the given datasets. We added this dataset as a neutral dataset to the originally neutral 'Attack' dataset for testing. and below are the results.

	Naive Bayes		**Linear Classifier**	
Features	Accuracy	F1 score	Accuracy	F1 score
Count Vectors	0.9371 **+5%**	0.9332 **+6%**	0.9468 **+6%**	0.9216. **+4%**
Word-Level TF-IDF	0.9545. **+2%**	0.9444. **+2%**	0.9665 **+2%**	0.9627 **+2%**
N-Gram Vectors	0.9485. **+5%**	0.9361 **+6%**	0.9515 **+5%**	0.9352 **+5%**
CharLevel Vectors	0.9553. **+3%**	0.9466. **+3%**	**0.9668 +2%**	**0.9629 +2%**

VII. Part Two: Z. Zhang Dataset

A. Dataset Description

In this part, we use the dataset collected by Z. zhang et al. from the collection of publicly available English Twitter datasets. As Z Zhang reported, this is the largest set (in terms of tweets) of Twitter based dataset used in hate speech detection[6]. There are seven types of datasets which each dataset differs from the other by its topic and the level of annotating. However, in this part, our main concern is on the hate speech dataset which is named DT and that has binary labeling into 'hate' and 'non-hate' classes.
These were collected based on the principle of keyword or hashtag filtering from the public Twitter stream. DT consolidates the dataset by into two types, 'hate' and 'non-hate'. The dataset is originally labeled as 2 for 'non-hate' and 0 for 'hate'. for convenience, we have relabeled the dataset as 0 for 'non-hate' or 'neutral and 1 for 'hate', then we cleaned the text and removed the usernames.

B. Results of classification

The results of classification in using Z. Zhang dataset are not of much difference compared to the previous results with wikipedia dataset.
Starting with table 9 which shows the classifier accuracy and f1 score the 'hate-speech' dataset analysis.
Reading the table 9 below, we can observe from the results that the results are pretty much close to each other, denoting the best accuracy for the linear classifier through character level vector features

	Naive Bayes		**Linear Classifier**		**SVM classifier**	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count Vectors	0.8384	0.8666	0.9402	0.9129	0.9414	0.9131
Word-Level TF-IDF	0.9414	0.9131	0.9410	0.9221	0.9414	0.9131
N-Gram Vectors	0.9398	0.9138	0.9419	0.9141	0.9414	0.9131
CharLevel Vectors	0.9392	0.9138	**0.9421**	**0.9236**	0.9414	0.9131

Table 9: accuracy and F1 scores for different classifiers with different features from Z. Zhang dataset.

Now the second part using the different types of neural networks for the classification task which involves FeedForward neural Networks Convolutional neural networks CNNs and Long short-term memory or LSTMs. table 10 shows the results briefly.

	Feed Forward NNs

Features	Accuracy	F1 score
Count Vectors	0.9408	0.9132
Word-Level TF-IDF	**0.9414**	**0.9131**
N-Gram Vectors	**0.9414**	**0.9131**
CharLevel Vectors	0.9404	0.9130

	CNNs		LSTMs	
Features	Accuracy	F1 score	Accuracy	F1 score
Word Embeddings	**0.9427**	**0.9338**	0.9402	0.9174

Table 10: accuracy and F1 scores for different Neural networks with different features from Z. Zhang dataset.

from the table above, we observe that the highest score overall is using the CNNs with word embedding features at 94.27% and F1 score at 93.38%.

Features Concatenation results

By concatenating Features we hope to see some improvements on the classification scores. Below is the table including the results of different concatenation with different classifiers. We deduce the highest accuracy score is from concatenating N-gram feature and LIWC feature by 94.67% with less than **1%** improvement. and the best F1 score is from concatenating Count and TF-IDF features at 93.15% with almost **2%** improvement.

	Naive Bayes		**Linear Classifier**		**SVM classifier** (gamma=auto/scale)	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count and LIWC	0.9455	0.9190	0.9398	0.9134	0.9414/0.9455	0.9131/0.9190
Count and TF-IDF	0.9459	**0.9315**	0.9416	0.9239	0.9414/0.9455	0.9131/0.9190
Count and Ngram	0.9308	0.9161	0.9402	0.9136	0.9414/0.9455	0.9131/0.9190
Count and Ngram Chars	0.9354	0.9321	0.9423	0.9245	0.9414/0.9455	0.9131/0.9190
TFIDF and LIWC	0.9455	0.9190	0.9408	0.9217	0.9414/0.9455	0.9131/0.9190
Ngram and LIWC	0.9419	0.9176	0.9419	0.9141	0.9414/0.9455	0.9131/0.9190
Ngram Chars and LIWC	**0.9467**	0.9251	0.9427	0.9248	0.9414/0.9455	0.9131/0.9190
TFIDF and Ngram	0.9414	0.9181	0.9423	0.9240	0.9414/0.9455	0.9131/0.9190
tfidf_ngram_char	0.9459	0.9274	0.9421	0.9254	0.9414/0.9455	0.9131/0.9190
ngram_ngram_char	0.9427	0.9227	0.9423	0.9245	0.9414/0.9455	0.9131/0.9190

Table 11: accuracy and F1 scores for different classifiers with different **concatenated** features from Z. Zhang dataset.

Next is to use the Neural networks results. from the table down below we notice the results are pretty much close to each other and the highest score was from concatenating the Count feature and TF-IDF feature with accuracy of 94.59% and F1 score of 92.04%.

	Feed Forward NNs	
Features	Accuracy	F1 score
Count and LIWC	0.9455	0.9190
Count and TF-IDF	**0.9459**	**0.9204**
Count and Ngram	0.9455	0.9190
Count and Ngram Chars	0.9457	0.9195
TFIDF and LIWC	0.9455	0.9190
Ngram and LIWC	0.9455	0.9190
Ngram Chars and LIWC	0.9455	0.9190
TFIDF and Ngram	0.9455	0.9190
tfidf_ngram_char	0.9455	0.9190
ngram_ngram_char	0.9455	0.9190

	CNNs		**LSTMs**	
Features	Accuracy	F1 score	Accuracy	F1 score
liwc embedding	0.9455	0.9190	0.9455	0.9190
count Embedding	0.9455	0.9190	0.9455	0.9190
TFIDF Embedding	0.9455	0.9190	0.9455	0.9190
TFIDF ngram Embedding	0.9455	0.9190	0.9455	0.9190
TFIDF ngram chars Embedding	0.9455	0.9190	0.9455	0.9190

Table 12: accuracy and F1 scores for different Neural networks with different **concatenated** features from Z. Zhang dataset.

Effect of Using Portion of Dataset

One question comes to mind is do we really need to use a huge amount of data and what if we use just a small portion of the data. We took a small part of the Z. zhang data and performed the previous tasks and reported the results.

	Naive Bayes		***Linear Classifier***		***SVM classifier***	
Features	*Accuracy*	*F1 score*	*Accuracy*	*F1 score*	*Accuracy*	*F1 score*
Count Vectors	*0.8467*	*0.8770*	*0.9465*	*0.9215*	*0.9475*	*0.9220*
Word-Level TF-IDF	*0.9475*	*0.9220*	*0.94758*	*0.9274*	*0.9475*	*0.9220*
N-Gram Vectors	*0.9475*	*0.9220*	*0.94758*	*0.9220*	*0.9475*	*0.9220*
CharLevel Vectors	*0.9475*	*0.9220*	*0.94858*	*0.9295*	*0.9475*	*0.9220*

	Feed Forward NNs	
Features	*Accuracy*	*F1 score*
Count Vectors	*0.9475*	*0.9220*
Word-Level TF-IDF	*0.9475*	*0.9220*
N-Gram Vectors	*0.9475*	*0.9220*
CharLevel Vectors	*0.9475*	*0.9220*

	CNNs		*LSTMs*	
Features	*Accuracy*	*F1 score*	*Accuracy*	*F1 score*
Word Embeddings	*0.9475*	*0.9220*	*0.9475*	*0.9220*

Table 13: accuracy and F1 scores for different classifiers with different features from small Z. Zhang dataset.

Comparing the results we can say that using the small portion of this dataset did not affect the results badly. The results are slightly higher than the original datasets in both accuracy and F1 score.

Effect of balancing the dataset

One of the most debated questions in data handling when using classification is do we have to have a balanced dataset or an unbalanced one. Actually it depends on the application itself as for this hate speech detection problem we know the fact that the hate speech in real life is not as much as the neutral 'non-hate' speech so it is better to keep the data in its unbalanced form so this would help for the classifier to act better real life examples as we will see in the results below.

	Naive Bayes		**Linear Classifier**		**SVM classifier**	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count Vectors	0.6923	0.6799	0.7342	0.7340	0.7307	0.7308
Word-Level TF-IDF	0.7814	0.7808	0.8409	0.8409	0.48426	0.3159
N-Gram Vectors	0.5734	0.5569	0.5821	0.5596	0.48426	0.3159
CharLevel Vectors	0.7744	0.7714	0.8391	0.8392	0.48426	0.3159

	Feed Forward NNs	
Features	Accuracy	F1 score
Count Vectors	0.6643	0.6635
Word-Level TF-IDF	0.8041	0.8035
N-Gram Vectors	0.5839	0.5767
CharLevel Vectors	0.8094	0.8092

	CNNs		**LSTMs**	
Features	Accuracy	F1 score	Accuracy	F1 score
Word Embeddings	0.7534	0.7533	0.75874	0.7579

Table 14: accuracy and F1 scores for different classifiers with different features after balancing Z. Zhang dataset.

The results as expected lower when using the balanced dataset than when using the original unbalanced dataset. Now we can see the effect of weighted SVM classification on this balanced dataset. We gave the weight of **10** for the neutral dataset over the 'hate' dataset and the results are below:

Weighted SVM Vs normal (weight for neutral=10)		
Features	Accuracy	F1 score
Count Vectors	0.7307/0.6888	0.7308/0.6831
Word-Level TF-IDF	0.4842/0.5157	0.3159/0.3509
N-Gram Vectors	0.4842/0.5157	0.3159/0.3509
CharLevel Vectors	0.4842/0.5157	0.3159/0.3509

Table 15: accuracy and F1 scores for weighted SVM classifier with different features after balancing Z. Zhang dataset.

From the results we can say that giving some weight to the neutral dataset would improve the score with around 3% improvement.

Transfer Learning

To assure that the model which has been trained on a given dataset is performing well it is common to make validation under another dataset collected from different sources but in the same type, this is what is called transfer learning. In this part we are going to use the model trained on the wikipedia 'Attack' and 'Aggression' datasets and make the validation and testing on the Z. Zhang dataset.

table below shows the results of the Transfer Learning of Z. Zhang by model trained on wikipedia 'Attack' dataset.

	Naive Bayes		**Linear Classifier**		**SVM classifier**	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count Vectors	0.8652	0.8849	0.9445	0.9193	0.9112	0.9085
Word-Level TF-IDF	0.4285	0.5535	0.5306	0.6500	0.9455	0.9190
N-Gram Vectors	0.9265	0.9132	0.9271	0.9135	0.9455	0.9190
CharLevel Vectors	0.6807	0.7678	0.6424	0.7396	0.9455	0.9190

	Feed Forward NNs	
Features	Accuracy	F1 score
Count Vectors	0.8997	0.90333
Word-Level TF-IDF	0.4077	0.53218
N-Gram Vectors	0.9202	0.91022
CharLevel Vectors	0.5292	0.64718

	CNNs		**LSTMs**	
Features	Accuracy	F1 score	Accuracy	F1 score
Word Embeddings	0.3730	0.4912	0.3484	0.4630

Table 16: accuracy and F1 scores of the Transfer Learning of Z. Zhang by model trained on wikipedia 'Attack' dataset.

The scores are low and high depending on the feature and classifier. If we talk about features then on average the lowest accuracy and F1 score were from Word-Level Tf-Idf feature, and the highest are from N-gram feature vectors and Count feature vectors. If we talk about the classifiers then the SVM classifier performed the best overall, and the CNNs and LSTMs were at lowest scores. Performing the

same for the "Aggression" dataset leads to the same conclusion.

Now let us see the results if we do the inverse, that means we try to train the model on the Z. Zhang dataset and then we test and validate on the wikipedia ''Attack' dataset.

table 17 below show the results obtained from the experiment.

	Naive Bayes		Linear Classifier		SVM classifier	
Features	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
Count Vectors	0.3662	0.4478	0.8766	0.8332	0.8846	0.8305
Word-Level TF-IDF	0.7869	0.4478	0.8860	0.8347	0.8846	0.8305
N-Gram Vectors	0.8614	0.8203	0.8846	0.8307	0.8846	0.8305
CharLevel Vectors	0.8602	0.8222	0.8865	0.8356	0.8846	0.8305

	Feed Forward NNs	
Features	Accuracy	F1 score
Count Vectors	0.8849	0.8312
Word-Level TF-IDF	0.8849	0.8313
N-Gram Vectors	0.8846	0.8305
CharLevel Vectors	0.8854	0.8326

	CNNs		LSTMs	
Features	Accuracy	F1 score	Accuracy	F1 score
Word Embeddings	0.8867	0.8368	0.8847	0.8437

Table 17: accuracy and F1 scores of the Transfer Learning of wikipedia 'Attack' by model trained on Z. Zhang dataset.

Overall the results are more satisfactory than the previous Transfer Learning of Z. Zhang by wikipedia dataset. If we talk in both cases, the feature vector 'N-gram' was more robust in both sides, and the classifier SVM was also more robust.

VIII. Conclusion

We have start this project with two (2) parts. First part, using the wikipedia dataset, for the sake of preparation we created a programmatically generated negation dataset from existing hate dataset and another dataset that also generated programmatically from wikipedia random sentences. In the feature engineering section we have used six (6) different type of features and various types of classifier including linear, naive bayes , svm, Neural Networks, CNNs and LSTMs. We have successfully implemented all classifications, and yet we used feature concatenation trick in order to see some improvements, we analyzed the results by comparing F1 score and accuracy score, and have used comparatively best classifier in the web based GUI. Besides that, we employed the dataset generated as neutral dataset and this has made some noticeable improvements. Second part we have employed the Z. zhang where we have done the same as for wikipedia dataset, and we have gone through other experiments such as observing the effect of using small dataset versus large dataset, where we found that it doesn't affect badly for this experiment. Besides of seeing the effect of balancing the dataset, which has affected the results pretty badly. At last, we have gone through the Transfer Learning experiment, where we used the two dataset Z. Zhang and wikipedia, one for training and the other for testing, and vice versa. We concluded that N-gram feature for different classifiers, and SVM classifier for different features are more robust for this work.

IX. Acknowledgement

We would like to thank Mr. Radouane Kaddari and Mr.Nijar Hosain and Md. Saroar Jahan for any help and support.

References

[1] https://github.com/abderraouf2che/Hate-Speech-Detector-Project/

[2] https://figshare.com/projects/Wikipedia_Talk/16731

[3] https://github.com/abderraouf2che/Hate-Speech-Detector-Project/blob/master/Classification_Pipeline/attack_using_pipeline.py

[4] https://s3-us-west-1.amazonaws.com/fasttext-vectors/wiki-news-300d-1M.vec.zip

[5] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[6] Zhang, Z., & Luo, L. (2018). Hate Speech Detection: A Solved Problem? The Challenging Case of Long Tail on Twitter. arXiv preprint arXiv:1803.03662.

[7] Schmidt, A., & Wiegand, M. (2017). A survey on hate speech detection using natural language processing. In Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media (pp. 1-10).

[8] Ben-David, A., & Matamoros-Fernandez, A. (2016). Hate speech and covert discrimination on social media: Monitoring the Facebook pages of extreme-right political parties in Spain. International Journal of Communication, 10, 1167-1193.

[9] https://www.djangoproject.com/

[10] https://www.w3schools.com/html/html_css.asp

[11] https://en.wikipedia.org/wiki/Special:Random

[12] https://github.com/Ejhfast/empath-client

[13] Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016, April). Abusive language detection in online user content. In Proceedings of the 25th international conference on world wide web (pp. 145-153). International World Wide Web Conferences Steering Committee.

[14] Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). Automated hate speech detection and the problem of offensive language. arXiv preprint arXiv:1703.04009.