Hate Speech Detection on Social Media

Gökhan Özeloğlu
Hacettepe University
b21627557@cs.hacettepe.edu.tr

Ege Çınar
Hacettepe University
b21627136@cs.hacettepe.edu.tr

Yiğit Barkın Ünal
Hacettepe University
b21627763@cs.hacettepe.edu.tr

Abstract

In this century, using social media is increasing year by year. This leads to some of the big problems for social media users. One of this problem is hate speech. Machine Learning is maybe one of the biggest weapons to prevent hateful speeches on social media platforms. In this paper, we introduce our research project which is detecting hate speech on social media. We built and tested our Machine Learning models on different datasets which are collected from Twitter. We introduced our model, which are ***Naïve Bayes, Logistic Regression, Support Vector Machines****, and* ***1D CNN****, and their accuracy results that we trained and tested on 4 different datasets.*

1. Introduction

Hate speech is a growing problem in relation with the growing user-base of the social media platforms such as Twitter, YouTube, Facebook, and Reddit. But first of all, what is hate speech? Hate speech is defined by Cambridge Dictionary as *"public speech that expresses hate or encourages violence towards a person or group based on something such as race, religion, sex, or sexual orientation"*. It effects millions of people every day and just because of this, people are getting depressed and tend to be suicidal. Hate speech occurs in everywhere such as social life and social media. Hate speech can scare away quality discussions which is something most social media platforms do not want. Hate speech can also increase polarization of different groups in society and lead to inter-group violence between different racial or political groups. The only efficient way to detect hate speech is using machine learning techniques which are already used by some social media platforms to some degree.

In this project we aim to detect content which include hate speech. We have focused on Twitter because millions of tweets are shared every day. We used datasets which have multi-class labels. So far, we used four differnet dataset which are multi-classes or two-labeled [5, 6, 7, 8]. We implemented **Naïve Bayes, Logistic Regression, Support Vector Machines(SVM)**, and **1D CNN** models. We trained and tested our models with all datasets. We are going to explain details in the following sections. Also we discussed experimental results and stated what we can be do in the future.

2. Related Work

There is a lot of research done in recent years in detecting and identifying types of hate speech. The articles generally concentrate on either binary classification of hate speech or multi-class classification. Dictionary based approaches fall short on detecting hate speech that is expressed in implicit terms. So corpus based approach is more suited to our project.

Shervin et. al. [1] applied supervised classification methods. Their data-set is Hate Speech Identification [9]. Their system uses *n-gram, word n-gram* and *word-skip grams*. They obtained 78% accuracy. Their data-set has three different classes, which are *hate, offensive*, and *no offensive content*. This detest is our data-set's older version which has 15K tweets.

Tulkens et. al. [4] used a dictionary based approach. They created their dictionary from the comments sections of the Facebook pages then they extended their dictionary by taking similar words to the ones already in their dictionary. They used word2vec to get the most similar words. After expansion they manually discarded obviously incorrect terms. The researchers did not get the accuracy they expected.

Salminen et. al. [2] created a multi-class classification which identified hate targets and theme of the hate speech. They first detect if a comments is hateful or not using a binary classifier. Then they use a multi-class classifier to de-

tect hate targets and the theme of the hate speech. They used different models in both learning algorithms and feature categories. The classifiers are logistic regression, decision tree, random forest, adaboost and support vector machine. The features are extracted using TF, TF-IDF, semantic features, and word2vec. The average precision of the best model, SVM, was 0.90.

Xu, et. al. [3] focused on bullying and they tried to identify actors in a bullying session such as victim, bully, accuser and reporter. They got their data from Twitter. The data they work on is a collection of tweets which have indication of bullying. They used part of speech tagging and named entity recognition. They detected the author of the tweets' roles using linear support vector machine with unigram plus bigram feature representation. The actors can have interchangeable roles so that is a challenging part of detecting the actors involved. They had a cross validation accuracy of 61 percent.

Lilleberg et. al. [10] first tried word2vec with stop words then compared it to TF-IDF without stop words. TF-IDF without stop words performed better than word2vec with stop words. Then they tried word2vec without stop words weighted by TF-IDF and word2vec with stop words. The former outscored the latter. TF-IDF without stop words did better than word2vec without stop words weighted by TF-IDF. Lastly they combined TF-IDF without stop words and word2vec without stop words weighted by TF-IDF and this performed better than TF-IDF without stop words.

The assumption is word2vec delivers extra semantic feaures because word2vec takes word positions into consideration when calculating the vectors. This is a new approach because most word2vec models don't involve TF-IDF. Their work uses skip-gram model by default which has highest semantic accuracy. They used scikit-learn's LinearSVC (Linear Support Vector Classification) to classify text classes. The combination returned best results most of the time but TF-IDF without stop words returned slightly better results in some cases.

Dinakar et. al. [11], classified text documents some of Machine Learning techniques in the project. They used the **YouTube** comments as dataset. They, firstly, applied preprocessing operations on the comments. Stop words and insignificant words, like usernames, abbreviations, were removed. They labelled data into three classes. These are **sexuality, race and culture**, and intelligence. They built **Naïve Bayes, Rule-based JRip, Tree-based J48**, and **SVM** classifiers. Then, they merged the data and made classification with multi-class classifiers. They obtained better results in binary classifiers. Rule-based JRip gives the best performance in terms of accuracy.

We have studied these research papers and came to the conclusion that we are going have a corpus based approach and we will implement different multi-class classifiers and incorporate feature extraction tools such as TF-IDF.

3. The Approach

3.1. Datasets

We used four different datasets in the project to make sure that our models work well. Each of them has different sizes and labels. We, firstly, cleaned and concatenated the texts before starting the project. All datasets were stored in **comma separated values(csv)** format. There were redundant strings in each tweet, like *hashtags, usernames, links, meaningless characters*. These could have badly affected the accuracy of the models. There are some examples about our preprocessing below. Also, we applied **cross-validation** to prevent wasting any data. We split the data into **10-fold** and trained each of them. Before applying cross-validation, we were splitting data by 70-30%. 70% of was used for training the model, 30% of was used for testing the model. Nevertheless, we were wasting more data in **test-set** approach.

Before: @delfuego @PendragonTarot TILL I'm the district attorney.

After: TILL I'm the district attorney.

Before: Aw there's nothing to cry about Lynn xx #MKR

After: Aw there's nothing to cry about Lynn xx

Before: RT @PatriceChienne: I honestly think Kat has mental health issues, #mkr

After: I honestly think Kat has mental health issues,

Before: @Dabiq_Warrior @P_Valerianus @Totenleserin We only need to count the Daesh corpses every day. http://t.co/FJBB496YiM

After: We only need to count the Daesh corpses every day

In the first dataset, which is Thomas Davidson dataset [5], we have three different classes. These are grouped as HATE SPEECH, OFFENSIVE LANGUAGE, and NEITHER. You can see the text counts of each classes in Table 1. To make a classification for each tweet, they used CrowdFlower users. The users coded each tweet to make the decision which label the tweet includes. You can see the distribution of classes in Table 1.

Class Name	Class Count
HATE	1430
OFFENSIVE	19190
NEITHER	4163
Total	24783

Table 1: Class Distribution of classes and tweets in dataset 1

In the second dataset, which is Aitor García Pablos dataset [6], has two different classes. The dataset is grouped as **hate** and **noHate**. You can see the text counts of each classes in Table 2.

Class Name	Class Count
hate	1197
noHate	9507
Total	10704

Table 2: Class Distribution of classes and tweets in dataset 2

As we can see that labels are not uniformly distributed. There are almost 1/9 ratio in terms of classes.

In the third dataset [7], we have three different labels which are **Racism, Sexism, None**. You can see the text counts of each classes in Table 3. We assumed that **racism** and **sexism** tweets as **hate speech** tweets.

Class Name	Class Count
Racism	1970
Sexism	3378
None	11559
Total	16907

Table 3: Class Distribution of classes and tweets in dataset 3

In the fourth dataset, which is ENCASE H2020 dataset [8], has almost **100K** tweets. Moreover, it has 4 different classes in total. These classes are **abusive, hateful, spam** and **normal**. But we dropped the **spam** tweets from dataset to get more accurate results. So, we decreased the class count from 4 to 3. You can see the text count. After reducing the text count to 3, we have made an assumption. *Abusive* and *Hateful* labels can turn into one class which is *Hate Speech*. After making this assumption, we have changed the label names into **1** as a hate speech, and **0** as a normal tweet.

Class Name	Class Count
Abusive	27037
Hateful	4948
Normal	53790
Spam	14024
Total	99799

Table 4: Class Distribution of classes and tweets in dataset 4

3.2. Naive Bayes

We used Naive Bayes model as baseline model. It simply calculates the probability of each words in given text and assigns the text to one of the classes. Also, Naive Bayes uses class priors. It was built by using **sklearn** in Python. Two different Naive Bayes model is created for datasets. **MultinomialNB** is created for dataset-1-3-4 [5, 7, 8]. **GaussianNB** is called for dataset-2 [6]. The reason is only dataset-2 is binary-class, the other ones are multi-class datasets. Text documents should be converted to numerical representations, it means *feature extraction*. We chose **Bag of Words model**, or **BoW** in short. It is called "*bag*", because the word order is not important in this model. Moreover, **Term Frequency-Inverse Document Frequency (TF-IDF)** was applied to each tweet to extract the features. It is a technique used to understand how important a word is in a document. It is reversely proportional to the number of documents in the corpus that word appears. In addition, we tested two different situations that stop words were included and excluded. In general, including or excluding stop words did not affect much. Effects of stop words are discussed in the next section.

Moreover, we tried **n-gram model** to see the affect of continuous sequence of words. Basically, *n-gram model* combines the n adjacent words and counts them as a unit in a given text. If *n* is 1, it just takes the words. If n is 2, it takes two words which are adjacent, and combines them as unit. It has special names if *n* is 1,2, or 3. If *n* is 1, **unigram**, *n* is 2, **bigram**, and *n* is 3, **trigram**. We have discussed the result comparatively.

3.3. Logistic Regression

Our second model is **Logistic Regression**. Logistic regression is another basic supervised Machine Learnnig model that classify the binary data. It tries to fit a line to

classify data correctly. It works on two-labelled data such as **Yes-No, True-False, New-Old**. As we mentioned Dataset section, we had four different datasets. Three of them are 3 or 4 labeled data, only one of them is two-labelled data. We made some assumptions to reduce the number of classes, but we discussed all assumptions on the following section, Experimental Results.

Moreover, we used **sklearn** library to build Logistic Regression. To extract features from text documents, we used **Bag-of-Words (BoW) model**, **n-gram model**, and **TF-IDF model**. Same procedures that applied in Naive Bayes were applied into each tweets. Also, stop words were included and excluded in training/testing. Finally, we found the best appropriate **C parameter**. It is a tune parameter which determines the strength of the regularization. The higher C values are corresponding to less regularization, and the smaller values specify strong regularization. We started **C** at 1, and it finished at 20. After all we got distinct accuracy results by changing **C** value, including and excluding stop words, and trying different n-grams.

3.4. Support Vector Machine

In this section, we simply trained the SVM model from the **sklearn** library on our all datasets. SVM is a supervised learning algorithm that maps the data as points into space. It tries to make the gap between different categories as much as possible. The new data to be classified is added to the same space and predicted according to the place they are mapped on the space.

We used **TF-IDF, Bag-of-Words**, and **n-gram** model in the feature extraction phase. We also included and excluded stop words here as well. We tested *RBF, poly, Sigmoid*, and *linear* kernels. We decided to use the linear kernel after seeing it had performed the best. We also choose the one-versus-rest approach to handle multi-class classification. We tested different C values in this part too. We tried all C values from 1 to 20 and used the C value that gave the best results.

3.5. 1-D CNN

In this section, we simply trained the 1-D CNN model from the **keras** library on our all datasets. 1-D CNN is one of the fundamental machine learning algorithms. Even though it is used mostly on computer vision problems, 1-D CNN can be used for text classification. Different than the classical 2-D CNN, 1-D CNN gets a patch of words in the convolution layer with shifting only horizontally, not vertically.

We have used **max-pooling** for the pooling layer and calculated the loss using **binary cross-entropy** in our algorithm. To achieve the minimum loss, we have changed different attributes in the algorithm, such as **batch size, epoch size, activation functions**.

4. Experimental Results

We are going to give some details about our experiments and results. We choose one dataset for each model and gave accuracy results. Also, in Figure 1 and 2, we shared all results for each dataset.

Dataset	Model Name	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Dataset-1	Naïve Bayes	0.8005	0.8034	0.7760	0.7751	0.7750	0.7748
Dataset-1	Logistic Regression	0.9439	0.8516	0.8348	0.8340	0.8337	-
Dataset-1	SVM	0.8889	0.8089	0.7800	0.7766	0.7762	-
Dataset-2	Naïve Bayes	0.5992	0.4405	-	-	-	-
Dataset-2	Logistic Regression	0.8997	0.8916	0.8913	0.8913	0.8913	-
Dataset-2	SVM	0.8978	0.8913	0.8913	0.8913	0.8913	-
Dataset-3	Naïve Bayes	0.7296	0.7661	0.7400	0.7171	0.6872	0.6836
Dataset-3	Logistic Regression	0.8173	0.7728	0.7405	0.7100	0.6898	-
Dataset-3	SVM	0.8188	0.7752	0.7392	0.7124	0.6905	-
Dataset-4	Naïve Bayes	0.7306	0.7178	0.6557	0.6382	0.6334	0.6331
Dataset-4	Logistic Regression	0.7840	0.7481	0.6845	0.6715	0.6667	-
Dataset-4	SVM	0.62154	0.6164	0.5983	0.5772	0.5811	-

Figure 1: Accuracy Results - Including Stop Words

Dataset	Model Name	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
Dataset - 1	Naïve Bayes	0.8102	0.7861	0.7750	0.7748	0.7743	0.7743
Dataset - 1	Logistic Regression	0.9453	0.8435	0.8342	0.8337	0.8327	-
Dataset - 1	SVM	0.8887	0.7961	0.7774	0.7760	0.7752	-
Dataset - 2	Naïve Bayes	0.5943	0.0947	-	-	-	-
Dataset - 2	Logistic Regression	0.8961	0.8911	0.8913	0.8913	0.8913	-
Dataset - 2	SVM	0.8952	0.8911	0.8913	0.8913	0.8913	-
Dataset - 3	Naïve Bayes	0.7527	0.7360	0.6885	0.6836	0.6836	0.6836
Dataset - 3	Logistic Regression	0.8159	0.7377	0.6936	0.6879	0.6867	-
Dataset - 3	SVM	0.8138	0.7398	0.6949	0.6893	0.6873	-
Dataset - 4	Naïve Bayes	0.7333	0.7016	0.6388	0.6309	0.6270	0.6259
Dataset - 4	Logistic Regression	0.7859	0.7326	0.6696	0.6642	0.6608	-
Dataset - 4	SVM	0.6190	0.5986	0.5751	0.5722	0.5620	-

Figure 2: Accuracy Results Excluding Stop Words

4.1. Naive Bayes

We used Multinomial Naïve Bayes model for datasets 1, 3, and 4. For dataset 2 we used Gaussian Naïve Bayes because the dataset had binary labels. After using TF-IDF for feature extraction we tested from 1-gram to 6-gram. We tested the datasets both including and excluding stop words. We used **10-fold cross-validation** while training and testing. We had memory problems and we ignored words whose documentation frequencies are lower than 0.0005.

In dataset-4 we had the best results including stop words with bi-gram and we had the best results excluding stop words with uni-gram. We couldn't test dataset 2 more than bi-gram because the dataset included tweets shorter than three words. All accuracy results can be seen in Table 5. Figures 3 and 4 shows change in accuracy with respect to change in n-gram values.

Model Name	Stop word	Accuracy
Naïve Bayes 1-gram	Included	0.7102
Naïve Bayes 1-gram	**Excluded**	**0.7342**
Naïve Bayes 2-gram	**Included**	**0.7273**
Naïve Bayes 2-gram	Excluded	0.6943
Naïve Bayes 3-gram	Included	0.6537
Naïve Bayes 3-gram	Excluded	0.6340
Naïve Bayes 4-gram	Included	0.6345
Naïve Bayes 4-gram	Excluded	0.6247
Naïve Bayes 5-gram	Included	0.6300
Naïve Bayes 5-gram	Excluded	0.6197
Naïve Bayes 6-gram	Included	0.6277
Naïve Bayes 6-gram	Excluded	0.6164

Table 5: Naïve Bayes Accuracy results - Dataset-4

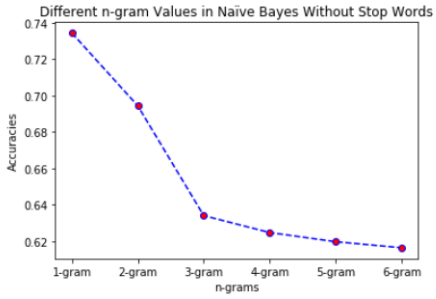

Figure 3: Naïve Bayes accuracy values without stop words (Dataset-4)

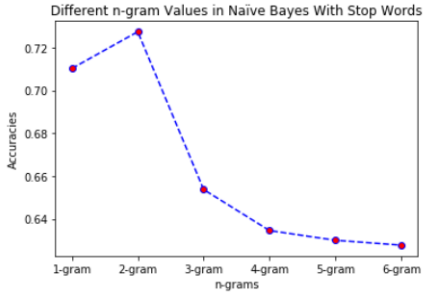

Figure 4: Naïve Bayes accuracy values with stop words (Dataset-4)

4.2. Logistic Regression

In Logistic Regression model, we trained and tested each dataset. Three of them are multi-class data[5, 7, 8], only one of them are binary-class data [6]. So, we made some assumptions in multi-class data to reduce class count into two. Also, we extracted text features by using **Bag-of-Words, TF-IDF** and **n-gram**. Actually, we applied the same procedures as in other models.

In dataset-3 [7], we firstly made an assumption over classes. We had 3 different classes, which were SEXISM, RACISM, and NONE. Our assumption is considering SEXISM and RACISM classes as HATE SPEECH. We found optimal **C** parameter for our model. It is a regularization parameter that shows us to strength of the regularization. Small **C** values specify the stronger regularization. To find this parameter, we started to **C** value from 1 and it went until 20. As you can see on Figure 5, we got the highest accuracy for $C = 4$. So, we declared C value as 4 in Logistic Regression models for dataset-3. We did not extend C values until 50 or 100, because after 4, the line is showing a downward trend.

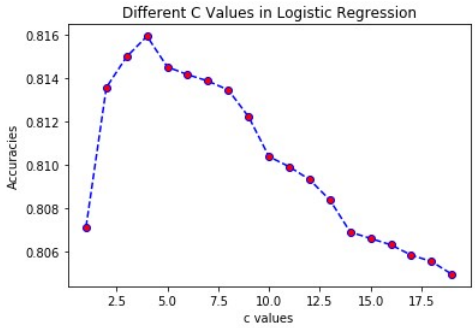

Figure 5: C Values in Logistis Regression (Dataset-3)

After finding **C**, we trained and tested our model until **5-gram** by including and excluding stop words. As you can see in Figure 6 and Figure 7, after **2-gram** learning process is almost done and goes almost steady. You can find accuracy values in Table 6.

Also, accuracy values were decreased drastically after 1-gram model. It might be happened because of loosing meaningful features after joining more than 1 words in text documents. We got the highest accuracy in **1-gram model** when stop words were excluded. So, excluding stop words were affected well over our model's accuracy.

4.3. Support Vector Machine

In Support Vector Machine(SVM) model, we trained and tested each dataset. As we did other models, we used **n-gram, Bag-of-Words, TF-IDF models** to extract features. Also, we split the dataset by using **10-fold cross-validation** to not waste the data. Firstly, we determined C value for

Model Name	Stop word	Accuracy
Logistic Regression 1-gram	**Included**	**0.8173**
Logistic Regression 1-gram	**Excluded**	**0.8159**
Logistic Regression 2-gram	Included	0.7728
Logistic Regression 2-gram	Excluded	0.7377
Logistic Regression 3-gram	Included	0.7405
Logistic Regression 3-gram	Excluded	0.6936
Logistic Regression 4-gram	Included	0.7100
Logistic Regression 4-gram	Excluded	0.6879
Logistic Regression 5-gram	Included	0.6898
Logistic Regression 5-gram	Excluded	0.6867

Table 6: Logistic Regression Accuracy results - Dataset-3

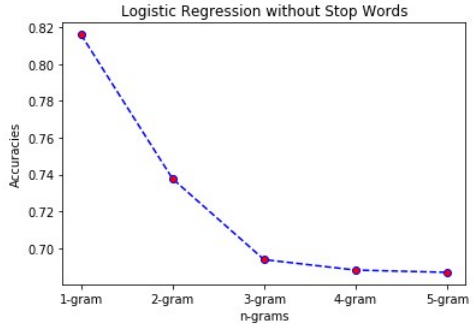

Figure 6: Logistic Regression accuracy values without stop words (Dataset-3)

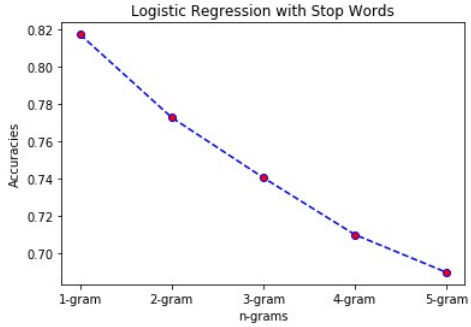

Figure 7: Logistic Regression accuracy values with stop words (Dataset-3)

each dataset by training and testing dataset. Our C value range was between 1 to 20. After that we tried four different kernel functions, which are *linear, polynomial, rbf*,and *Sigmoid*.

In dataset-1 [5], we had 3-labelled dataset. Labels are grouped as HATE, OFFENSIVE and NEITHER. We used *one-vs-rest scheme* to classify text documents, because dataset-1 is a multi-class dataset. Firstly, we determined C value by trying different values in 1-20 range. As you can see in Figure 8, we reached out the highest accuracy in $C = 1$. So, we continued our experiments with this value.

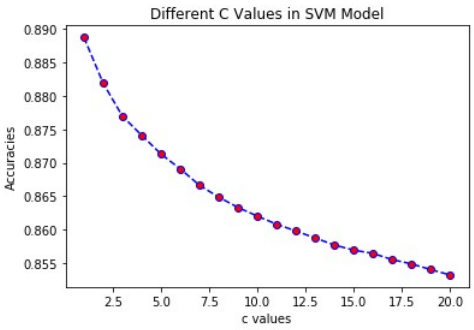

Figure 8: C Values in SVM (Dataset-1)

Then, we tried different kernel functions. You can see accuracy results in Table 7.

Kernel Function	Accuracy
Linear	0.9004
Polynomial	0.7905
Radial Basis Function (rbf)	0.8969
Sigmoid	0.9000

Table 7: Different Kernel Functions - Dataset-1

Finally, we trained and tested our dataset with different **n-grams** and **stop words**. You can see the all values in Table 8.

Model Name	Stop word	Accuracy
SVM 1-gram	**Included**	**0.8889**
SVM 1-gram	**Excluded**	**0.8887**
SVM 2-gram	Included	0.8089
SVM 2-gram	Excluded	0.7961
SVM 3-gram	Included	0.7800
SVM 3-gram	Excluded	0.7774
SVM 4-gram	Included	0.7766
SVM 4-gram	Excluded	0.7760
SVM 5-gram	Included	0.7762
SVM 5-gram	Excluded	0.7752

Table 8: SVM Accuracy results - Dataset-1

After some point accuracy almost did not change and continued steadily. Overfitting may be occured and learning process was terminated. As you can see in Figure 9-10, line would be straight after 2-gram for including stop word models. Nevertheless, choosing **1-gram** model would be the best choose for our dataset.

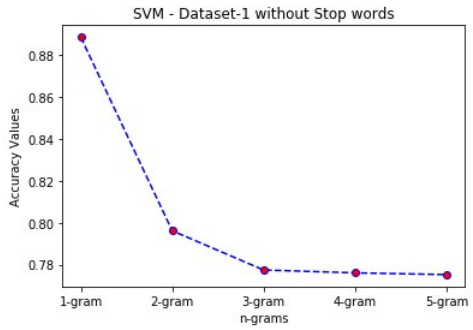

Figure 9: SVM accuracy values without stop words (Dataset-1)

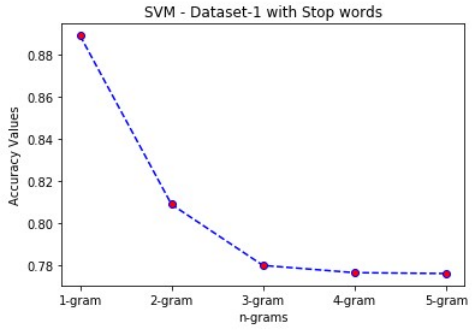

Figure 10: SVM accuracy values with stop words (Dataset-1)

4.4. 1-D CNN

In 1-D CNN model, we trained and tested 2 datasets but we focused on dataset-2. We used CountVectorizer to transform the text into vectors by converting all characters to lowercase and removing stop words. In dataset-2, there were 2 labels: **hate** and **noHate** and we converted them into **1** as **hate** and **0** as **noHate**. Then we split the data into pad sequences with length 400 words using **Keras**. In the convolution layer, we used **ReLu** as an activation function with 1 stride. In the pooling layer, we used **max pooling**. After doing that, we added an **Embedding Layer** and **Dropout Layer** into our neural network. **Embedding Layer** turns positive integers into fixed-size dense vectors and the **Dropout Layer** basically prevents our model from overfitting.

In the first try, we used ReLu and Sigmoid as activation functions in our model and we tried **20 epochs** with **32 batch sizes**. The lowest validation loss was with 0.3239 at 5^{th} epoch. You can see the Figure 11.

In the plot above, blue line represents training loss and red line represents validation loss.

We found the early stop at the 5^{th} epoch with 32 batch size and ReLu - Sigmoid activation functions but we tried

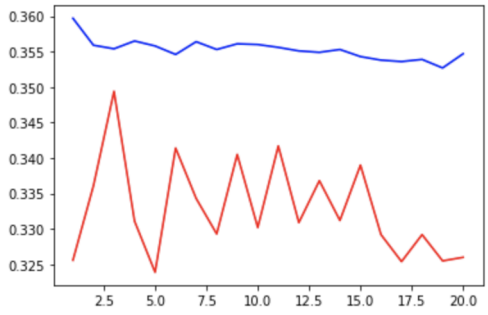

Figure 11: 1-D CNN accuracies using ReLu - Sigmoid with 20 epochs (Dataset-2)

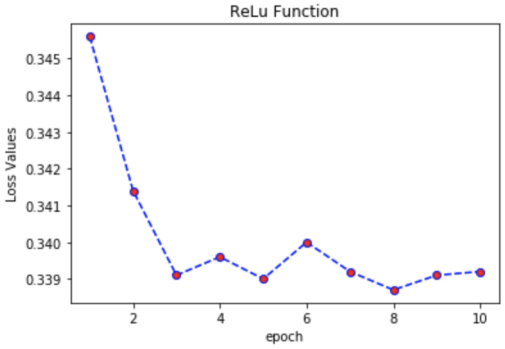

Figure 12: 1-D CNN accuracies using ReLu - ReLu with 10 epochs (Dataset-2)

10 epochs when we changed the attributes because early stop does not have to be the same as before.

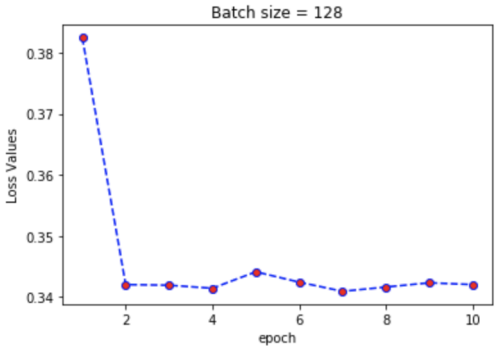

Figure 13: 1-D CNN accuracies using ReLu - Sigmoid with 10 epochs and 128 batch size (Dataset-2)

Then, we tried to change the batch size to get optimum value. We tried 16, 64, 128 batch sizes and tried to get the minimum loss. We found the minimum loss with 32 batch size. You can see the results of 128 batch size in Figure 13.

Then, we changed the activation functions to ReLu -

ReLu, Sigmoid - Sigmoid, and Tanh - Tanh. We obtained the lowest validation loss with ReLu - Sigmoid. You can see the Figure 11 and 12.

Our best-obtained result was **0.3239 validation loss** and **0.9004 validation accuracy** with ReLu - Sigmoid activation functions, with 32 batch size and at 5^{th} epoch. You can see the Figure 11.

5. Conclusions

In this research project, we tried to detect hateful speech text documents by using different Machine Learning methods. We used four different datasets and built four different models. Datasets were collected from Twitter. Before starting to build models, we applied some preprocess operations on the data. We removed some of noisy text like hashtags, usernames, emojis, links. Then, we built our models and trained them with each dataset. Also, we used different feature extraction methods such as **TF-IDF, n-gram, Bag-of-Words**. We experimented with each of them's impact on datasets. Finally, we took into account the effect of stop words by including and excluding them. **n-gram** experiments were done in *1-gram-5-gram* range for Logistic Regression and SVM, *1-gram-6-gram* for Naïve Bayes. In Logistic Regression model, we made an assumption for multi-label datasets. We reduced the number of classes from 3 to 2. We considered similar classes in one class.

Also, our research project can be extended by adding new models, or applying some other processes. For instance, tweets may be represented as vectors using **doc2vec** in feature extraction part. In the data preprocessing part, emojis' meaning can be put into their places instead of removing them. Moreover, hashtags can be replaced with a clear version of them. Putting emojis' meaning and clean hashtags may increase the accuracy of the models. Due to the origin of our dataset, there can be lots of typos, and these typos can be fixed before training models. Also, Twitter users can be used some different words from other languages in their tweets. So, words from different languages can be added to the dictionary.

References

[1] Shervin Malmasi and Marcos Zampieri *Detecting Hate Speech in Social Media*. arXiv:1712.06427 [cs.CL]

[2] Joni Salminen, Hind Almerekhi, Milica Milenković, Soon-gyo Jung, Jisun An, Haewoon Kwak, and Bernard J. Jansen *Anatomy of Online Hate: Developing a Taxonomy and Machine Learning Models for Identifying and Classifying Hate in Online News Media*

[3] Jun-Ming Xu, Kwang-Sung Jun, Xiaojin Zhu, and Amy Bellmore *Learning from Bullying Traces in Social Media* In HLT-NAACL, ACL (2012), 656–666.

[4] Stéphan Tulkens, Lisa Hilte, Elise Lodewyckx, Ben Verhoeven, Walter Daelemans *A Dictionary-based Approach to Racism Detection in Dutch Social Media* arXiv:1608.08738v1 [cs.CL]

[5] Dataset1 which we used to evaluate our model. [Online]. Avaliable: https://github.com/t-davidson/hate-speech-and-offensive-language.

[6] Dataset2 which we are going to use to evaluate our model. [Online]. Avaliable: https://github.com/aitor-garcia-p/hate-speech-dataset.

[7] Dataset3 which we are goint to use to evaluate our model. [Online]. Avaliable: https://github.com/wvs2/data-hate/tree/master/wassen.

[8] Dataset4 which we are going to use to evaluate our model. [Online]. Avaliable: https://github.com/ENCASEH2020/hatespeech-twitter.

[9] Hate Speech Identification. [Online]. Avaliable: https://data.world/crowdflower/hate-speech-identification.

[10] Joseph Lilleberg, Yun Zhu, Yanqing Zhang [] *Support Vector Machines and Word2vec for Text Classification with Semantic Features* 2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), Beijing, 2015, pp. 136-140.

[11] Karthik Dinakar and Roi Reichart and Henry Lieberman, *Modeling the Detection of Textual Cyberbullying*, The Social Mobile Web, 2011