NATIONAL AND KAPODESTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSC THESIS

Detecting Hate Speech Online using Machine Learning

Antonios N. Danezis

Supervisors: **Alexios Delis,** Professor NKUA

ATHENS

SEPTEMBER 2020

ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ

Αναγνώριση Ρητορικής Μίσους στο Διαδίκτυο με χρήση Μηχανικής Μάθησης

Αντώνιος Ν. Δανέζης

Επιβλέποντες: **Αλέξιος Δελής**, Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2020

BSC THESIS

Detecting Hate Speech Online using Machine Learning

Antonios N. Danezis
S.N.: 1115201300033

SUPERVISORS: **Alexios Delis,** Professor NKUA

ΠΤΥΧΙΑΚΗ

Αναγνώριση Ρητορικής Μίσους στο Διαδίκτυο με χρήση Μηχανικής Μάθησης

Αντώνιος Ν. Δανέζης
Α.Μ.: 1115201300033

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αλέξιος Δελής**, Καθηγητής ΕΚΠΑ

CONTENTS

FIGURES LIST

ABSTRACT

For the past two decades we've witnessed rapid growth of the internet as a platform to grow communities. Large online communities have formed on social media, forums, broadcasting platforms, live chats and online video games. However, with all this rise in popularity of these platforms, so has the difficulty of moderating them. In this paper, we'll propose several deep learning models trained on a toxic comment dataset, evaluate and compare them. Due to the nature of our dataset our models are mostly aimed for use in forums and social media where the message length is longer. We will study the training dataset and discuss its issues. Finally, we will demonstrate different preprocessing techniques and decide which ones are beneficial to our models and which are detrimental.

1. INTRODUCTION

1.1 Background

In recent years, there has been a steep increase in the userbase of online platforms such as social media, forums, broadcasting platforms, live chats and online video games. According to a 2020 report by DataReportal [1], the number of internet users around the world has grown by 7% since 2019, reaching 4.54 billion. Out of those, 3.80 billion are social media users whose numbers have increased by 9% 2019. This means that in just one year 321 million new users have joined online communities. With the growth of these online communities the amount of hate speech, toxic behavior and cyberbullying has also increased. A 2018 survey by common sense media reported that 64% of 13-17 year olds in North America have witnessed some form of hate speech online [2]. Another study by Data& Society Research Institute reports that 91% of internet users between the ages 18-29 have witnessed someone being harassed online.

It becomes clear that there is a need for more thorough moderation of online communities. However, the sheer amount of user generated content makes it impossible to manually moderate. Some platforms use a report system to recruit the help of the users in moderation but this is a slow process and often requires manual review, it also fails to filter toxic messages before them being shared. Many moderators use blacklist or regular expression filters, but those solutions fail to detect more subtly toxic messages. Short text messages, due to the low amount of word tokens, generate sparse matrices with little to no token co-occurrence. This makes keyword extraction and context extrapolation especially difficult. The challenges in sentiment analysis and classification of short text has been explored by several other papers [4] [5].

1.2 Related work

There have been a number of papers dedicated in sentiment analysis of online short text as well as in the area of hate speech detection. A study by I. Hemalatha, Dr. G Varma & Govardhan and Dr Indukuri Latha offers examples on how sentiment analysis can be useful for social media platforms such as Twitter [7]. Proposed solutions for toxic comment detection and classification include the use of convolutional neural networks [9] or a combination of multiple classification models [11]. Other studies focus less on measuring the performance of certain classification models and instead analyze the effects of certain preprocessing transformations in toxic comment classification [21].

1.3 Our Objective

Our objective is to create a system that can analyze and detect toxic comments at scale. Our model needs to detect intent in messages and not just flag bad words. It also needs to produce minimal false positives in order to reduce the need for manual review. Creating a complete replacement for a human moderator is not under the scope but we want to offer a powerful enough tool to help with most of the workload. For this we will be proposing and evaluating 5 different models, 4 feature-based models using Logistic Regression, K-Nearest Neighbor, Support Vector Machine and Decision Tree classifiers and a Fasttext neural network model. We will be training our models using a dataset provided by Jigsaw in a Kaggle competition [19]. Due to the nature of our dataset our models are mostly aimed for use in forums and social media where the message length is longer. However, they should remain viable for use in livechats where user messages are short, especially if a dataset containing shorter messages is included during training. In section 2 we will be presenting different preprocessing steps that we will be applying to our models. Section 3 focuses on the machine learning methods used. In Section 4 we analyze the experimental results

of the proposed approaches, as well as the effects of different preprocessing methods. Section 5 is discussion and future work.

2. PREPROCESSING

Processing natural language and especially online comments presents a lot of challenges. The text is riddled with symbols, non-ASCII characters, spelling mistakes and attempts to circumvent blacklist filters by obfuscating words. Especially due to misspelling (whether intentional or not) the text corpus contains many different variations of the same word. Those variations pose challenges during feature extraction as they result to an extremely large amount of word features being created. Words are often poorly separated, with spaces missing after periods and mixed numeric characters within words, thus complicating tokenization.

- **Text cleanup** : The first preprocessing step is to clean up the text. This includes:
  - *Removing Unicode characters*
    This is done using the unidecode library (https://github.com/takluyver/Unidecode). The library allows us to take a Unicode string and represent it (usually via transliteration) in ASCII characters. If it fails to represent a Unicode character in ASCII it replaces it with ”[?]”, which we later remove from the string.

    ”HÈLLO WÖRﻉLÐ” $\xrightarrow{\text{clean\_unicode}}$ ”HELLO WORLD”

  - *Removing irrelevant ASCII characters*
    Since certain ASCII characters do not play a role in conveying sentiment, we remove them from both the training and test data. We do however keep punctuation symbols, since they play important factor in sentiment analysis. This is done by removing all characters that fit the following regular expression from the provided text:

```
# Compile regex that defines every character which is not alphanumeric,
    whitespace, or one of the listed symbols
re.compile(r'[^a-z0-9\.\?\!\,\-\#\@\% ]', re.IGNORECASE)
```

  - *Converting text to lowercase*
    We will also be converting our text to lowercase before vectorization, this will prevent ”Hello” and ”hello” from being viewed as different words by our model.

- **Tokenization** : For tokenization we’ll be considering any non alphanumeric character as a delimiter. The reason we consider numbers as part of words is because in text they can be used to replace certain letters. For example ”hello” could be written as ”h3llo” and we want it to be tokenized into a singular token including the number 3. Furthermore, we only accept tokens with length larger than one character -this helps reduce the amount of features that are not important when analyzing sentiment-. To achieve the above we use the regex ”(?u)\b\w\w+\b” as a token pattern during the tokenization step.

- **Spell correction** : This step helps minimize differently typed variations of the same word due to either incorrect spelling or attempt of the users to circumvent moderation. There are two approaches to achieve this:

  The first approach is based on a simple spellchecker made by Peter Norvig[12]. However, in our variation we'll be using an already compiled **Fasttext** [15] library as a dictionary. The library we'll be using is based on common crawl, containing 2million words ordered by their frequency in the training corpus.

  We load the word collection into a dictionary, the key being the word and the value is a rank which represents their frequency (Figure 1: lines 3-6). Then, when we receive a string of words to spell-correct, for every word -if it doesn't exist without our dictionary- we find 2 edit variations of it (Figure 1: line 25). This includes insertions, deletions, transpositions and word splits. Every one of those edits that also exists in the dictionary is added to a list.

  Finally, we pick the word with the highest ranking (highest frequency) from that list (Figure 1: line 21). This approach helps us replace unknown words with similar words that are known, however, it only takes into account the sequence similarity and ignores the linguistic context of the involved words. It also is a very slow and memory heavy process since it requires generating a dictionary with millions of entries.

```
def load_words(self):
  # Load word2vec model into a dictionary and set the value of the element to its frequency
    self.w2v = gensim.models.KeyedVectors.load_word2vec_format(EMBEDDING_FILE_GOOGLE, binary=True)
    self.words = {}
    for i, word in enumerate(self.w2v.index2word):
        self.words[word] = i

def get_words(self, text):
    return re.findall(r'\w+', text.lower())

def P(self, word):
    N = sum(self.words.values())
    # Probability of 'word'.
    try:
        return self.words[word] / N
    except:
        return 1

def correction(self, word):
    # Most probable spelling correction for word.
    return max(self.candidates(word), key=self.P)

def candidates(self, word):
    # Generate possible spelling corrections for word.
    return (self.known([word]) or self.known(self.edits1(word)) or self.known(self.edits2(word)) or [word])

def known(self, words):
    # The subset of 'words' that appear in the dictionary of WORDS.
    return set(w for w in words if w in self.words)

def edits1(self, word):
    # All edits that are one edit away from 'word'.
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [L + R[1:] for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
    inserts = [L + c + R for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(self, word):
    "All edits that are two edits away from 'word'."
    return (e2 for e1 in self.edits1(word) for e2 in self.edits1(e1))
```

Figure 1: Python Spell Checker using word dictionary

The second approach to spell correction also involves a **Fasttext** [15] model. This time, we create a **Fasttext** model by training it on our comment dataset. We also use the pre-trained library to assist with the process. Afterwards, for every input word, we find the top n (5) similar words using the trained model. We can pick the most similar word to correct to, but given that this similarity is lingual and does not represent the difference between the two sequences it becomes apparent that we're missing another metric. A metric that can help us determine the similarity between two strings is the **Levenshtein distance**[13]. This metric represents the minimum amount of single character edits required to change one word to another.

$$lev_{a,b}(i,j) = \begin{cases} max(i,j), & \text{if } \min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Now we can use this metric to discard words from the similar word list that have a high Levenshte distance from our input. This approach takes into account both the linguistic context and the edit distance between the words. On large word2vec models, however, getting the nearest neighbors of a word can be very time consuming at around 1-2 seconds.

```
def is_valid(term, w2c_item):
  # A correction is considered valid only if its linguistic similarity with the original word is more than 60% and its lavenshtein distance from the original is lesser than the length of the word/4 + 1
    return w2c_item[1] > 0.6 and nltk.edit_distance(term, w2c_item[0]) < (len(term)/4 + 1)

def check(terms, model):
    buffer = []
    for term in terms:
        recommendations = []
        # Get the 5 nearest neighbors of the original word, as described by the model
        for similar_item in model.nearest_neighbors(term, 5):
            if is_valid(term, similar_item):
              # Append the corrections to the recommendations list only if they are considered valid
                recommendations.append(similar_item[0])
        buffer.append((term, recommendations))
    return buffer
```

Figure 2: Python Spell checker using fasttext

In the above example we get the 5 nearest neighbors of each term. We could increase it to a higher number like 10 to slightly increase accuracy, but it will further impact performance.

- **Vectorization**: The two most prevalent options when it comes to text vectorization are **count** and **tf idf** (term frequency-inverse document frequency). A **count vectorizer** will create a unique integer *i* for each word. That word will be later vectorized into a vector with its *i*th direction set to 1 and all other dimensions set to 0. When vectorizing a message consisted of multiple words it will add up all the vectors and create a singular vector. The issue with the count vectorizer is that a message with more words will create larger resulting vectors, this results to a scenario where the length of the message seems to be almost as important as the contents themselves. The **tf idf vectorizer** [14] solves that issue by using word frequency instead of word count to create the message vector.

- **Word n-grams**: In natural languages, words often have multiple meanings which change depending of the context the word is being used in. Word n-grams represent a series of words in a given sample of text (or speech). They allow to store context around a word so meaning can be derived more accurately. N-grams can have different sizes depending on

how many words they're consisted of, a unigram is a single word n-gram, size 2 is bigram, size 3 trigram and so on.

During our vectorization preprocessing step, we have the ability to extract n-grams from the given data. The vectorizer allows us to define a range of n-grams to extract so we'll be extracting n-grams of size 1,2,3 and 4. This should allow the estimator to better understand the context the words are being within, at the cost of a much higher memory footprint.

```
>>> vectorizer = TfidfVectorizer(ngram_range=(1, 2))
>>> print(vectorizer.fit(["the quick brown fox jumped over the lazy dog"]).
    vocabulary_)
{'the': 13, 'quick': 11, 'brown': 0, 'fox': 3, 'jumped': 5, 'over': 9, 'lazy':
     7, 'dog': 2, 'the quick': 15, 'quick brown': 12, 'brown fox': 1, 'fox
    jumped': 4, 'jumped over': 6, 'over the': 10, 'the lazy': 14, 'lazy dog':
    8}
```

Figure 3: Example of the TfidfVectorizer creating a dictionary of unigrams and bigrams from a sentence

- **Extracting Surface features**: During the above steps we've done some processing that has destructively removed some characteristics of the comment text. Most importantly, the vectorizer ignores the characters case and importance of punctuation. This means that sentence typed in all caps and a sentence typed in lowercase appear the same to the model. While that is good in order to help extrapolate the linguistic context of the words within the sentences, it also removes an important metric for analyzing sentiment.

  In order to mitigate the situation we extract a few quality based surface features.

  - *Lengthiness*
    A measure of the length of the comment compared to the average length of comments [22].
  - *Capital word Frequency*
    The frequency of words that are written in all uppercase letters [23], often associated with shouting.
  - *Uppercase Frequency*
    The frequency of uppercase character in the message. This is extracted in addition to the capital word frequency in order to cover scenarios where sentences without spaces don't get tokenized into multiple words.
  - *Exclamation mark Frequency*
    Measuring the frequency of exclamation marks is based on the surface feature proposed in this paper regarding the assessment of post quality in online discussion [23]. However, instead of the proposed percentage of sentences that end with a question mark, we'll be measuring exclamation frequency by calculating the ratio of the amount of question marks in a comment by the length of the comment.

  All those metrics are passed through a standard scaler before they are fed, along with the text data, into the classifier.

3. PROPOSED MODELS

We apply various different techniques to classify our data. Since our dataset is already labeled with the toxicity of each comment we will be using supervised learning models.

3.1 Regression and Classification Models

- **Logistic Regression**
  The Logistic Regression algorith is very competitive for use in binary classification, used in numerous papers [24][25][6]. In Logistic Regression, we train a linear model that predicts unknown parameters using the known parameters (data). The trained linear model uses a logistic function ($f(x) = \frac{L}{1+e^{-k(x-x_0)}}$) to estimate the binary (with two possible values) dependent variable. By default, the linear model only provides a probability of the dependent variable having a specific value, however, by specifying a probability cutoff (usually set to 0.5) we can turn it into a classifier. Furthermore, that probability can also be used to describe the classifiers confidence on a prediction.

- **Support Vector Machine (linear kernel)**
  Support vector machines are a type of supervised learning classifier. When trained, an SVM model creates a map of points using the training vectors. Those points define gaps that surround the clusters formed by the training vectors. When classifying new input the SVM maps the input vectors into the generated space and categorizes them based on which side of the gap they fall on.

  There is a linear and non-linear version of Support Vector Classifiers (SVC) in the library we use. The non-linear SVC uses the library LIBSVM, which implements the Sequential Minimal Optimization (SMO) algorithm. The linear SVC on the other hand uses the LIBLINEAR library. This library implements linear SVMs and logistic regression models trained using a coordinate descent algorithm.

  Since we use a relatively large dataset and our problem is a binary classification one, we will be using the linear SVC. This will retain the performance of the non-linear SVC but will be much more efficient.

- **Decision Trees**
  There are two main types of decision tree classifiers: Random Forest and Extra (Randomized) Trees Classifier. We will be studying the performance of the Extra Trees Classifier.

- **K-Nearest Neighbors**
  KNN is another popular classifier in the field of text classification. It finds the nearest neighbors of a given input and classifies the input based on the distance of the input from them using a provided weight system. We will be using euclidean distance to measure the distance between vectors.

3.2 Neural Network Model

As a Neural Model we'll be using a Fasttext model. The **Fasttext** system is a shallow neural network based on these papers: [15][16] that generates word embeddings. A primary way of generating word embeddings is the skip-gram model, used by the word2vec neural networks [17]. Skip-gram is a generalization of ngrams as it does not require for words to be in consecutive order, instead it can ”skip” over words to account for sparse data. This is limiting however, since it creates poor representations of rare words, especially if they're not a part of the training corpus. Fasttext attempts to mitigate this issue by introducing the ngram-model, which further builds upon the skip-gram model by introducing subword information (character n-grams). This allows it to extrapolate the linguistic meaning of unknown words by using the extracted

subword information.

The supervised Fasttext model supports multiple labels but not multiple features. This means that our preprocessing step which adds extra features can’t be included. However, due to word n-grams and subword information is still very capable of understanding intent and sentiment.

4. EXPERIMENTAL RESULTS

In this section we will explore the dataset used to train and test the previously discussed models. Then, we will analyze their performance experimentally.

4.1 Dataset Description

The dataset used for the evaluation of the models consists of 159 thousand wikipedia comments with a unique id and a toxicity rating. It is provided by Jigsaw[20] for a competition on Kaggle[19]. The comments are written in english but can contain non-english words, non-ASCII characters and spelling mistakes. The length of the comments ranges from 1 word to hundreds of words.

The original dataset contained 6 flags for each comment:

- Toxic
- Severe Toxic
- Obscene
- Threat
- Insult
- Identity Hate

A comment can be tagged with none, any or all of the above tags. The multiple tags, however, are not very helpful in the context of whether a context should be flagged or not. In that scenario we'd want a single "Yes" or "No" flag, or some confidence percentage on how toxic a comment is considered. Also, as we'll see below they can even be detrimental.

Let's examine some entries from the original dataset:

	Comment	Tags
1	What are the numerical parameters used in the plot?	-
2	UNBLOCK ME OR I'LL GET MY LAWYERS ON TO YOU FOR BLOCKING MY CONSTITUTIONAL RIGHT TO FREE SPEECH	Toxic
3	I think that your a Fagget get a oife and burn in Hell I hate you 'm sorry we cant have any more sex i'm running out of conndoms	Toxic Obscene Threat Insult Identity Hate
4	Current source. I'm not sure what you're asking about the op-amp current. - 21:39, August 6, 2005 (UTC)	-
5	you have a sandy vagina	Toxic Obscene Insult
6	Don't be such a sandy vagina. Leave it to a fag to be such a let down. People are just having fun editing joke pages. No real person would ever sight wikipedia as a credible source. Your job and everything you do is worthless. You're worthless as a human being. Take the dick out of your ass and calm down.	Toxic

Figure 4: Entries from the original dataset

We can see that comments that are mildly toxic (such as comment 2) are marked with the tag Toxic only. Also, usually, the more labels a comment is tagged with the more severely toxic it is. Exceptions to that are comments like entry 6 above, it is certainly more toxic than comment 5 and contains the same insults as both comment 5 and 3, yet it is only marked as toxic and nothing else. Clearly there are some errors in the original dataset, this issue was further explored in one of the posts on the kaggle thread[18]. However, since it would be very difficult to manually correct the issues with the dataset, we can instead try some solutions to mitigate this issue.

First, let's reduce the labels to only two: toxic and not toxic. Any comment that contains any of the tags on the original dataset is marked as toxic, if it isn't marked with any tags then it's tagged as non-toxic.

Let's update the above entries:

	Comment	Tags
1	What are the numerical parameters used in the plot?	Not-Toxic
2	UNBLOCK ME OR I'LL GET MY LAWYERS ON TO YOU FOR BLOCKING MY CONSTITUTIONAL RIGHT TO FREE SPEECH	Toxic
3	I think that your a Fagget get a oife and burn in Hell I hate you 'm sorry we cant have any more sex i'm running out of conndoms	Toxic
4	Current source. I'm not sure what you're asking about the op-amp current. - 21:39, August 6, 2005 (UTC)	Not-Toxic
5	you have a sandy vagina	Toxic
6	Don't be such a sandy vagina. Leave it to a fag to be such a let down. People are just having fun editing joke pages. No real person would ever sight wikipedia as a credible source. Your job and everything you do is worthless. You're worthless as a human being. Take the dick out of your ass and calm down.	Toxic

Figure 5: Dataset entries after relabeling

Now we can work on the data a bit more consistently as well as produce a singular value severity based on the confidence of the prediction.

The resulting statistics for the updated dataset are:

Uppercase Characters	17124
Exclamation marks	323
Total Characters	392779
Uppercase Words	949
Stop Words	143040
Total words	392779
Toxic Comments	16225
Non Toxic Comments	143346
Total Comments	159571

Figure 6: Dataset statistics

For the spellchecker we used several different word2vec and fastext models. One is made by Google, it has a 3 million word vocabulary with a vector length of 300 features and a size of 1.5 GB. The other word2vec model is trained by Stanford and contains 1.2 million words with a 200 vector feature length and a total size of 974MB. For the fasttext model we used a model of 2 million word vectors trained with subword information on Common Crawl. The vector length was 300 features and the size of the dataset is 7.2GB.

4.2 Evaluation Factors

When evaluating and parameterizing models we used a GridSearch algorithm. GridSearch searches the parameter space exhaustively, considering all possible combinations of parameter values. The approach takes its name from the grid-like structure the parameter values are provided in. The scoring factor takes into account precision, recall, f1-score and accuracy. In addition to that, we will examine the amount of false positives each model produces. A model with low false positives would allow for a more automated flagging system, where comments are automatically flagged and filtered if there is a high enough confidence of a comment being toxic.

4.3 Evaluating the systems

4.3.1 Spell Checker

Let’s look at some examples of the spell checkers work. We ran the spell checker on the data set and selected 10 suggested corrections to study. These are the 10 first suggestions, ignoring the ones that include adding or removing a dash (ex. make-up to makeup) or proposing a plural form of a word.

Original	Spellchecked
editting	editng
happend	happened
seperate	separte
well-intentioned	good-intentioned
naivity	naivite
kilometres	kilometeres
anti-European	pro-European
vandalise	vandalising
harrasement	harrassement
organisations	organizations

Figure 7: Spell checker predictions

We notice that a lot of the misspelled words are changed to words that are also incorrectly spelled. This happens because the fasttext model contains incorrectly spelled words too.
We can attempt to correct this issue by skipping suggestions that are not included in the english word dictionary.

Let’s see some more results after the above update:

Original	Spellchecked
vandalising	vandalizing
sub-sections	sections
anyones	someones
becuase	because
harrass	harass
seperate	separate
naivity	naivite
happend	happed
vandalise	vandalize
organisations	organizations
vandalised	vandalized
uptil	uptill

Figure 8: Updated spell checker predictions

Now the suggestions are valid English words and they express a similar meaning even if the suggestion is incorrect (see happend and sub-sections). There still exist some slightly erroneous corrections (vandalising to vandalizing) but that's due to the dictionary not containing the original words. As such, it can be corrected by just introducing said words to the dictionary. With these changes however less than 5% of the comments and about 0.06% of the words get corrected at all. This raises serious questions about the meaningfulness of such a preprocessing operation.

Let's measure the performance too:

	Time(seconds)
Average Time	5.27
Max Time	95.2

Figure 9: Spell checking the first 1k entries of the dataset: Performance information

With this average spell checking duration it's clearly not a viable option for live chats and most

larger communities. This, combined with its unimpressive effectiveness made us discard it as a preprocessing option. Perhaps a vastly more efficient and more effective spell checker could be helpful, but it seems that training the model on the incorrectly spelled words so that it understands intent is the best approach for our use case.

4.3.2 Preprocessing Evaluation

A number of studies have analyzed the effect of data preprocessing in text classification. One such study [21] explores the effect of different text transformations on models during sentiment classification. Another study [22] investigates the usefulness of content based (comment length, uppercase frequency and more) and quality based features in predicting comment popularity and user engagement. We are applying three different optional preprocessing steps. Non-alphanum character removal, stopword removal, ngram generation and adding extra quality-based features. We will study how useful each of those transformations are on each of our models.

We performed a 2-fold validation by dividing 90% of our dataset to 2 nearly equal parts. The other 10% will be used for testing.

	Logistic Regression			**Linear SVC**			**Extra Trees Classifier**		
	accuracy	recall	f1-score	accuracy	recall	f1-score	accuracy	recall	f1-score
No Preprocessing	0.95133	0.95469	0.95119	0.95493	0.95833	0.95658	0.93478	0.93552	0.92400
Remove non-alphanum chars	0.95711	0.96077	0.95826	0.95870	0.96159	0.96022	0.93912	0.93859	0.93051
Remove stopwords	-	-	-	-	-	-	-	-	-
Lowercase	0.95334	0.95651	0.95320	0.95568	0.95820	0.95637	0.93511	0.93740	0.92548
Generate ngrams	0.94361	0.94492	0.94141	0.95200	0.95476	0.95324	0.93166	0.93439	0.92224
Quality-based features	0.95129	0.95382	0.95039	0.95474	0.95764	0.95598	0.93308	0.93408	0.92332
Quality-based features LC	0.95367	0.95682	0.95374	0.95614	0.95914	0.95756	0.93542	0.93558	0.92310
All sans ngrams	0.95961	0.96422	0.96205	0.96000	0.96347	0.96220	0.94102	0.94291	0.93398
All	0.95270	0.95726	0.95499	0.96051	0.96510	0.96420	0.93680	0.93909	0.92836

Πίνακας 3: Accuracy, recall and f1-scores for different preprocessing transformations

As the above data indicates, the benefit of preprocessing our text data is small but noteworthy.

Removing non-alphanumeric characters is beneficial for all our models. The Logistic Regression Classifier benefiting the most, where removing non-alphanumeric characters yielded an increase from a 0.95119 f1-score to 0.95826. Converting our input text to lowercase doesn't offer much of a benefit, however, it seems to greatly increase the effect if combined with quality based features. This is most likely the case because the classifier can better derive linguistic meaning out of the lowercase words, without losing the important information that character case provides. The Linear Support Vector Classifier seems to benefit the most from the addition of such features, while the Logistic Regression and Extra Trees Classifier models experience no practical improveme Generating ngrams yields more surprising results, we don't know why but it ended up being detrimental for all our models. Even when combined with removal of non alphanumeric characters and lowercase text they still negatively affect the performance of the models. We will not be using ngrams when evaluating our models for this reason.

Applying all the preprocessing steps sans ngrams boosts the f1-score of our Logistic Regression model from 0.95119 to 0.96205 - a significant improvement -. Our Extra Trees and Linear Support Vector Classifiers also benefited from data preprocessing, with a 0.00998 and 0.00562 improvement in their f1-score respectively.

4.3.3 KNN Evaluation

The K-Nearest Neighbors algorithm requires very little parameterization since the main tuning parameter is the amount of nearest neighbors (N) used to classify the comment.

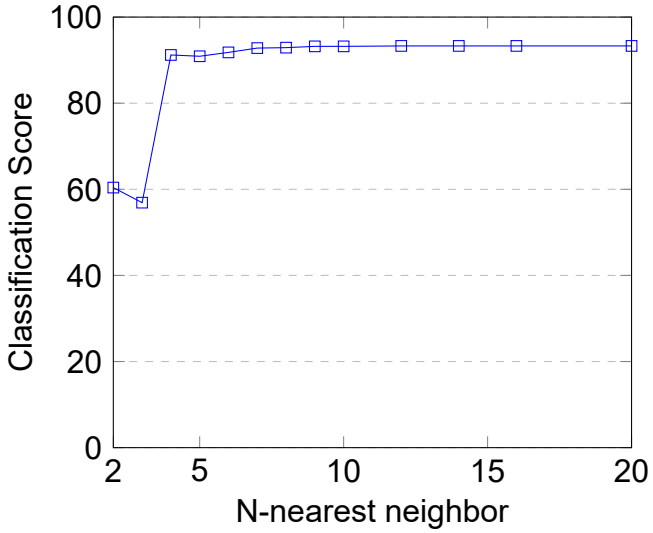

Figure 10: KNN classifier score with parameterization

In the diagram above we observe that the score for nearest neighbors is very mediocre for less than 4 neighbors and it begins to stabilize at 12 nearest neighbors or higher. On the best case scenario of 20 neighbors we can further analyze the results:

	Precision	Recall	F1-score	Support
not-toxic	0.94	0.90	0.96	14383
toxic	0.83	0.41	0.55	1575
macro avg	0.89	0.70	0.76	15958
weighted avg	0.93	0.93	0.92	15958

Figure 11: KNN Classifier Classification report

False Negatives	922
False Positives	132
Correct Predictions	14904
Total Predictions	15958
Accuracy	0.933
Total Prediction Time	1138.67s
Avg Prediction Time	71ms

Figure 12: KNN Classifier Prediction test information

4.3.4 Logistic Regression Evaluation

The Logistic Regression Classifier (LRC) has a few parameters we can tweak:

C: This parameter defines the inverse regularization strength, the lower this parameter is the more it reduces the coefficients in the resulting regression. That results in a model with more variance. If C is high then variance is reduced but this can lead to overfitting. So we want to keep this parameter relatively low.

Solver: This is the algorithm to use in the optimization problem. For a dataset of the size we're working with we have mainly two options. Saga and Sag.

Tolerance: This value is used for stopping criteria. The lower it is the longer it takes for the solver to converge.

Max Iterations: The maximum number of iterations allowed until the solver converges. If we lower the tolerance mentioned above, we may need to increase the maximum iterations so the solvers don't fail to converge. After experimentation this value was set to 700 so we can allow lower tolerances to converge.

First let us examine the regularization parameter and how it affects both solvers:

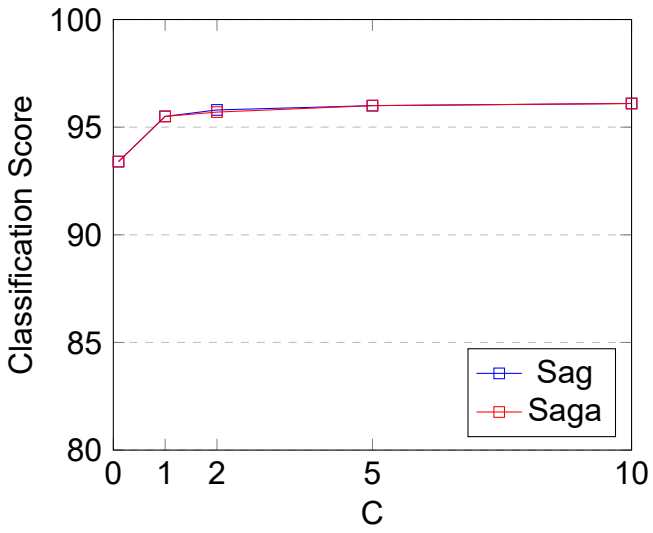

Figure 13: Logistic Regression Classifier score for different C values

The tolerance was set to 0.0005 with 700 max iterations so we can ensure convergence. We observe that the higher parameter C is, which means the lower the variance, we get better results. This is expected since we're essentially tightening our criteria. However, since higher values for the parameter C can result in over-fitting, we'd like to choose a low enough value before we experience a lot of diminishing returns. A good value for C seems to be between 2 and 5. There is also little to no difference between the two solvers. We'll continue to study them both below though.

Now lets study the performance of different tolerance levels:

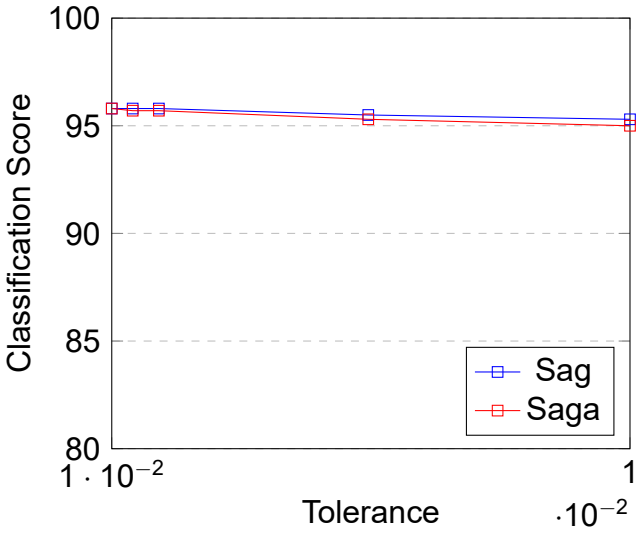

Figure 14: Logistic Regression Classifier score for different Tolerance values

As expected lower tolerance does help improve accuracy, but below 0.0001 the solvers start failing to converge and with the diminishing returns the desired tolerance ends up in the 0.0001 - 0.0005 range.

Now that we've chosen our parameters let's look into the performance of the selected model in more detail.

	Precision	Recall	F1-score	Support
not-toxic	0.97	0.99	0.98	14383
toxic	0.92	0.70	0.79	1575
macro avg	0.94	0.85	0.89	15958
weighted avg	0.96	0.96	0.96	15958

Figure 15: LR Classifier Classification report

False Negatives	476
High Confidence False Negatives	130
False Positives	130
High Confidence False Positives	12
Correct Predictions	15382
Total Predictions	15958
Accuracy	0.964
Total Prediction Time	6.02s
Avg Prediction Time	0.37ms

Figure 16: LR Classifier Prediction test information

This approach produces fairly good results, false positives are low. Especially the ones with high confidence (confidence > 90%), consist 12% of the false positive set. In total, the incorrectly classified comments that were predicted with high confidence are only 0.89% of the predictions. The fact that high confidence false positives make only 0.075% of the total predictions also means that the algorithm could automatically flag and filter comments when confidence is high enough without making many mistakes.

4.3.5 Linear SVC Evaluation

The Linear Support Vector Classifier shares similarities with the Logistic Regression Classifier (LRC) in terms of parameterization. The parameters are:

C: Same as for the LRC, defines the inverse regularization strength.

Tolerance: The value used for stopping criteria. The lower it is the longer it takes for the solver to converge.

Max Iterations: The maximum number of iterations allowed until the solver converges. This time, since the SVC converges much more slowly, we'll need a significantly higher value (than for the LRC) of 5000.

Optimization problem type: We can solve for one of two optimization problems, primal and dual. Usually, when the amount of samples is larger than the amount of features (like in our problem), selecting an algorithm that solves the primal optimization problem is preferred. We will study both of the options though.

First we should decide on the optimization problem type to solve:

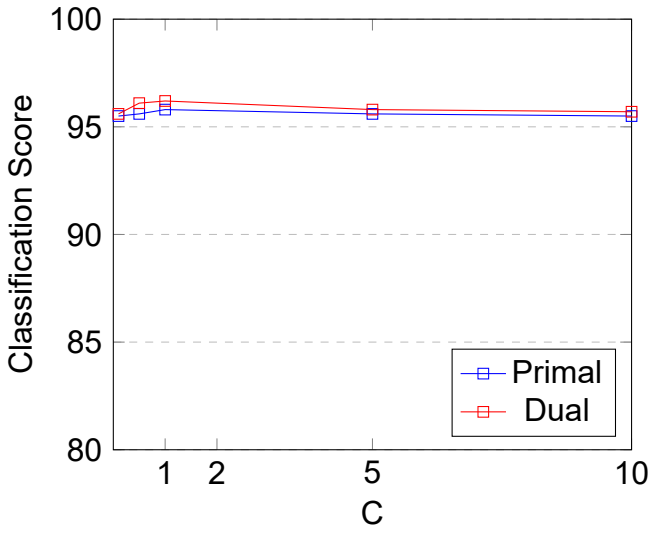

Figure 17: Support Vector Classifier score Primal/Dual

As expected, solving for the primal optimization problem gives us better results with our dataset, so we'll focus on that from now on.

Next step is adjusting our regularization strength:

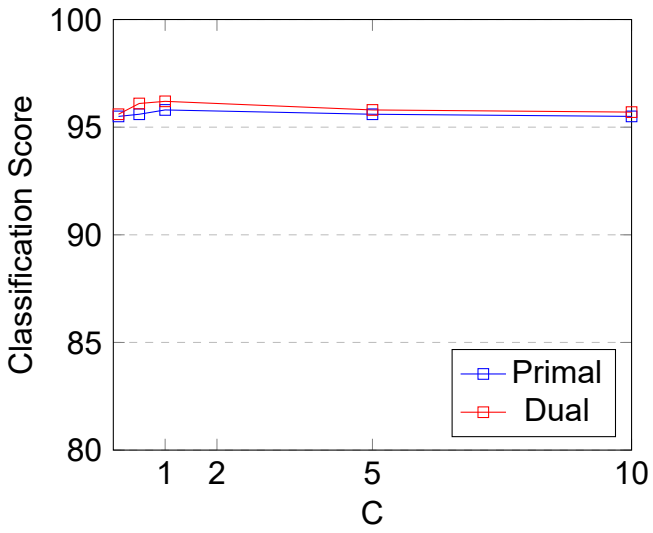

Figure 18: Support Vector Classifier score for different C values

We can observe that similarly to our Logistic Regression Classifier the best C value is around 1. This time however we notice a regression happening earlier on as the value of C increases.

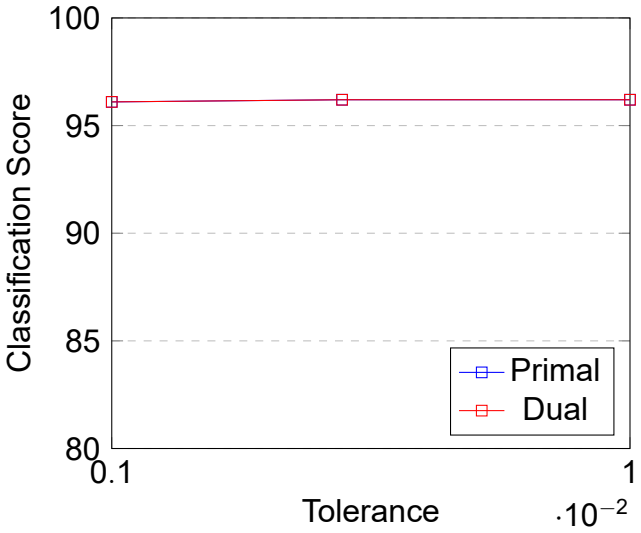

Figure 19: Support Vector Classifier score for different Tolerance values

Examining tolerance parameterization gives us much less varying results than for the LRC. We can pick a value of 0.005 and proceed into further analyzing the model.

	Precision	Recall	F1-score	Support
not-toxic	0.97	0.99	0.98	14383
toxic	0.89	0.73	0.80	1575
macro avg	0.93	0.86	0.89	15958
weighted avg	0.96	0.96	0.96	15958

Figure 20: SV Classifier Classification report

False Negatives	433
False Positives	130
Correct Predictions	15382
Total Predictions	15958
Accuracy	0.964
Total Prediction Time	6.14s
Avg Prediction Time	0.38ms

Figure 21: SV Classifier Prediction test information

The Support Vector Classifier performs very well, it has less false negatives than any of the other classifiers while still maintaining a low false positive ratio. However, it does not provide us with a confidence for every prediction. This means that we cannot as reliably automatically flag comments with high probability of being toxic.

4.3.6 Decision Trees Evaluation

The Extra Trees Classifier main adjustable parameter is the **n estimators** parameter. There are more parameters that can be tuned such as **max depth** and **max features** but those are aimed to optimize the model for very large datasets, so we will keep them off since training and prediction times weren't unreasonably high for our dataset. The **n estimators** parameter defines the amount of trees in the forest the estimator fits. Generally it is avoided to use a very small amount of estimators since it increases the chances of overfitting, but higher values significantly increase training time.

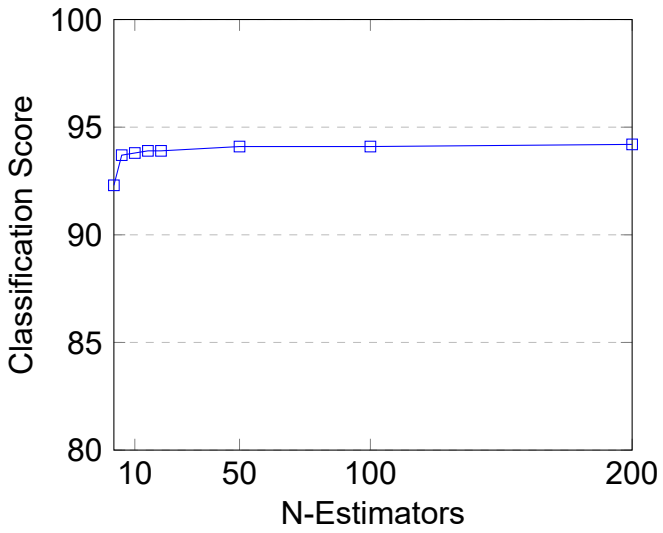

Figure 22: Extra Trees Classifier score for different N-Estimators values

We notice above that when using more than 20 estimators the classification score is very stable. In order to reduce the chance of overfitting we'll be using 100 estimators, higher values suffer from diminishing returns and greatly increase training and prediction speed.

	Precision	Recall	F1-score	Support
not-toxic	0.94	1.00	0.97	14383
toxic	0.93	0.45	0.61	1575
macro avg	0.94	0.73	0.79	15958
weighted avg	0.94	0.94	0.93	15958

Figure 23: ET Classifier Classification report

False Negatives	865
High Confidence False Negatives	83
False Positives	53
High Confidence False Positives	7
Correct Predictions	15382
Total Predictions	15958
Accuracy	0.942
Total Prediction Time	7.12s
Avg Prediction Time	0.44ms

Figure 24: ET Classifier Prediction test information

According to the data above the classifier creates many more false negatives than it does false positives. The accuracy in fact suffers from the large amount of false negatives. We'll go into more detail in section 3.4 when comparing it to the other classifiers.

4.3.7 Fast text classifier Evaluation

Fasttext has a plethora of parameters to adjust. However, due to their sheer amount we will only be evaluating the most prevalent ones:

Pretrained vectors: The fasttext model can be trained with the assistance of a pretrained word library. This can help the model have a better understanding of the similarity between and their linguistic meaning outside of the training dataset. As a result, not only can it help increase accuracy

Word vector dimensions: The amount of dimensions a word vector is comprised of. A higher vector length can allow the model to better describe the relation between words, however it can also suffer from diminishing returns. Furthermore, when using a pretrained library, the word vector length cannot be larger than the length of the vectors in the trained library. Since our library contains vectors of length 300, we will be setting that value when using it. When we are not using the pretrained library we will evaluate the model for the vector lengths of 1000 and 300.

Character ngram size: The size of the character ngrams that the model will extract. This is expressed by two numbers, one defining the lower bound (min) and the other the higher bound (max). When min is set to 3 and max is set to 5, the model will generate character ngrams of size 3, 4 and 5. For our evaluation we will be comparing models using character ngrams of range (3,6) with models that do not use this feature.

The parameters we will not be evaluating:

Word ngram size: Similar to character ngrams but for word tokens. We will always be extracting word ngrams of sizes 1 to 5.

Loss function: The loss function is what the algorithm uses in order to calculate the weights of the neural network during the optimization process. We will be using the default softmax function as a loss function.

Learning rate: The learning rate describes how much the model adjusts the weights when they're updated, based on the calculated error. If the learning rate is too low, the neural network can fail to effectively reduce the error. If the learning rate is too high, then the model can end up overadjusting on the errors and training will be unstable. Thankfully, Fasttext uses a dynamic learning rate. This means that the learning rate will decrease as the model stabilizes, so setting a custom value usually has only a small impact. As such, we will be using the default learning rate of 0.1.

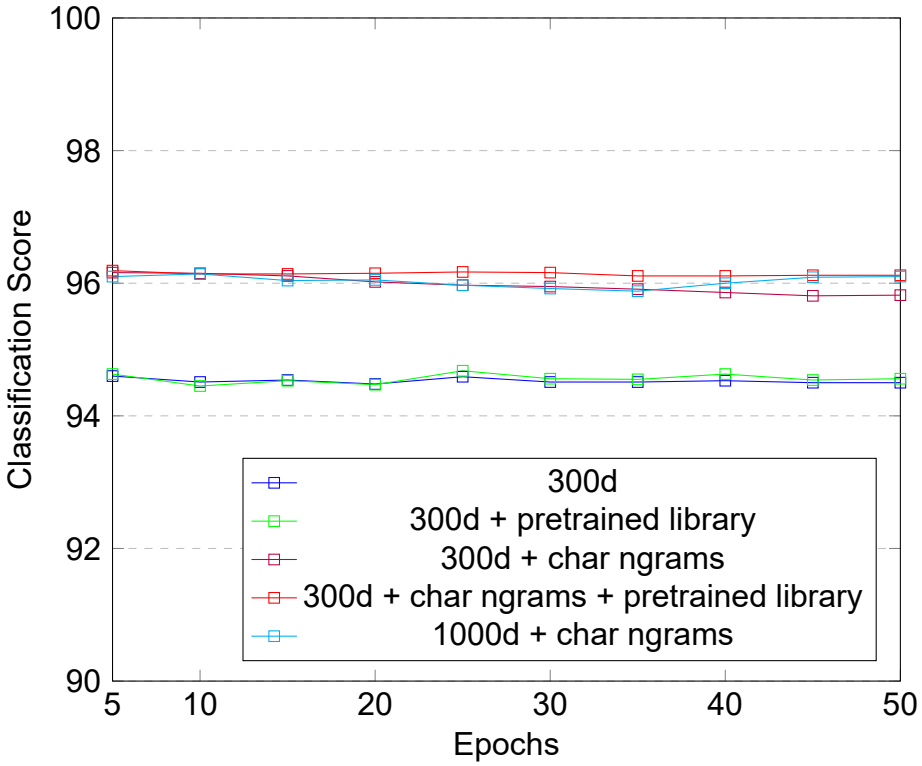

Figure 25: Fasttext Supervised Classifiers scores for different number of epochs

As shown in the figure above, character ngrams play a very important role in the accuracy of the model. By introducing character ngrams to our models we gained 1.5% on our classification score across the board. Introducing the pretrained word library made little to no difference when not extracting character ngrams. However, when we did introduce ngrams the use of the pretrained library had a noteworthy positive impact, especially for a higher number of epochs. We are unsure why, but perhaps this can be attributed to the increased stability the library can offer which made the model avoid the falloff that the alternative experienced as the amount of epochs increased. We can mostly make up for the absence of a pretrained library by increasing the vector dimensions to 1000. Nevertheless, since it still failed to achieve a higher classification score and it is less likely to handle words outside of the vocabulary well, it is a less favorable solution. To conclude, we will be using character ngrams and the pretrained library on our proposed model.

False Negatives	416
High Confidence False Negatives	348
False Positives	182
High Confidence False Positives	124
Correct Predictions	15360
Total Predictions	15958
Accuracy	0.962
Total Prediction Time	9.55s
Avg Prediction Time	0.59ms

Figure 26: Fasttext Classifier Prediction test information

The Fasttext classifier performed reasonably well. It's main weakness however is how often its incorrect predictions are high confidence ones. 78.9% of the incorrect predictions were in fact predicted with high confidence. This makes it harder to recommend for automatic comment flagging.

4.4 Comparing the classifiers

Overall, the only classifier that performed notably poorly was the KNN classifier. Not only did it have the lowest f1-score of 0.76 and the lowest accuracy at 93.3%, it also had by far the highest prediction time out of all the proposed models. This makes it very hard to recommend for use in any scenario. The Extra Trees classifier also had a relatively low f1-score of 0.79, however it had the lowest amount of false positives. Coupled with the confidence metric it produces, the extra trees classifier seems as a fairly appealing for automatic comment flagging, perhaps in combination with one of the other classifiers to make up for the large amount of false negatives. It is important to note that these characteristics it presents may be attributed to the dataset we're using, perhaps this behavior will not persist when training on different toxic comment datasets. The FastText model had one of the higher f1-scores, but it failed to match the performance of the Logistic Regression and SVM Classifiers. This may be attributed to its lack of support for extra generated features that the regression classifiers do support. The Linear SVM and Logistic Regression classifiers had the highest classification score out of all the proposed models. Out of the two, however, the Logistic Regression model provides a confidence output value which makes it considerably more practical. As such it is the recommend solution for online hate speech detection, out of the models we evaluated.

5. CONCLUSION

In this thesis, we proposed several regression models and a shallow neural network model for online hate speech detection. We discussed inconsistencies and errors within the dataset, and examined the importance of preprocessing transformations. Removing non alphanumeric characters and extracting quality based features were two transformations that achieved the most positive results. On the other hand, we noticed that generating n-grams had little to no benefit and even yielded worse results on some of our models. Out of the proposed models, the Linear SVC and Logistic Regression models achieved the highest classification scores. Our Decision Trees classifier presented the interesting quality of a very low false positive rate. This characteristic could make it useful when used in combination with other classification models. It is important to note that our Fasttext neural network, while it did not perform as well as the above regression models, is a shallow neural network and therefore doesn’t have the representational power of deep neural networks.

6. FUTURE WORK

The toxic comment dataset used in this paper is limited to forum comments extracted from wikipedia. This means that we do not properly examine the performance of the models in shorter text messages, where sentiment extraction becomes much more challenging. It would be interesting to study the proposed models on a dataset containing shorter messages, similar to those found in live chats. During our preprocessing step we examined two different spell correction algorithms and found them lacking. In a future paper, we would like to research the viability of a smaller scale spell corrector that focuses on correcting common typos and spelling mistakes on frequently used words. Finally, we suggest further research in deep neural networks (such as convolutional neural networks) in order to examine how they compare to our shallow fasttext model and regression classifiers.

REFERENCES

[1] Digital 2020 - Global Digital Interview https://datareportal.com/reports/digital-2020-global-digital-overview

[2] Common Sense ”SOCIAL MEDIA, SOCIAL LIFE. Teens Reveal Their Experiences” 2018 https://www.commonsensemedia.org/sites/default/files/uploads/research/2018_cs_socialmediasociallife_fullreport-final-release_2_lowres.pdf

[3] Amanda, Lenhart & Kathryn, Zickuhr ”ONLINE HARASSMENT, DIGITAL ABUSE, AND CYBERSTALKING IN AMERICA” 2016 https://www.datasociety.net/pubs/oh/Online_Harassment_2016.pdf

[4] Li, Yichen & Tripathi, Arvind & Srinivasan, Ananth. ”Challenges in Short Text Classification: The Case of Online Auction Disclosure” (2016). MCIS 2016 Proceedings. 18. http://aisel.aisnet.org/mcis2016/18

[5] Chen, Jindong & Hu, Yizhou & Liu, Jingping & Xiao, Yanghua & Jiang, Haiyun. (2019). Deep Short Text Classification with Knowledge Powered Attention. Proceedings of the AAAI Conference on Artificial Intelligence. 33. 6252-6259. 10.1609/aaai.v33i01.33016252.

[6] van Aken, Betty & Risch, Julian & Krestel, Ralf & Löser, Alexander. (2018). Challenges for Toxic Comment Classification: An In-Depth Error Analysis. 10.18653/v1/W18-5105.

[7] Hemalatha, I & Varma, G & Govardhan, Dr & Latha, Indukuri. (2014). Automated Sentiment Analysis System Using Machine Learning Algorithms. 3. 300-303.

[8] Gao, Lei & Huang, Ruihong. (2018). Detecting Online Hate Speech Using Context Aware Models. https://arxiv.org/pdf/1710.07395.pdf

[9] Georgakopoulos, Spiros & Vrahatis, Aristidis & Tasoulis, Sotiris & Plagianakos, Vassilis. (2018). Convolutional Neural Networks for Toxic Comment Classification. https://arxiv.org/pdf/1802.09957.pdf

[10] Maas, Andrew & Daly, Raymond & Pham, Peter & Huang, Dan & Ng, Andrew & Potts, Christopher. (2011). Learning Word Vectors for Sentiment Analysis. 142-150. https://ai.stanford.edu/~ang/papers/acl11-WordVectorsSentimentAnalysis.pdf

[11] Nobata, Chikashi & Tetreault, Joel & Thomas, Achint & Mehdad, Yashar & Chang, Yi. (2016). Abusive Language Detection in Online User Content. 145-153. 10.1145/2872427.2883062.

[12] Peter Norvig ”How to Write a Spelling Corrector”. [Online] https://norvig.com/spell-correct.html

[13] The Levenshtein-Algorithm. [Online] http://levenshtein.net/

[14] Rajaraman, Anand & Ullman, Jeffrey D. (2011). ”Data Mining” http://i.stanford.edu/~ullman/mmds/ch1.pdf

[15] Piotr, Bojanowski & Edouard, Grave & Armand, Joulin & Tomas, Mikolo. ”Enriching Word Vectors with Subword Information” [Online] https://arxiv.org/abs/1607.04606

[16] Armand, Joulin & Edouard, Grave & Piotr, Bojanowski & Tomas, Mikolov. ”Bag of Tricks for Efficient Text Classification” [Online] https://arxiv.org/abs/1607.01759

[17] Tomas, Mikolov& Kai, Chen& Greg, Corrado& Jeffrey, Dean. ”Efficient Estimation of Word Representations in Vector Space” [Online] https://arxiv.org/abs/1301.3781

[18] A very late look at data. [Online] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52217

[19] Kaggle competition data https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[20] Jigsaw website https://jigsaw.google.com/

[21] Fahim Mohammad ”Is preprocessing of text really worth your time for online comment classification?” https://arxiv.org/abs/1806.02908

[22] Brand, Dirk Johannes & Brink van der Merwe. ”Comment classification for an online news domain.” (2014).

[23] Weimer, Markus & Gurevych, Iryna & Mühlhäuser, Max. (2007). Automatically Assessing the Post Quality in Online Discussions on Software.

[24] Maas, Andrew & Daly, Raymond & Pham, Peter & Huang, Dan & Ng, Andrew & Potts, Christopher. (2011). Learning Word Vectors for Sentiment Analysis. 142-150.

[25] Lei Gao, Ruihong Huang ”Detecting Online Hate Speech Using Context Aware Models”. [Online] https://arxiv.org/abs/1710.07395