Is preprocessing of text really worth your time for toxic comment classification?

Fahim Mohammad
Intel Corporations, Hillsboro, OR, USA

Abstract—*A large proportion of online comments present on public domains are usually constructive, however a significant proportion are toxic in nature. The comments contain lot of typos which increases the number of features manifold, making the ML model difficult to train. Considering the fact that the data scientists spend approximately 80% of their time in collecting, cleaning and organizing their data [1], we explored how much effort should we invest in the preprocessing (transformation) of raw comments before feeding it to the state-of-the-art classification models. With the help of four models on Jigsaw toxic comment classification data, we demonstrated that the training of model without any transformation produce relatively decent model. Applying even basic transformations, in some cases, lead to worse performance and should be applied with caution.*

Keywords: toxic comment classification, deep learning, preprocessing, NLP, AI

1. Introduction

Lately, there has been enormous increase in User Generated Contents (UGC) on the online platforms such as newsgroups, blogs, online forums and social networking websites. According to the January 2018 report, the number of active users in Facebook, YouTube, WhatsApp, Facebook Messenger and WeChat was more than 2.1, 1.5, 1.3, 1.3 and 0.98 billions respectively [2]. The UGCs, most of the times, are helpful but sometimes, they are in bad taste usually posted by trolls, spammers and bullies. According to a study by McAfee, 87% of the teens have observed cyberbullying online [3]. The Futures Company found that 54% of the teens witnessed cyber bullying on social media platforms [4]. Another study found 27% of all American internet users self-censor their online postings out of fear of online harassment [5]. Filtering toxic comments is a challenge for the content providers as their appearances result in the loss of subscriptions. In this paper, we will be using *toxic* and *abusive* terms interchangeably to represent comments which are inappropriate, disrespectful, threat or discriminative.

Toxic comment classification on online channels is conventionally carried out either by moderators or with the help of text classification tools [6]. With recent advances in Deep Learning (DL) techniques, researchers are exploring if DL can be used for comment classification task. Jigsaw launched Perspective (`www.perspectiveapi.com`), which uses ML to automatically attach a confidence score to a comment to show the extent to which a comment is considered toxic. Kaggle also hosted an online competition on toxic classification challenge recently [7].

Text transformation is the very first step in any form of text classification. The online comments are generally in non-standard English and contain lots of spelling mistakes partly because of typos (resulting from small screens of the mobile devices) but more importantly because of the deliberate attempt to write the abusive comments in creative ways to dodge the automatic filters. In this paper we have identified 20 different atomic transformations (plus 15 sequence of transformations) to preprocess the texts. We will apply four different ML models which are considered among the best to see how much we gain by performing those transformations. The rest of the paper is organized as follows: Section 2 focuses on the relevant research in the area of toxic comment classification. Section 3 focuses on the preprocessing methods which are taken into account in this paper. Section 4 is on ML methods used. Section 5 is dedicated to results and section 6 is discussion and future work.

2. Relevant Research

A large number of studies have been done on comment classification in the news, finance and similar other domains. One such study to classify comments from news domain was done with the help of mixture of features such as the length of comments, uppercase and punctuation frequencies, lexical features such as spelling, profanity and readability by applying applied linear and tree based classifier [8]. FastText, developed by the Facebook AI research (FAIR) team, is a text classification tool suitable to model text involving out-of-vocabulary (OOV) words [9] [10]. Zhang et al shown that character level CNN works well for text classification without the need for words [11].

2.1 Abusive/toxic comment classification

Toxic comment classification is relatively new field and in recent years, different studies have been carried out to automatically classify toxic comments.Yin et.al. proposed a supervised classification method with n-grams and manually developed regular expressions patterns to detect abusive language [12]. Sood et. al. used predefined blacklist words and

edit distance metric to detect profanity which allowed them to catch words such as sh!+ or @ss as profane [13]. Warner and Hirschberg detected hate speech by annotating corpus of websites and user comments geared towards detecting anti-semitic hate [14]. Nobata et. al. used manually labeled online user comments from Yahoo! Finance and news website for detecting hate speech [6]. Chen et. al. performed feature engineering for classification of comments into abusive, non-abusive and undecided [15]. Georgakopoulos and Plagianakos compared performance of five different classifiers namely; Word embeddings and CNN, BoW approach SVM, NB, k-Nearest Neighbor (kNN) and Linear Discriminated Analysis (LDA) and found that CNN outperform all other methods in classifying toxic comments [16].

2.2 Preprocessing of online comments

We found few dedicated papers that address the effect of incorporating different text transformations on the model accuracy for sentiment classification. Uysal and Gunal shown the impact of transformation on text classification by taking into account four transformations and their all possible combination on news and email domain to observe the classification accuracy. Their experimental analyses shown that choosing appropriate combination may result in significant improvement on classification accuracy [17]. Nobata et. al. used normalization of numbers, replacing very long unknown words and repeated punctuations with the same token [6]. Haddi et. al. explained the role of transformation in sentiment analyses and demonstrated with the help of SVM on movie review database that the accuracies improve significantly with the appropriate transformation and feature selection. They used transformation methods such as white space removal, expanding abbreviation, stemming, stop words removal and negation handling [18].

Other papers focus more on modeling as compared to transformation. For example, Wang and manning filter out anything from corpus that is not alphabet. However, this would filter out all the numbers, symbols, Instant Messages (IM) codes, acronyms such as $#!+, 13itch, </3 (broken heart), a$$ which gives completely different meaning to the words or miss out a lot of information. In another sentiment analyses study, Bao et. al. used five transformations namely URLs features reservation, negation transformation, repeated letters normalization, stemming and lemmatization on twitter data and applied linear classifier available in WEKA machine learning tool. They found the accuracy of the classification increases when URLs features reservation, negation transformation and repeated letters normalization are employed while decreases when stemming and lemmatization are applied [19]. Jianqiang and Xiaolin also looked at the effect of transformation on five different twitter datasets in order to perform sentiment classification and found that removal of URLs, the removal of stop words and the removal of numbers have minimal effect on accuracy whereas replacing negation and expanding acronyms can improve the accuracy.

Most of the exploration regarding application of the transformation has been around the sentiment classification on twitter data which is length-restricted. The length of online comments varies and may range from a couple of words to a few paragraphs. Most of the authors used conventional ML models such as SVM, LR, RF and NB. We are expanding our candidate pool for transformations and using latest state-of-the-art models such as LR, NBSVM, XGBoost and Bidirectional LSTM model using fastTextâĂŹs skipgram word vector.

3. Preprocessing tasks

The most intimidating challenge with the online comments data is that the words are non-standard English full of typos and spurious characters. The number of words in corpora are multi-folds because of different reasons including comments originating from mobile devices, use of acronyms, leetspeak words (`http://1337.me/`), or intentionally obfuscating words to avoid filters by inserting spurious characters, using phonemes, dropping characters etc. Having several forms of the same word result in feature explosion making it difficult for the model to train. Therefore, it seems natural to perform some transformation before feeding the data to the learning algorithm.

To explore how helpful these transformations are, we incorporated 20 simple transformations and 15 additional sequences of transformations in our experiment to see their effect on different type of metrics on four different ML models (See Figure 1).

- Remove rare words: In the Jigsaw toxic text corpora, a staggering 65.3% of the words occurred just once and 88.3% of the words appeared five or less number of times (See Fig. 2(a)). This shows that there are many different ways to represent the same words. The Fig. 2(b) below shows different number of ways (that we could identify, actual number may be more) some of the abusive words are written in the Jigsaw corpora.
- Use regular expression for blacklisted words: A regular expression is created for each one of the blacklisted word and every word in corpora is compared to see which is matched. The âĂŸ*âĂŹ (asterisk) is assumed to be the wild character that can match any character. Our algorithm knows that s**t, S***T, sh**, shi*, s*it:), SHYT, sHYt, shiiiit, shiiiiiiiiiiit and siht, all represent the same word.
- Check if the words if they look like proper name: A large number of words with frequency less than 10 looked like proper names (person, city or other proper names). We matched each words with compiled list of 1) city names 2) countries 3) nationalities 4) ethnicities 5) names of persons (a. English names, b. Spanish

	Preprocessing Step	Description	Example: Before (After)
1	Raw	This is raw data without any preprocessing.	Hell$$o (Hell$$o)
2	To_lower	Convert the texts to lowercase	Hello (hello)
3	Remove_whitespaces	Replace multiple whitespaces (newlines, spaces) to one	hi hello (hi hello)
4	Remove_leaky	Remove leaky information such as IP addresses and user_ids and replace with abstract notation such as ipaddress or userid	[[charles123]] (' userid '), 11.11.11.11 (' ipaddress ')
5	trim_words_len	Trim long words (len >30) to a word of length 30.	
6	Strip_non_printables	Remove all non-printable characters.	HelʃʃΔlo (Hello)
7	Replace_contractions	Replace contraction words with their expanded counterpart.	haven't (have not), hell (he will)
8	Replace_acronyms	Replacing the acronyms with corresponding meaning. We crawled websites such as www.noslang.com and www.urbandict.com and aggregated more than 7000 popular acronyms being	b4n, (best friend for life),<3 (love)
9	Remove_stopwords	Stop words such as the, a, am, is, on, etc. are extremely common words assumed to provide a little or no help in information retrieval.	the (''), am (''), is ('')
10	Remove_rare_words	Remove words which are rare	khng (''), ladyofshalott ('')
11	Remove_non_alnum_chars	Remove all non-alphanumeric characters from strings. Available in common text processing tools	H$ello123# (Hello123), $h1+ (h1)
12	Remove_non_alpha_chars	Remove all non-alphabets in strings	H$ello123# (Hello), $h1+ (h)
13	Remove_non_alpha_words	Remove entire word if it has characters other than alphabets.	H$ello12'3@# ('') , $hit ('')
14	Regex_maping_black_list	Use regular expression to match each word	shi+ (shit)
15	Check_if_name	Clean words if they look like proper name	charles123 (charles), *roy* (roy)
16	Fuzzy_profane_map	Fuzzy match with the list of profane words.	SHUIT (sh*t), SHiiT (sh*t)
17	Fuzzy_common_map	Fuzzy match all words with frequent words in the same corpora.	cla$$ificat0n (classification)
18	Lemmatize	Lemmatization is a method of converting the words of a sentence to its dictionary form.	goodies(goody), women (woman).
19	Stemming	Stemming is a method to heuristically remove the affixes of a word to get the root of the word.	goodies (goodi), happiness (happi)
20	URL_info_extract	URL addresses may contain terms which are abusive in nature. Extract words from the URL.	http://youareidiot.com (http you areidiot com)
21	PPO-1-lower_ws_trim	toLower - whitespaces - trim	
22	PPO-2-LWTN-Lk	LWTN - leaky	
23	PPO-3-LWTN-LkCnAc	LWTN - leaky - contraction - acronym	
24	PPO-4-LWTN-St	LWTN - stopwords	
25	PPO-5-LWTN-Ra	LWTN - rarewords	
26	PPO-6-LWTN-CoAcStRa	LWTN - contraction - acronym - stopwords -rarewords	
27	PPO-7-LWTN-An	LWTN - removeNonAlphanumeric	
28	PPO-8-LWTN-Aw	LWTN - removeNonAlphabets	
29	PPO-9-LWTN-AnAw	LWTN - removeNonAlphanumeric - removeNonAlphabets	
30	PPO-10-LWTN-CoAcBk	LWTN - contraction - acronym -blacklist	
31	PPO-11-LWTN-CoAcBkPrCm	LWTN - contraction - acronym - blacklist - profane - common	
32	PPO-12-LWTN-CoAcLkBkPrCmNm	LWTN - contraction - acronym - leaky - blacklist - profane - common - checkName	
33	PPO-13-LWTN-CoAcLkAwStSm	LWTN - contraction - acronym - removeNonAlphabets - stopwords - stemming	
34	PPO-14-lower_lemma	toLower - lemmatize	
35	PPO-15-lower-AwBkCmSm	toLower - removeNonAlphabets - blacklist - common - Stemming	
		*LWTN (toLower - removeWhitespaces - trim_words_len - remove_NonPrintable)	

Fig. 1: List of transformations.

names, c. Hindi first names, d. Hindi last names e. Muslim names).

- Replace profane words using fuzzy matching: We used fuzzy matching to see how close a word is to the abusive words based on Levenshtein distance. By carefully selecting the threshold based on empirical value, the algorithm can detect that the words; SHUIT, SHYT, SHIZZ, SHiiT, SHITV, $h1+, $hit, 5h1t; represent the same word.
- Replace common words using fuzzy matching. In this transformation, we assumed that any word with a frequency of more than 100 (empirically chosen) is frequent word. Then we normalized these frequent words by removing all non-alphanumeric characters and resulted in 4,606 unique frequent words. Then, we fuzzy matched all the raw words in corpora with frequent word to get the closest word. A matching percent threshold matching_pct is used to decide if a word is a match with a frequent word)

$$matching\_pct = 1 - len(word)/50. \quad (1)$$

The preprocessing steps are usually performed in sequence of multiple transformations. In this work, we considered 15 combinations of the above transformations that seemed natural to us: Preprocess-order-1 through 15 in the above table represent composite transformations. For instance, PPO-11-LWTN-CoAcBkPrCm represents sequence of the following transformations of the raw text in sequence: Change to lower case $\rightarrow$ remove white spaces $\rightarrow$ trim words len $\rightarrow$ remove Non Printable characters $\rightarrow$ replace contraction $\rightarrow$ replace acronym $\rightarrow$ replace blacklist using regex $\rightarrow$ replace profane words using fuzzy $\rightarrow$ replace common words using fuzzy.

4. Methods

4.1 Datasets

We downloaded the data for our experiment from the KaggleâĂŹs toxic comment classification challenge sponsored by Jigsaw (An incubator within Alphabet). The dataset contains comments from WikipediaâĂŹs talk page edits which have been labeled by human raters for toxicity. Although

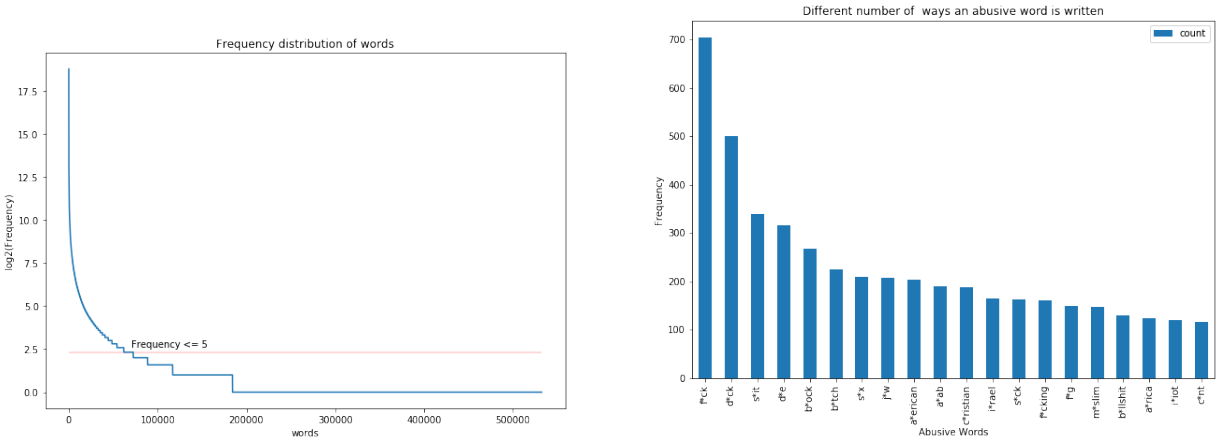

Fig. 2: a) Frequency distribution plot of the Jigsaw Toxic classification corpora. b) Different number of ways some of the commonly abusive words are written in the corpora.

there are six classes in all: âĂŸtoxicâĂŹ, âĂŸsevere toxicâĂŹ, âĂŸobsceneâĂŹ, âĂŸthreatâĂŹ, âĂŸinsultâĂŹ and âĂŸidentity hateâĂŹ, to simplify the problem, we combined all the labels and created another label âĂŸabusiveâĂŹ. A comment is labeled in any one of the six class, then it is categorized as âĂŸabusiveâĂŹ else the comment is considered clean or non-abusive. We only used training data for our experiment which has 159,571 labeled comments.

4.2 Models Used

We used four classification algorithms: 1) Logistic regression, which is conventionally used in sentiment classification. Other three algorithms which are relatively new and has shown great results on sentiment classification types of problems are: 2) NaÃŕve Bayes with SVM (NBSVM), 3) Extreme Gradient Boosting (XGBoost) and 4) FastText algorithm with Bidirectional LSTM (FastText-BiLSTM).

The linear models such as logistic regression or classifiers are used by many researchers for Twitter comments sentiment analyses [8] [19] [20] [21]. Naveed et. al. used logistic regression for finding interestingness of tweet and the likelihood of a tweet being retweeted. Wang and Manning found that the logistic regressionâĂŹs performance is at par with SVM for sentiment and topic classification purposes [22].

Wang and Manning, shown the variant of NB and SVM gave them the best result for sentiment classification. The NB did a good job on short texts while the SVM worked better on relatively longer texts [22]. Inclusion of bigrams produced consistent gains compared to methods such as Multinomial NB, SVM and BoWSVM (Bag of Words SVM). Considering these advantages, we decided to include NBSVM in our analyses as the length of online comments vary, ranging from few words to few paragraphs. The features are generated the way it is generated for the logit model above.

Extreme Gradient Boosting (XGBoost) is a highly scalable tree-based supervised classifier [23] based on gradient boosting, proposed by Friedman [24]. This boosted models are ensemble of shallow trees which are weak learners with high bias and low variance. Although boosting in general has been used by many researchers for text classification [25] [26], XGBoost implementation is relatively new and some of the winners of the ML competitions have used XGBoost [27] in their winning solution. We set the parameters of XGBoost as follows: number of round, evaluation metric, learning rate and maximum depth of the tree at 500, logloss, 0.01 and 6 respectively.

FastText [10] is an open source library for word vector representation and text classification. It is highly memory efficient and significantly faster compared to other deep learning algorithms such as Char-CNN (days vs few seconds) and VDCNN (hours vs few seconds) and produce comparable accuracy [28]. The fastText uses both skipgram (words represented as bag of character n-grams) and continuous Bag of Words (CBOW) method. FastText is suitable to model text involving out-of-vocabulary (OOV) or rare words more suitable for detecting obscure words in online comments [10].

The Long Short Term Memory networks (LSTM) [29], proposed by Hochreiter & Schmidhuber (1997), is a variant of RNN with an additional memory output for the self-looping connections and has the capability to remember inputs nearly 1000 time steps away. The Bidirectional LSTM (BiLSTM) is a further improvement on the LSTM where the network can see the context in either direction and can be trained using all available input information in the past and future of a specific time frame [30] [31]. We will be training our BiLSTM model on FastText skipgram (FastText-BiLSTM) embedding obtained using FacebookâĂŹs fastText algorithm. Using fastText algorithm, we created embedding

matrix having width 100 and used Bidirectional LSTM followd by GlobalMaxPool1D, Dropout(0.2), Dense (50, activation = âĂŸreluâĂŹ), Dropout(0.2), Dense (1, activation = âĂŸsigmoidâĂŹ).

5. Results

We performed 10-fold cross validation by dividing the entire 159,571 comments into nearly 10 equal parts. We trained each of the four models mentioned above on nine folds and tested on the remaining tenth fold and repeated the same process for other folds as well. Eventually, we have Out-of-Fold (OOF) metrics for all 10 parts. We calculated average OOF CV metrics (accuracy, F1-score, logloss, number of misclassified samples) of all 10 folds. As the data distribution is highly skewed (16,225 out of 159,571 (10%) are abusive), the accuracy metric here is for reference purpose only as predicting only the majority class every single time can get us 90% accuracy. The transformation, âĂŸRawâĂŹ, represents the actual data free from any transformation and can be considered the baseline for comparison purposes.

Overall, the algorithms showed similar trend for all the transformations or sequence of transformations. The NBSVM and FastText-BiLSTM showed similar accuracy with a slight upper edge to the FastText-BiLSTM (See the logloss plot in Fig. 3). For atomic transformations, NBSVM seemed to work better than fastText-BiLSTM and for composite transformations fastText-BiLSTM was better. Logistic regression performed better than the XGBoost algorithm and we guess that the XGBoost might be overfitting the data. A similar trend can be seen in the corresponding F1-score as well. One advantage about the NBSVM is that it is blazingly fast compared to the FastText-BiLSTM. We also calculated total number of misclassified comments (see Fig. 4).

The transformation, Convert_to_lower, resulted in reduced accuracy for Logit and NBSVM and higher accuracy for fastText-BiLSTM and XGBoost. Similarly, removing_whitespaces had no effect on Logit, NBSM and XGBoost but the result of fastText-BiLSTM got worse. Only XGBoost was benefitted from replacing_acronyms and replace_contractions transformation. Both, remove_stopwords and remove_rare_words resulted in worse performance for all four algorithms. The transformation, remove_words_containing_non_alpha leads to drop in accuracy in all the four algorithms. This step might be dropping some useful words (sh**, sh1t, hello123 etc.) from the data and resulted in the worse performance.

The widely used transformation, Remove_non_alphabet_chars (strip all non-alphabet characters from text), leads to lower performance for all except fastText-BiLSTM where the number of misclassified comments dropped from 6,229 to 5,794. The transformation Stemming seemed to be performing better compared with the Lemmatization for fastText-BiLSTM and XGBoost.

For logistic regression and the XGBoost, the best result was achieved with PPO-15, where the number of misclassified comments reduced from 6,992 to 6,816 and from 9,864 to 8,919 respectively. For NBSVM, the best result was achieved using fuzzy_common_mapping (5,946 to 5,933) and for fastText-BiLSTM, the best result was with PPO-8 (6,217 to 5,715) (See Table 2). This shows that the NBSVM are not helped significantly by transformations. In contrast, transformations did help the fastText-BiLSTM significantly.

We also looked at the effect of the transformations on the precision and recall the negative class. The fastText-BiLSTM and NBSVM performed consistently well for most of the transformations compared to the Logit and XGBoost. The precision for the XGBoost was the highest and the recall was lowest among the four algorithm pointing to the fact that the negative class data is not enough for this algorithm and the algorithm parameters needs to be tuned.

The interpretation of F1-score is different based on the how the classes are distributed. For toxic data, toxic class is more important than the clean comments as the content providers do not want toxic comments to be shown to their users. Therefore, we want the negative class comments to have high F1-scores as compared to the clean comments. We also looked at the effect of the transformations on the precision and recall of the negative class. The F1-score for negative class is somewhere around 0.8 for NBSVM and fastText-BiLSTM, for logit this value is around 0.74 and for XGBoost, the value is around 0.57. The fastText-BiLSTM and NBSVM performed consistently well for most of the transformations compared to the Logit and XGBoost. The precision for the XGBoost was the highest and the recall was lowest among the four algorithm pointing to the fact that the negative class data is not enough for this algorithm and the algorithm parameters needs to be tuned.

6. Discussion and Future Work

We spent quite a bit of time on transformation of the toxic data set in the hope that it will ultimately increase the accuracy of our classifiers. However, we empirically found that our intuition, to a large extent, was wrong. Most of the transformations resulted in reduced accuracy for Logit and NBSVM. We considered a total of 35 different ways to transform the data. Since, there will be exponential number of possible transformation sequences to try, we selected only 15 that we thought reasonable. Changing the order can have a different outcome as well. Most of the papers on sentiment classification, that we reviewed, resulted in better accuracy after application of some of these transformations, however, for us it was not completely true. We are not sure about the reason but out best guess is that the twitter data is character-limited while our comment data has no restriction on the size.

The toxic data is unbalanced and we did not try to balance the classes in this experiment. It would be interesting to

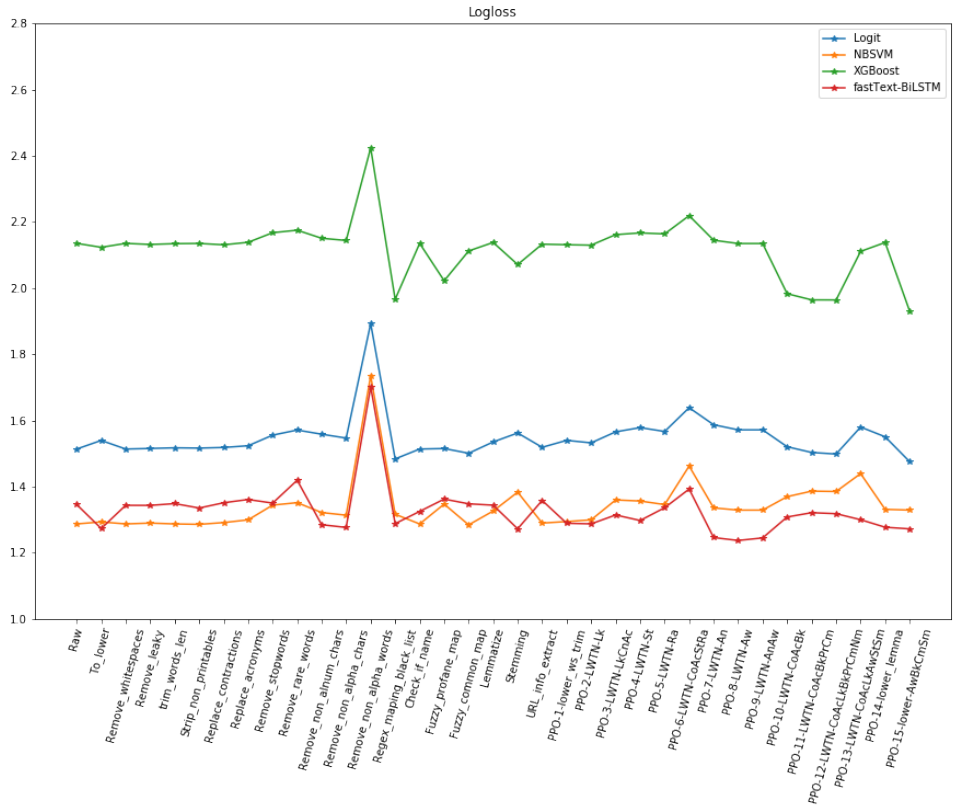

Fig. 3: Log loss plot for all four models on different transformations.

	F1-score				Overall Accuracy				Total Misclassified Comments (out of 159580)			
Preprocessing Steop	Logit	NBSVM	fastText-BiL	XGBoost	Logit	NBSVM	fastText-BiL	XGBoost	Logit	NBSVM	fastText-BiL	XGBoost
Raw	0.7407	0.7957	0.7906	0.5727	0.9720	0.9773	0.9775	0.9196	6992	5946	6217	9864
To_lower	0.7352	0.7936	0.8055	0.5757	0.9715	0.9773	0.9787	0.9196	7112	5974	5882	9809
Remove_whitespaces	0.7407	0.7957	0.7903	0.5727	0.9720	0.9773	0.9774	0.9196	6992	5946	6206	9864
Remove_leaky	0.7405	0.7951	0.7916	0.5735	0.9721	0.9773	0.9766	0.9198	7000	5958	6205	9849
trim_words_len	0.7401	0.7958	0.7906	0.5726	0.9720	0.9773	0.9774	0.9197	7009	5946	6230	9860
Strip_non_printables	0.7402	0.7959	0.7947	0.5729	0.9719	0.9772	0.9778	0.9197	7005	5940	6168	9863
Replace_contractions	0.7399	0.7951	0.7885	0.5736	0.9720	0.9773	0.9769	0.9201	7014	5965	6242	9844
Replace_acronyms	0.7393	0.7934	0.7876	0.5738	0.9719	0.9769	0.9775	0.9198	7038	6003	6287	9879
Remove_stopwords	0.7302	0.7860	0.7904	0.5643	0.9706	0.9733	0.9773	0.9007	7186	6209	6237	10013
Remove_rare_words	0.7297	0.7849	0.7735	0.5608	0.9681	0.9719	0.9737	0.9142	7257	6243	6556	10048
Remove_non_alnum_chars	0.7307	0.7885	0.8028	0.5680	0.9705	0.9761	0.9791	0.9163	7199	6105	5935	9935
Remove_non_alpha_chars	0.7337	0.7905	0.8040	0.5697	0.9709	0.9762	0.9796	0.9165	7145	6068	5897	9905
Remove_non_alpha_words	0.6577	0.7084	0.7208	0.4824	0.9462	0.9481	0.9549	0.8866	8744	8012	7859	11196
Regex_maping_black_list	0.7488	0.7913	0.8006	0.6252	0.9736	**0.9775**	0.9796	0.9303	6854	6081	5950	9083
Check_if_name	0.7407	0.7957	0.7947	0.5727	0.9720	0.9773	0.9774	0.9196	6992	5946	6121	9864
Fuzzy_profane_map	0.7422	0.7855	0.7910	0.6082	0.9718	0.9753	0.9775	0.9258	6999	6223	6293	9342
Fuzzy_common_map	0.7438	**0.7968**	0.7914	0.5794	0.9724	0.9769	0.9774	0.9224	6933	**5933**	6227	9758
Lemmatize	0.7377	0.7888	0.7918	0.5722	0.9698	0.9734	0.9774	0.9194	7091	6126	6208	9877
Stemming	0.7322	0.7782	0.8023	0.5919	0.9683	0.9715	0.9794	0.9225	7216	6390	5878	9568
URL_info_extract	0.7396	0.7953	0.7828	0.5735	0.9719	0.9773	0.9776	0.9199	7016	5958	6274	9853
PPO-1-lower_ws_trim	0.7351	0.7934	0.8006	0.5735	0.9715	0.9773	0.9783	0.9195	7113	5979	5955	9845
PPO-2-LWTN-Lk	0.7366	0.7926	0.7994	0.5738	0.9716	0.9773	0.9789	0.9193	7078	6003	5947	9839
PPO-3-LWTN-LkCnAc	0.7311	0.7825	0.7961	0.5689	0.9709	0.9760	0.9777	0.9195	7232	6281	6071	9986
PPO-4-LWTN-St	0.7247	0.7826	0.7970	0.5641	0.9702	0.9734	0.9783	0.9007	7291	6266	5994	10010
PPO-5-LWTN-Ra	0.7298	0.7846	0.7932	0.5641	0.9690	0.9733	0.9756	0.9159	7237	6216	6172	9996
PPO-6-LWTN-CoAcStRa	0.7148	0.7647	0.7830	0.5538	0.9660	0.9672	0.9733	0.8958	7569	6754	6433	10251
PPO-7-LWTN-An	0.7240	0.7844	0.8076	0.5694	0.9699	0.9761	0.9801	0.9157	7331	6170	5756	9908
PPO-8-LWTN-Aw	0.7278	0.7859	**0.8117**	0.5724	0.9702	0.9762	0.9795	0.9161	7260	6139	**5715**	9862
PPO-9-LWTN-AnAw	0.7278	0.7859	0.8079	0.5724	0.9702	0.9762	**0.9802**	0.9161	7260	6139	5751	9862
PPO-10-LWTN-CoAcBk	0.7421	0.7815	0.7995	0.6236	0.9727	0.9763	0.9780	0.9306	7024	6327	6043	9161
PPO-11-LWTN-CoAcBkPrCm	0.7466	0.7790	0.7993	0.6302	0.9733	0.9759	0.9778	0.9325	6944	6404	6103	9075
PPO-12-LWTN-CoAcLkBkPrCmNm	0.7477	0.7792	0.8004	0.6305	0.9733	0.9759	0.9775	0.9322	6922	6399	6089	9074
PPO-13-LWTN-CoAcLkAwStSm	0.7292	0.7680	0.8009	0.5871	0.9693	0.9709	0.9778	0.9123	7299	6649	6005	9752
PPO-14-lower_lemma	0.7338	0.7868	0.8038	0.5721	0.9701	0.9743	0.9787	0.9208	7163	6150	5900	9876
PPO-15-lower-AwBkCmSm	**0.7519**	0.7884	0.8076	**0.6348**	**0.9739**	0.9768	0.9786	**0.9327**	**6816**	6139	5877	**8919**

Fig. 4: Results: F1 scores, accuracies and total number of misclassified.

know what happens when we do oversampling [32] of the minority class or under-sampling of majority class or a combination of both. Pseudo-labeling [33] can also be used to mitigate the class imbalance problem to some extent.

We did not tune the parameters of different algorithms presented in our experiment. It will also be interesting to use word2vec/GloVe word embedding to see how they behave during the above transformations. Since the words in these word embedding are mostly clean and without any spurious/special characters, we can't use the pre-trained word vectors on raw data. To compare apple to apple, the embedding vectors needs to be trained on the corpora from scratch which is time consuming. Also, we only considered six composite transformations which is not comprehensive in any way and will be taking this issue up in the future. We also looked only at the Jigsaw's Wikipedia data only.

This paper gives an idea to the NLP researchers on the worth of spending time on transformations of toxic data. Based on the results we have, our recommendation is not to spend too much time on the transformations rather focus on the selection of the best algorithms. All the codes, data and results can be found here: `https://github.com/ifahim/toxic-preprocess`

7. Acknowledgements

We would like to thank Joseph Batz and Christine Cheng for reviewing the draft and providing valuable feedback. We are also immensely grateful to Sasi Kuppanagari and Phani Vadali for their continued support and encouragement throughout this project.

References

[1] CrowdFlower, "CrowdFlower Data Science Report," pp. 8–9, 2016.

[2] "Social Netword Ranking." [Online]. Available: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

[3] "McAfee Study," 2014. [Online]. Available: https://www.mcafee.com/us/about/news/2014/q2/20140603-01.aspx

[4] The Futures Company, "2014 Teen Internet Safety Survey," 2014.

[5] A. Lenhart, M. Ybarra, K. Zickuhr, and M. Price-feeney, "ONLINE HARASSMENT , DIGITAL ABUSE , AND CYBERSTALKING IN AMERICA," 2016.

[6] C. Nobata and J. Tetreault, "Abusive Language Detection in Online User Content," pp. 145–153, 2016.

[7] "Kaggle Toxic Classification Challenge." [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[8] D. Brand, "Comment Classification for an Online News Domain."

[9] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.zip: Compressing text classification models," pp. 1–13, 2016.

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," 2006.

[11] X. Zhang, J. Zhao, and Y. Lecun, "Character-level Convolutional Networks for Text," pp. 1–9, 2015.

[12] D. Yin, "Detection of Harassment on Web 2 . 0."

[13] S. O. Sood, J. Antin, and E. F. Churchill, "Using Crowdsourcing to Improve Profanity Detection," pp. 69–74, 2009.

[14] W. Warner and J. Hirschberg, "Detecting Hate Speech on the World Wide Web," no. Lsm, pp. 19–26, 2012.

[15] H. Chen, S. Mckeever, and S. J. Delany, "Presenting a labelled dataset for real-time detection of abusive user posts," *Proceedings of the International Conference on Web Intelligence - WI '17*, pp. 884–890, 2017.

[16] S. V. Georgakopoulos, S. K. Tasoulis, A. G. Vrahatis, and V. P. Plagianakos, "Convolutional Neural Networks for Toxic Comment Classification," feb 2018.

[17] A. K. Uysal and S. Gunal, "The impact of preprocessing on text classification," *Information Processing and Management*, vol. 50, no. 1, pp. 104–112, 2014.

[18] E. Haddi, X. Liu, and Y. Shi, "Haddi 2013 The Role of Text Preprocessing in Sentiment Analysis."

[19] Y. Bao, C. Quan, L. Wang, and F. Ren, "The Role of Pre-processing in Twitter Sentiment Analysis," pp. 615–624, 2014.

[20] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning Word Vectors for Sentiment Analysis," *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, 2011.

[21] H. Hamdan, P. Bellot, and F. Bechet, "Lsislif : CRF and Logistic Regression for Opinion Target Extraction and Sentiment Polarity Analysis," no. SemEval, pp. 753–758, 2015.

[22] S. Wang and C. Manning, "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification," *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, no. July, pp. 90–94, 2012.

[23] T. Chen, C. G. P. o. t. n. acm sigkdd International, and undefined 2016, "Xgboost: A scalable tree boosting system," *Dl.Acm.Org*, pp. 785–794, 2016.

[24] J. H. Friedman, "Greedy Function Approximation a Gradient Boosting Machine," 1999.

[25] S. Bloehdorn and A. Hotho, "Text classification by boosting weak learners based on terms and concepts," *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 4–7, 2004.

[26] T. Kudo and Y. Matsumoto, "A Boosting Algorithm for Classification of Semi-Structured Text," *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 17–24, 2004.

[27] D. Nielsen, "Tree Boosting With XGBoost Why Does XGBoost Win "Every" Machine Learning Competition?" *NTNU Tech Report*, no. December, p. 2016, 2016.

[28] A. Joulin, "Bag of Tricks for Efficient Text Classification," 2015.

[29] S. Hochreiter and J. urgen Schmidhuber, "LONG SHORT-TERM MEMORY," vol. 9, no. 8, pp. 1–32, 1997.

[30] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," vol. 45, no. 11, pp. 2673–2681, 1997.

[31] A. Graves, "Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures," 2004.

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[33] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," *ICML 2013 Workshop: Challenges in Representation Learning*, pp. 1–6, 2013.