Creating Adversarial Attacks for Offensive Language Detection on Twitter Data

Hope Elizabeth McGovern
Cambridge Computer Laboratory
hem52@cam.ac.uk

Abstract

In this paper, we investigate the robustness of a number of machine learning-based offensive language detection systems to adversarial attacks of two kinds: the insertion of 'not' before offensive terms and random misspellings of key offensive words ('sutpid'). We compare results from a supervised FastText classifier as well as two bi-directional long-short term memory (Bi-LSTM) networks: one which incorporates character-level information and one which operates at the word-level. The models which incorporate subword information, FastText and the char-level LSTM, are more robust to the adversarial attacks we pose than the model which only incorporates word-level information.

1 Introduction

Over the past two decades, the popularity of social media has grown immensely with the introduction of platforms like Facebook, Twitter, Reddit, Youtube, and Instagram, which allow everyday people to post their opinion to a public forum without quality control checks beyond general community guidelines. Concurrently, the amount of offensive language present on the internet has only grown, and there is now considerable interest in automatically detecting toxic, offensive, or hateful content online (Zampieri et al., 2019b; von Däniken et al., 2020; Pamungkas et al., 2020).

Offensive language detection is ultimately a variant of spam filtering. Early approaches to spam filtering included rule-based and Naive Bayes approaches, but over the past decade, machine learning has been able to outperform older methods (Davidson et al., 2017). However, the data domain of social media creates challenges for even adavnced machine learning approaches – in particular, the noisiness, small document length (for Twitter, a maximum of 280 characters), and the tendency toward new and emerging vocabulary. Offensive language detection for social media data has become a topic of interest; in 2019, the International Workshop of Semantic Evaluation (SemEval) introduced a shared task on identifying and categorizing offensive language detection (OffensEval), for which over 100 groups submitted a system (Zampieri et al., 2019a). In this paper, we re-implement some of the systems of the shared task and attempt to expose their fragility to these domain-specific weaknesses.

1.1 Adversarial Examples

While teams participating in the shared task were able to demonstrate a high accuracy on a small dataset in a relatively sterile setting, this is not necessarily a good measure of how the system would perform in an online learning or deployed setting in the real world. In addition to the inevitable typos, social media data is sometimes noisy because of self-censorship on behalf of the author: for example, someone might intentionally misspell 'shit' as 'sh!t' if they are attempting to bypass known weaknesses of the offensive speech taggers Twitter already has in-place.

It is therefore useful to generate adversarial examples, i.e. inputs designed to pinpoint weaknesses in the systems, to get a fuller picture of the robustness of our detection models. For example, Hosseini et al. (2017) demonstrate that Google's API for detecting toxic comments, 'Perspective', predicts significantly lower toxicity scores when the input is modified with various types of misspellings (e.g. replacing

'idiot' with 'iidiot' or 'i.diot'). They also note that the system is not robust to 'false alarms' (i.e. inserting the word 'not' before offensive terms) and continues to predict high toxicity to phrases that have been negated to be semantically benign. We adapt their approach to a number of the systems produced by the OffensEval 2019 task to evaluate their robustness to adversarial attacks.

2 Data

2.1 OLID

The dataset we use was introduced by Zampieri et al. (2019a) for OffensEval; it is the Offensive Language Identification Dataset (OLID) compiled from 14,100 tweets with a three-tiered hierarchy of offensive content annotation:

- Level A: is the tweet offensive or not offensive (OFF/NOT)?
- Level B: is the tweet a targeted or an untargeted offense (TIN/UNT)?
- Level C: is the target of the tweet an individual, a group, or other (IND/GRP/OTH)?

We only address the top-level hierarchy in this paper – whether or not the tweet is offensive, without attempted to identify the target – although a more fine-grained analysis could be a subject of future work. The shared task also proposes a few simple baselines (SVM, Bi-LSTM, and CNN), of which the CNN performs the best on subtasks A and B, while the CNN and Bi-LSTM tie for performance for subtask C. The F1-macro scores for the subtasks are 0.80, 0.69, and 0.47, respectively. This indicates that it remains a challenging task to predict the target of an offensive comment but is considerably easier to only detect the top-level category of offensive or not.

2.2 Pre-processing

The data provided in OLID has already been cleaned to some extent by replacing all user handles with '@USER' and URLS with the token 'URL'. On top of that, we perform a number of pre-processing steps before we pass the data into the model. First, we remove hashtags, punctuation, and a plethora of other non-ascii characters that show up in the tweets. Then, we lowercase everything and replace emojis and emoticons with their equivalent text representations. Finally, we standardize words with the help of regular expressions, limiting the number of times a character may appear to twice in a row (e.g 'no wayyyyyyy' becomes 'no wayy'). For example, the tweet:

> "#MAGA @USER 🎶 Sing like no one is listening ❤️ Love like you've never been hurt ✔️ Vote GOP when no one is watching 🙈 And don't listen to Liberals' dirt URL"

After all pre-processing becomes:

> "@user musical_notes sing like no one is listening red_heart love like you 've never been hurt check_mark vote gop when no one is watching hear_no_evil_monkey and do n't listen to liberals ' dirt url"

These pre-processing steps help to reduce sparsity in the vocabulary.

Because the task in question involves misspelling key offensive words, we want to catch and correct as many of the errors as possible that may be already present in the training data in order to have a clean baseline. As there were relatively few examples of this in the training data, we were able to correct the training set by hand without too much effort. The full codebase, including the pre-processed training, dev, and test sets used in our experiments are made available online for reproducibility[1].

Finally, we split 10% from the training set to act as a development set for hyperparameter tuning. We then report our results in Section 2 on the separate test set which has not been previewed by the model to preserve experiment objectivity.

2.3 Class Imbalance

Figure 1 illustrates the class imbalance present in the OLID training set; there are 8840 non-offensive tweets and only 4400 offensive tweets. Heavy class imbalance causes many deep learning models to exhibit bias towards the majority class (Johnson and Khoshgoftaar, 2019), so we

[1] https://www.github.com/hmcgovern/offens-eval

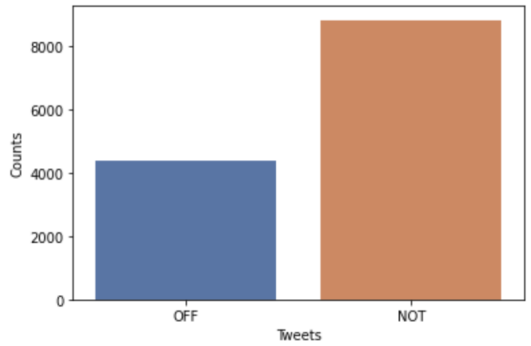

Figure 1: Number of examples of offensive and non-offense inputs in the main subtask of the OLID dataset.

downsample the majority class such that both contain 4400 examples. As a caveat, we found experimentally that the downsampled dataset led to overfitting on the FastText model, so we used the full dataset to train that classifier.

3 Models

We use three models: a supervised text classification model with FastText (Joulin et al., 2017), a bidirectional Long Short-Term Memory Network (Bi-LSTM) that uses words as features, and a nearly identical Bi-LSTM that incorporates subword information. We choose these models not only because they are not too computationally intensive and work well out-of-the-box, but because they all performed well at OffensEval (Zampieri et al., 2019b). Two of the three models use subword information, which should be to recover morphological information even from noisy text.

BERT, a powerful pre-trained unsupervised language model introduced by (Devlin et al., 2019) performed well in the shared task and was used by multiple groups and so it was considered for this experiment; but, it requires a substantial amount of computational power to train. Rather than re-implementing the most-cutting edge and complicated model, we find it more informational to produce a few well-performing models and use them as baselines for our adversarial attacks.

3.1 FastText

FastText[2] is an open-source library that can be used as either a word representation model to learn word vectors via unsupervised training or as a text classifier. FastText begins with word vectors that are averaged into a text representation before they are fed into a linear classifier; it is able to obtain high performance on text classification by leveraging subword information in the form of character n-grams (Bojanowski et al., 2017).

3.2 LSTM

We implement our LSTMs with Tensorflow[3], a popular deep learning library. The LSTMs that we use for our experiments were adapted from the description in Zampieri et al. (2019a) and contains these elements: an input embedding layer, a bidirectional LSTM layer with multiple hidden units, and an average pooling layer, whose output is passed through a *softmax* function to produce the prediction. For their baseline, they use both pre-trained word embeddings as well as a learned embedding matrix. We only use FastText pre-trained word embeddings rather than both kinds. Specifically, we use an embedding matrix pre-trained on Common Crawl and comprising 600B tokens in total, provided by Mikolov et al. (2018).

Pre-trained word embeddings have become nearly ubiquitous in NLP tasks because they have proven very effective at improving the generalization of models which only have access to a limited amount of data (Mikolov et al., 2018). This is ideal for our small, downsampled dataset of 4400 examples in each class. Having pre-trained, distributional word embeddings should help the LSTM models to generalize.

We compare a two, nearly identical LSTMs, one with and one without char-level features. The char-level model should demonstrate greater robustness to adversarial inputs as having access to subword information means that two words spelled slightly differently will still have very similar representations even though they are distinct tokens.

4 Creating Adversarial Examples

We examine two categories of adversarial examples that are enumerated in Hosseini et al. (2017). Examples of each type are given in Table 1.

[2] https://fasttext.cc/

[3] https://www.tensorflow.org/

Type	Original Phrase	Modified Phrase
I	@user someone should've taken this piece of **shit** to a volcano (95%)	@user someone should've taken this piece of **sht** to a volcano (70%)
	@user **liberals** are all kookoo (93%)	@user **liberaals** are all kookoo (80%)
	@user @user @user **trump** is a very sick man he is **toxic** and **mean** spirited (93%)	@user @user @user **turmp** is a very **siick** man he is **toxc** and **me.an** spirited (55%)
II	@user he is competing for worst president again (96%)	@user he is **not** competing for worst president again (85%)
	are you fucking serious url (69%)	are you **not** fucking serious url (32%)
	shit i will drown twitter in bullshhit if one of those is banned (94%)	shit i will **not** drown twitter in bullshhit if one of those is banned (85%)

Table 1: Examples of adversarial attacks. The left column shows the original phrase taken with its FastText probability of being classed as offensive along with the corresponding modified phrase and subsequent FastText probability.

1. **Type I** Spelling variants of keywords ('mo.ron', 'sutpid', 'dumbb', etc), that have a small edit distance to the original. To create these, we first make a lexicon of keywords that we want to permute. Included in this are common swear words as well as the key political terms like 'maga', 'liberal', and 'trump' that were used to create OLID in the first place. This lexicon can be found in the codebase. We then write a function that, for each instance of these keywords, randomly chooses to either drop a letter, double a letter, switch two adjacent letters, or insert a period in the word. These alterations are intended to mimic common mistypes. This function can be found in Appendix A. We permute all instances of the keywords present in the sentence.

2. **Type II** so-called 'False Alarms' – offensive phrases that have negated a keyword with, 'not'. We follow Hosseini et al. (2017) in only reporting on this adversarial category qualitatively, as this type of example must be made by manual modification such that they maintain grammaticality. We show a few salient examples in Table 1 and mention our qualitative assessment of this in Section 7.

5 Experimental Design

Once we have cleaned the data and balanced the classes as described in Section 2.2, we train the models.

5.1 FastText

Text classification with the Python FastText API worked well essentially out-of-the-box, but we did engage in some hyperparameter-tuning in order to maximize the model's results on the development set. As the training with FastText is very quick (on the order of seconds), this was simple to do many times to ensure a good outcome. In the end, we use a learning rate of 0.01, 100 epochs, bigram features, and a 20-dimensional representation vector. This tuning led to and accuracy on the training set of 98.6% and on the validation (dev) set of 77.7 %.

5.2 LSTM

Before training the LSTM model, we first need to tokenize each tweet, convert each token to a numerical index (or, for the character-level Bi-LSTM, we convert each character to a numerical index) including a special out-of-vocabulary token, then pad the sequences to be of constant length. For both models, we use a model with 300-dim pre-trained FastText embeddings, 50 hidden units in the LSTM layer, a dropout of 20%, an average pooling layer, and a final dense layer which performs a *softmax* classification over the two classes. The loss is binary cross-entropy, the optimizer is Adam, the learning rate is 0.001, and the number of epochs is 30, although we use early stopping criteria to halt training after five consecutive epochs for which the validation loss does not continue to decrease.

For the word-level LSTM, we set the maximum sequence length to be 50 words and the vocab size at 1000. There are 13781 unique words in the training data, and we set the vocabulary

	NOT			OFF			Weighted Avg.			
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1 Macro**
FastText	0.82	**0.89**	**0.86**	**0.64**	0.51	**0.57**	**0.77**	**0.78**	**0.78**	**0.71**
Word-LSTM	**0.85**	0.62	0.72	0.42	**0.73**	0.54	0.73	0.65	0.67	0.63
Char-LSTM	0.79	0.82	0.81	0.49	0.44	0.46	0.71	0.72	0.71	0.64
Adversarial Input										
FastText	0.78	**0.89**	**0.83**	**0.54**	0.35	0.43	**0.71**	**0.74**	**0.72**	**0.63**
Word-LSTM	**0.79**	0.74	0.77	0.43	**0.50**	**0.46**	0.69	0.50	0.46	0.61
Char-LSTM	0.78	0.83	0.80	0.47	0.40	0.43	0.69	0.71	0.70	0.62

Table 2: Results of our classification models on both the original OLID dataset and a dataset containing adversarial examples. Precision, Recall, and F1 scores are reported by label, as well as their weighted average. The final column contains each model's macro F1 score.

size to be smaller to filter out rare words. With this tuning, the model achieved an accuracy on the training set of 76% in the final epoch

For the char-level LSTM, we set the maximum sequence length to be 280, in keeping with Twitter's character limit, and set the vocab size to be the number of distinct characters in the training data, which is 60. With this tuning, the model achieved an accuracy on the training set of 70% in the final epoch.

6 Results

After we have built and tuned our models, we process the unseen test set in the same way as the development and train set. Then we generate the precision, recall, and F1-scores using scikt-learn's built-in classification report (Pedregosa et al., 2011).

We report the results of our models both on the original and perturbed input in Table 2; the best score in each column is marked in bold. On the original dataset, FastText obtains 71% macro F-1 score on the test data, follwed in order of decreasing accuracy by the char-level LSTM with 64% and then the the word-LSTM with 63%. FastText outperforms both LSTMs in recall and F1 for the 'NOT' label and in precision and F1 for the 'OFF' label, but the word-level LSTM has higher 'NOT' precision and 'OFF' recall. The char-level LSTM does not have the highest performance in any column, and noticeably under-performed in detecting instances of the minority class, 'OFF'.

For the perturbed, adversarial data, performance in every category for every model dropped. FastText still holds the overall highest macro F1 score, but they are all within 1% of each other. Performance is most affected for the 'OFF' label detection for all three models and does not change by as much for the 'NOT' designation.

7 Discussion

7.1 Type I

The main conclusion to be drawn from the results of our experiments is that misspellings do indeed have a measurable effect on the accuracy of offensive language detection models. This is evident from the decreased confidence in the predictions for the modified phrases in Table 1, as well as the metrics averaged over a modified test set in Table 2.

We expected to see that models which incorporate character-level information would be more robust to adversarial input; while the FastText model, which includes character n-grams, does better overall in both trials, it is the character-level LSTM which sees the smallest performance drop compared to the other two models when predicting on the adversarial input. Interestingly, although the char-LSTM appears to be the most robust, it did not exceed the performance of the other two models in any category. This may be a result of poor tuning of the hyperparameters.

The fact that the precision and recall of offensive tweets was affected far more than that of non-offensive tweets indicates obliquely that profanity is a good indicator of offense: we perturbed swear words without reference to the tweet's label, but the perturbation had a completely uneven effect in a given model's ability

to predict one label or the other. Investigating this further could be a direction for future work, in line with Pamungkas et al. (2020).

7.2 Type II

Table 1 demonstrates that the FastText model continues to assign a high probability of an offensive tag for the sentences which have been modified to be semantically harmless with the addition of a negating term. Only in the second example does the model actually predict the opposite tag (<50% probability) once the sentence has been perturbed. Although we use this evaluation technique to present our results in a way comparable to Hosseini et al. (2017), we find this method of evaluation flawed in that simply adding 'not' to a message with plenty of profanity or harsh sentiment does not always render the sentence benign. Therefore, it is difficult to draw a strong conclusion from this type of adversarial attack.

7.3 Future Work

Future work might also include producing adversarial examples for the other two labeled subtasks which have to do with identifying the target of offensive messages. As the best performance on those tasks from the shared task was lower than the best performance on the subtask we consider, it is likely that introducing adversarial attacks would have an even larger effect.

8 Conclusion

We have reported on the robustness of machine learning-based classifiers to a variety of adversarial inputs and found that all three models, even those which incorporate sub-word information, suffer losses in performance as a result. Those which do have character-level representations available suffer less performance loss. Of the models we tested, a simple supervised FastText sentence classifier consistently rendered the best results, while a character-level LSTM provided the most robustness to attack.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Pius von Däniken, Manuela Hürlimann, and Mark Cieliebak. 2020. Overview of the germeval 2020 shared task on Swiss German language identification. *CEUR Workshop Proceedings*, 2624(Konvens).

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. *Proceedings of the 11th International Conference on Web and Social Media, ICWSM 2017*, pages 512–515.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving Google's perspective API built for detecting toxic comments. *arXiv*.

Justin M. Johnson and Taghi M. Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Endang Wahyu Pamungkas, Valerio Basile, and Viviana Patti. 2020. Do you really want to hurt me? Predicting abusive swearing in social media. *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, (May):6237–6246.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar.

2019a. Predicting the type and target of offensive posts in social media. *arXiv*, pages 1415–1420.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019b. SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval). *arXiv*.

A Appendices

```
from random import choice
import numpy as np

def misspell(word):
    word = list(word)
    x = choice(range(4))

    if x == 0:
    # shuffle
        idx = choice(\
            np.arange(len(word)-2))
        before = word[:idx]
        subset = word[idx:idx+2]
        after = word[idx+2:]
        subset = subset[::-1]
        word = before + subset + after
    if x == 1:
    # drop letters
        idx = choice(np.arange(len(word)))
        word = word[:idx] + word[idx+1:]
    if x == 2:
    # repeat a letter
        idx = choice(np.arange(len(word)))
        word = word[:idx] + [word[idx]] + \
            word[idx:]
    if x == 3:
    # add period at random place
        idx = choice(np.arange(len(word)))
        word = word[:idx] + ['.'] + word[idx:]
    return ''.join(word)
```