Semi-Supervised Learning for Abusive Language detection

Sriharsha Srungaram
Indiana University
srisrung@iu.edu

Saumya Mehta
Indiana University
mehtasau@iu.edu

Abstract

This document is the report of our project as part of coursework for CSCI-B659: Advanced Natural Language Processing (Fall 2021). We work with OLID dataset for this project, where we focus on Task A. The first task was to replicate results from a research paper. Further, we propose few ideas on how to improve the results in the paper. We further document the changes that were successful in improving the baseline performance achieved in the paper.

For this project, you will work with the OLID and SOLID datasets. The data set is annotated on three levels, but we will focus on task A. You will train and optimize (parameters, features) three different classifiers of your choice on the OLID training set. Then you will randomly sample 15,000 tweets from SOLID and annotate them using the classifiers trained on OLID. Then you will perform tri-learning: You will choose those tweets where two of the three classifiers agree and add them to third classifier's training set, retrain all classifiers, and test on the OLID test set. Compare the results of tri-learning to adding the same amount of randomly sampled SOLID tweets with their provided label to the training sets. Create a learning curve for using 1,000, 5,000, 10,000, and 15,000 SOLID tweets for tri-learning.

1 Introduction

In Section 2, we discuss the dataset we will be using for all the analysis done in the document.
The baseline task for this project was to replicate results from the paper titled *Towards NLP with Deep Learning: Convolutional Neural Networks and Recurrent Neural Networks for Offensive Language Identification in Social Media* by Andrei-Bogdan Puiu and Andrei-Octavian Brabete (Puiu and Brabete, 2019). We discuss the details and results in section 3.
Next, we discuss three additional ideas in section 4 that could potentially improve the results obtained while replicating the paper.
Finally in section 5, we present results of all the experiments we had discussed in section 4.

2 Dataset

OLID dataset (Zampieri et al., 2019a) contains 14100 annotated tweets. It was used as the official dataset for OffensEval: Identifying and Categorizing Offensive Language in Social Media (SemEval 2019 - Task 6) (Zampieri et al., 2019b)

OLID was annotated using hierarchical annotation. Each tweet contains 3 labels, each label corresponds to one of the levels:

- **Sub-task A: Offensive Language identification**
  This gives information about offensiveness of a tweet, and it has two labels- *offensive* (OFF) and *not offensive* (NOT). There are 4400 samples that are offensive, and the remaining 8840 samples are not offensive.
- **Sub-task B: Categorization of offense types**
  This describes whether an offensive tweet is classified as **Targeted Insult and Threats** (TIN) or **Untargeted** (UNT).
  Among the offensive tweets, 524 are classified as untargeted, and 3876 are targeted.
- **Sub-task C: Offense target identification**
  This gives further information on Targeted Insult and Threats, whether the target is an **Individual** (IND), or a **Group** (GRP), or **Other** (OTH).
  Among the targeted offensive tweets, 2407 were targeted on individual, 1074 on group of individuals, and the remaining 395 were targeted on other categories.

Annotation for the dataset was done using crowdsourcing. Three annotators had labelled the data,

and gold labels were assigned based on agreement of these annotators.

3 Paper Replication

3.1 Data pre-processing

To eliminate redundant symbols and information from the tweets, the following pre-processing steps were applied before modelling.

- Emojis, hashtags, *@USER* tags, and other non-alphabetic characters were removed using regular expressions.
- Stemming is a process of replacing derived words with their root form. This would be helpful in reducing the number of unique words in the vocabulary, and hence improve the quality of embedding for count-based feature extractors.
- Lemmatization (Perkins, 2010) is a similar process provided by NLTK (Loper and Bird, 2002) that reduces the words to its valid root root based on Wordnet (Miller, 1995).
- The corpus was tokenized by splitting tweets into individual words. *word-to-index* and *index-to-word* mappings were created to facilitate looking at embeddings.

3.2 Type of input data

- **Term Frequency Inverse Document Frequency (TF-IDF)** was used for Machine learning models (Logistic regression, SVM, and Random Forest).
- **Look-up embedding Layer** was used for Deep Learning models (CNN, GRU and LSTM), where index of words in vocabulary were used to get a unique embedding from trainable embedding tensors which act as look up table.

3.3 Smart Baseline

Before training any models, to get a sense of minimum performance we can expect from the models, we built a smart baseline model. The idea was to create a simple heuristic that gives some baseline results, which we could then compare with the results we get from more complicated machine learning and deep learning models.
The idea was simple, we would compile a list of words that would be considered abusive. Any tweet that contains a word that exists in this list would be classified as offensive. If a tweet does not contain any words that are in the list of offensive words, then it would be classified as not-offensive.
To accomplish this, we calculated toxicity *(t)* of each word, which we define as the ratio of its occurrences in abusive tweets over all its occurrences.

$$t(w) = \frac{\#\ occurrences\ in\ abusive\ tweets}{total\ occurrences} \quad (1)$$

If toxicity of a word is 1, then it only occurs in abusive tweets. This indicates that if a new tweet has this word, then that tweet is very likely to be toxic.
We compiled the top *k* words in the vocabulary with highest toxicity to the list of toxic words, and used this list to classify tweets as abusive or not-abusive. A small portion of the training set was held out for evaluation, and based on the F1 scores on the evaluation set, value of *k* was chosen as 500.

3.4 Classifiers

The following classifiers were used to predict whether a tweet was abusive after applying the pre-processing steps described above.

- Logistic Regression
- Support Vector Machine (SVM)
- Random Forest
- Convolutional Neural Networks (CNN)
- Long Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

3.5 Hyperparameter Optimization

For each model that was implemented, we used grid search to optimize its hyper-parameters.
Further, we ran all neural network based models for five times, and report the best scores of the five attempts.

3.6 Results

We observed that LSTM provides best results on the OLID data test set. This is very closely followed by GRU, which is expected since both of them are part of the same family of Recurrent Neural Network architecture.

Model	Macro F1	Weighted F1
Smart Baseline	0.52	0.65
Logistic regression	0.54	0.67
SVM	0.55	0.68
Random Forest	0.57	0.68
CNN	0.64	0.71
LSTM	**0.71**	**0.78**
GRU	**0.71**	0.77

Table 1: Results for Task-A on the test set of OLID dataset.

4 Additional Experiments

Section 1.6 gave the baseline performance for all the six models. In this section, we discuss the experiments we conducted to try and improve the performance of these models.
The core ideas can be segregated into three major categories-

- Additional data pre-processing
- Pre-trained word embeddings
- A different architecture

4.1 Additional data pre-processing

Text from Twitter is different in two major ways- emojis and acronyms. People use both of them abundantly on the platform, and incorporating better ways to process them would result in a better performance of models.

4.1.1 Using description of emojis

emoji (Kyokomi, 2021) is a package for python which has mappings of emojis with their descriptions. We leveraged this mapping to replace emojis in tweets with their descriptions. Having word descriptions helps utilize already present word embeddings instead of learning them completely new embedding for emojis.

4.1.2 Twitter acronyms

TobiasWalter has compiled a dataset on Kaggle that contains a mapping of the most common acronyms in twitter and their full form. We utilize this data to replace all acronyms with their full form.

4.2 Pre-trained word embeddings

Since the dataset is limited, we believe that using pre-trained embeddings would improve performance of models since they wouldn't have to learn representation of words from scratch.

4.2.1 GloVe

Global Vectors for Word Representation (GloVe) is an unsupervised algorithm created by Jeffrey Pennington, Richard Socher and Christopher Manning (Pennington et al., 2014).
GloVe creates vector representations for words. Training is generally done on word to word co-occurence statistics from some corpus. The resulting representations show linear substructures in vector space of words.
We used *glove.840B.300d* embedding, which is trained from Common Crawn data with a vocabulary of 2.2 million and has 840 billion tokens. The embedding layer is of 300 dimensions.

4.2.2 fastText

FastText (Bojanowski et al., 2016) was developed by Facebook in 2016 and it addresses the problem of generalization to unknown words by using parts of words and characters to build the word embeddings. The building blocks are characters instead of words.
We used *wiki-news-300d-1M* embedding, which is trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset with a total of 16 million tokens. The embedding layer is of 300 dimensions.

4.2.3 emoji2vec

emoji2vec (Eisner et al., 2016) was developed by Eisner et al. in 2016 which have pre-trained embeddings for all Unicode emoji which are learned from their description in the Unicode emoji standard.
This embedding maps emoji symbols into the same space as the 300-dimensional Google News word2vec embeddings.
Pre-trained embedding from emoji2vec were used for emojis, and for all other words, embeddings were created from word2vec.

4.3 Different architecture

We decide to try try an architecture that utilizes layers from both convolutional and recurrent neural networks. We felt this could provide an increase in performance over just using networks with CNN or RNN layers.
Regarding the architecture of this model, we start with an embedding layer of size 300 and use a pre-trained embedding (GloVe, fastText, emoji2vec). We then apply a bi-directional LSTM with 300

units and dropout (of 0.35). This is then followed by a 1-dimensional convolutional layer with kernel size of 4x4, and 128 filters. We then apply max pooling with pool size of 4, and follow it with another bi-directional LSTM of size 100. Next, we apply a dense layer of size 128, which is finally followed by a single dense unit with sigmoid activation.
All layers except the final one use relu activation. The model was trained using binary crossentropy loss function, and adam optimizer was used.

5 Results

Throughout this section, we ran each experiment five times and report the best scores of the five attempts. This is done to prevent discarding a good architecture on the basis of one bad run.
Firstly, we check if the following modifications yield any better results over baseline models of CNN, LSTM and GRU:

- Applying the extra preprocessing steps discussed in Section 4.1.
- Using pre-trained GloVe embeddings instead of starting embedding layer with random weights.
- Extra pre-preocessing steps + GloVe embeddings.

	CNN	**LSTM**	**GRU**
Original	0.64	0.71	0.71
Extra pre-processing	0.67	0.73	0.72
GloVe	0.69	0.75	0.75
Extra pre-processing + GloVe	0.70	**0.77**	0.76

Table 2: Comparison of Macro F1 scores on the test set of OLID dataset for models with various pre-trained embedding layers

The results align with our expectations, with the F1 score gradually increasing with addition of each augmentation.

CNN performs much worse compared to LSTM and GRU, so we drop it from further analysis.

Next, we check if there is any difference in performance between freezing the GloVe embedding layer and keeping it trainable. For this, we look at the two Recurrent Neural Network based models- LSTM and GRU, and train them once with the additional preprocessing steps discussed in Section 4.1 along with a frozen pre-trained GloVe embedding layer, and then make the layer trainable while keeping all the other factors same.

	LSTM	**GRU**
Frozen embedding	0.75	0.75
Trainable embedding	**0.76**	0.75

Table 3: Comparison of Macro F1 scores on the test set of OLID dataset for RNN based models with frozen and trainable GloVe embedding layer

While it is not very significant, keeping the embedding layer trainable seems to increases the performance of the models marginally. It seems like a good idea to keep the embedding layers trainable since it gives models the freedom to modify certain weights to further make it easier for them to classify tweets correctly.

In our next experiment, we look at changes in performance of models when using the three pre-trained embeddings listed in Section 4.2. For this, we report scores of LSTM and GRU after applying the additional pre-processing steps and keeping the embedding layers trainable for best results.

	LSTM	**GRU**
Original	0.71	0.71
GloVe	0.76	0.75
fastText	**0.77**	**0.77**
emoji2vec	0.73	0.72

Table 4: Macro F1 scores on the test set of OLID dataset.

fastText embedding gives the best results among the three pre-trained embeddings used here as seen in Table 4. This closely followed by GloVe embeddings, and emoji2vec gives only marginally high F1 score compared to the original models which did not use any pre-trained embedding.

Further, we combine the above embeddings with one another and see if that improves the performance of models in any way. We consider the following combinations for both LSTM and GRU models:

- Concatenate fastText and emoji2vec
- Mean of fastText and emoji2vec

- Concatenate GloVe and emoji2vec
- Mean of GloVe and emoji2vec
- Concatenate GloVe and fastText
- Mean of GloVe and fastText
- Concatenate GloVe, fastText and emoji2vec

	LSTM	GRU
fastText + emoji2vec concat	0.77/0.81	0.76/0.81
fastText + emoji2vec mean	0.76/0.81	0.76/0.81
GloVe + emoji2vec concat	0.77/0.81	0.77/0.82
GloVe + emoji2vec mean	0.76/0.81	0.76/0.80
fastText + GloVe concat	**0.77/0.82**	**0.78/0.82**
fastText + GloVe mean	0.76/0.81	0.76/0.81
fastText + GloVe + emoji2vec, concat	0.77/0.82	0.77/0.81

Table 5: Comparison of Macro F1 / Weighted F1 scores on the test set of OLID dataset for RNN based models with frozen and trainable GloVe embedding layer.

We presented both Macro F1 and Weighted F1 scores to differentiate between the various methods more clearly, since most of the methods have very similar performance. Overall, concatenating embeddings gives better performance than taking their mean, which aligns with our intuition. Interestingly, concatenating all the three pre-trained embeddings does worse than most models. In conclusion, we feel that concatenating different embeddings does not provide any discernible advantage over using fastText embeddings.

Finally, we implement the model discussed in Section 4.3.

	LSTM	GRU	CNN+LSTM
Original	0.71	0.71	**0.72**
GloVe	**0.76**	0.75	**0.76**
fastText	0.77	0.77	**0.78**
emoji2vec	**0.73**	0.72	0.72

Table 6: Macro F1 scores on the test set of OLID dataset.

As shown in Table 6, this new architecture performs marginally better than the other two with all embeddings except for emoji2vec.

6 Conclusion

In conclusion, we have successfully replicated the results from the paper *Towards NLP with DeepLearning: Convolutional Neural Networks and Re-current Neural Networks for Offensive LanguageIdentification in Social Media*. We have also developed a smart baseline method (Section 3.3).
Further, we carried out various experiments to improve the results we obtained while replicating the paper.
Additional pre-processing steps discussed in Section 4.1 were all successful in improving the performance of models. The best configuration was to use an architecture with a combination of CNN and LSTM layers (as discussed in Section 4.3) along with fastText pre-trained embeddings while keeping the embedding layer trainable.

7 Future work

In future, we would like to try a pseudo-labelling technique called tri-learning, where we take three of the best performing models we discussed in section 4 and then train each of the models with labels generated by the other two models on an unseen test dataset.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. 2016. emoji2vec: Learning emoji representations from their description.

Kyokomi. 2021. emoji.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word

representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Jacob Perkins. 2010. *Python text processing with NLTK 2.0 cookbook*. PACKT publishing.

Andrei-Bogdan Puiu and Andrei-Octavian Brabete. 2019. Towards nlp with deep learning: Convolutional neural networks and recurrent neural networks for offensive language identification in social media.

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. In *Proceedings of NAACL*.

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).