SemEval 2020 Task 12:
Task 12: OffensEval 2: Multilingual Offensive Language Identification in Social Media

Zhalgasov Askhat
The Chinese University of Hong Kong
Department of Computer Science and Engineering
and

Rahman Faiyaz
The Chinese University of Hong Kong
Department of Computer Science and Engineering
(Dated: December 10, 2019)

This paper serves as our research project for ESTR3108, Fundamentals of Artificial Intelligence, instructed by Professor Leung Kwong-Sak in the Fall of 2019. The task itself is one of twelve tasks for the semantic evaluation workshop hosted annually. We take a mixed approach in creating models to identify and categorize hate speech in social media. The motivation behind choosing this topic is mainly its low barriers of entry which allows newcomers to gain a good first exposure into research in NLP and also the practical applications of it.

I. INTRODUCTION

Social media has become a very relevant topic in recent times. It provides a platform for free speech, an unsupervised platform initially. Many malicious individuals abuse this opportunity to spread hate and negative sentiments through social media. The lack of supervision in social media intensifies this effect, both in terms of its spread as well as its severity.

Steps have been taken to mitigate this, such as independent moderator teams that review this content. However, this places immense psychological strain on these individuals, and this also lacks scalability.

By using machine learning models, this classification problem can be automated and move the burden of sorting through the potentially hateful content away from human labour. It also poses as an interesting semantic problem that is likely to have applications elsewhere, such as in spam detection etc. This paper documents our approach to build a model for this task, along with the challenges we have faced and overcome.

II. APPROACH

The final models we have built was constructed using Keras, a part of the Tensorflow project by Google. It is a sequential model trained using Google Colab and its architecture included the following features: MaxPooling2D, Convolution2D, LSTM, Reshape, Dropout.

For labelling, we chose to simply use [0,4] as the labels as this is simpler for the model to interpret and also because only five combinations of labeling was possible. The data quality was not excellent as the language used in social media platforms is usually informal and unclear. We chose a few preprocessing techniques to remove noise from the data and also make it more uniform. The main preprocessing steps were Lemmatization, Stop-word removal, Lowercasing, and Tokenization.

The data was then vectorized into a numerical form using a word counting technique, and GloVe, a word embedding library that represented each word with a 200-dimensional vector. The architecture of the neural network will be discussed later on.

III. DESCRIPTION

The goal of OffensEval, or Task 12, is to create machine learning models that can reliable tweets from OLID(Offensive Language Identification Dataset) into categories that become more specific with each subtask.

A. Sub-task A

The task describes the classification of tweets into two broad categories, offensive and not offensive. These are labelled as OFF and NOT respectively.The model takes a tweet as input and predicts the corresponding label of that tweet.

B. Sub-task B

The second task described the classification of offensive tweets into two categories, targeted insult and untargeted. They would be labelled as TIN and UNT respectively.

C. Sub-task C

The final task was the classification of targeted insults into three classes, according to the entity they are slandering. The labels were IND, GRP, and OTH, representing Individual, Group, and Other respectively.

IV. DATA

Tweet
@USER He is so generous
IM FREEEEE!!!! WORST EXPERIENCE
@USER Fuk this fat cock sucker
@USER Figures! Thank God for @USER

TABLE I. Data Sample

A. Dataset

The dataset we used was provided by means of crowdsourcing. In total, it contains 14100 annotated tweets, where 13240 samples were provided for training and rest of 840 were used for model evaluation. However, there were a few difficulties with the dataset, such as the data imbalance, and the lack of contextual information.

B. Data Handling

Prior to model training, we cleaned and preprocessed the data. Preprocessing is a necessary step in NLP because the textual data can not be feed into the model as it is.

C. Text Preprocessing

A tweet usually contains noise that would unnecessarily waste computational power and decrease model performance, therefore noise removal and some normalization was at first applied.

1. User mentions

Since the data is anonymized, tweets contain user mentions in the form @USER. By intuition, we concluded that due to the anonymization, the model can not obtain the context and therefore would add a little value to the model. Such items were removed.

2. Lowercasing

Upon first impression, it is may not be obvious why Lowercasing is necessary. One argument can be that uppercase words bear more aggressive and strong meaning. However, due to the size of the dataset, such instances do not happen frequently, and it adds extra noise to the model. We decided that such trade-off is not worthwhile and applied lowercasing on the dataset.

3. Tokenization

Tokenization splits the tweet into the list of words. Since all preprocessing algorithms work with single words its primary step in any text processing.

4. Stop word removal

Stop word removal removes the words which contribute very little meaning to the context of the tweet. Such words are called stop words, without stop words, the model could have extracted the same subject of the tweet. For example: 'is', 'that', 'the', are all stop words. For such a small dataset, any noise can have a big influence, so we removed words contained in the NLTK library's stop word set.

5. Lemmatization

Lemmatization converts each word into its root form. The NLTK library first tagged words such as noun, adjective, adverb, verb and transformed them to their root, which is called its lemma. We believe that this is a very valuable technique as it employs morphological analysis and allows the model to recognize the same word in different forms, thus reducing noise and increasing model performance.

D. Vectorization

As plain text cannot be fed into the model, the need to represent it in numerical values arised. Vectorization does the aforementioned task.

1. Count vectorizer

Count is the simplest method of mapping the tweet to some vector. It first takes all unique words in the corpus and then calculates the frequency at which these words are present in the tweet. This is also called TF-IDF.

E. Word Embeddings

A word embedding layer is used for vectorization. This word embedding was GloVe and was used because it effectively captured syntactic and semantic information from the words in the tweet. It holds a set of pre-trained word vectors, trained on over 2 billion tweets, containing vector representations for over 1.2 million words in various dimensions (25, 50, 100, 200). It is publicly available on the author's website

V. METRICS

Primary metric we used is macro f1 score as it evaluates the overall performance of the model. However, we kept track of weighted f1 score to better understand how our model works on the current data since it does take into account the unequal class proportion. For tasks with imbalanced data, we believe its also useful to watch this parameter as well. A key advantage of F-1 score is that it is a very fair metric, one that overcomes the limitations of accuracy, precision, recall, and other derivatives.

VI. DEEP LEARNING

We implemented several deep learning architectures including convolutional and recurrent neural networks and tested on all subtasks. While designing our models we looked at past attempts and tried to reimplement as well as improve on them.

A. Architecture

The first model we attempted to implement was the classification phase of Long Short-Term Memory Network(LSTM) based on RNNs. We did some modification so the final architecture looks as follows:

Layer (type)	Output Shape	Param #
embedding_12 (Embedding)	(None, 44, 30)	472380
lstm_11 (LSTM)	(None, 50)	16200
dense_20 (Dense)	(None, 44)	2244
dropout_18 (Dropout)	(None, 44)	0
dense_21 (Dense)	(None, 2)	90

Total params: 490,914
Trainable params: 490,914
Non-trainable params: 0

The second model also was adopted solely by our intuition, we tried to further simplify the

previous model, so one dense layer was removed and drop-out was added after embedding. The idea behind the drop-out layer after embedding is to force the model to drop some words after embedding and therefore prevent the network to depend on certain words, ie, overfitting.

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 44, 30)	472380
dropout_5 (Dropout)	(None, 44, 30)	0
lstm_4 (LSTM)	(None, 100)	52400
dense_7 (Dense)	(None, 2)	202

Total params: 524,982
Trainable params: 524,982
Non-trainable params: 0

Final and the last model we carried out is a two-dimensional convolutional network with max-pooling. We wanted to compare this very popular technique with the RNN network. The architecture was empirically designed and there is no strong reason why we chose this composition. Figure:

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 44, 30)	472380
reshape_1 (Reshape)	(None, 30, 44, 1)	0
conv2d_1 (Conv2D)	(None, 30, 40, 32)	192
dropout_1 (Dropout)	(None, 30, 40, 32)	0
conv2d_2 (Conv2D)	(None, 29, 38, 16)	3088
dropout_2 (Dropout)	(None, 29, 38, 16)	0
conv2d_3 (Conv2D)	(None, 28, 37, 16)	1040
dropout_3 (Dropout)	(None, 28, 37, 16)	0
flatten_1 (Flatten)	(None, 16576)	0
dense_5 (Dense)	(None, 2)	33154

Total params: 509,854
Trainable params: 509,854
Non-trainable params: 0

All models were built with Keras which is user-friendly and enabled us to quickly change the structure and parameters of the model. It also provided an internal embedding layer so the task was partially automated. However, we still need to provide the size of vocabulary for the model and here we used Count vectorizer. We also one-hot encoded the data and labels such that the neural network will not treat them in a certain relation between each other. For the training data, we also padded them with maximum length so that they are equal.

B. Testing and Validation

For testing, we used the test-set provided by organizers on which the actual scores were distributed. Also, we used 70:30 proportion to split the given data into the training and validation set respectively. It needs to mention that stratification was used during splitting, such that sets have approximately the same percentage of each class.

C. Parameters

Across testing, on several subtasks, we slightly changed the parameters such as the number of epochs, batch sizes, as well as changing the dropout coefficients on different layers to get the best result. We also experimented over the shapes of neural net's layers to squeeze more performance. However, throughout the research, the general architecture of neural networks remained the same.

D. Optimizers

We mostly used Adam optimizer. After trying several options including AdaGrad and RMSprop we concluded that Adam works best for our task. For subtask A and B both binary and categorical cross-entropy loss functions were used, for subtask C only categorical cross-entropy loss function was used since there are more than two classes.
During the training, we also tried the early stopping strategies based on loss and accuracy computed at the end of each epoch.

E. Data Imbalance

As you can notice the data imbalance growth with every subtask. Particularly, in subtask B

around 88% of the labels belong to the same class. Firstly, we were thinking of downsampling the data by removing over-represented samples so that the training data becomes more balanced. This turned out not so beneficial as the number of samples becomes even smaller and yielded very bad performance.
Finally, we stopped at class weighting. It allows the under-represented samples to have a greater effect on the loss function. We used sklearn library to automatically compute weights for each class.

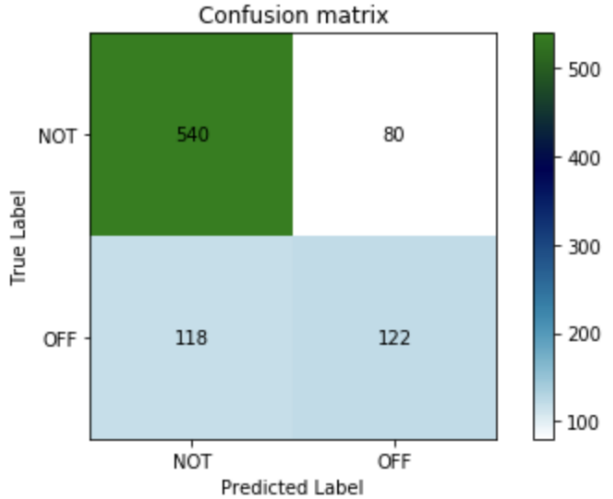
A	B	C	Train	Test	Total
OFF	TIN	IND	2407	100	2507
OFF	TIN	OTH	395	35	430
OFF	TIN	GRP	1074	78	1152
OFF	UNT	—	524	27	551
NOT	—	—	8840	620	9460
ALL			13240	860	14100

TABLE II. Data Labels

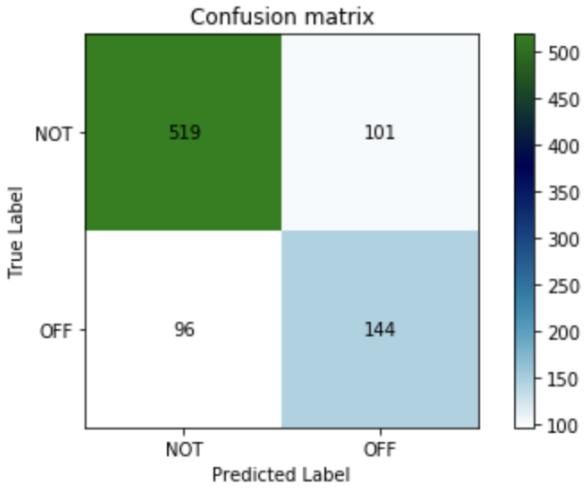
VII. RESULT

A. Task A

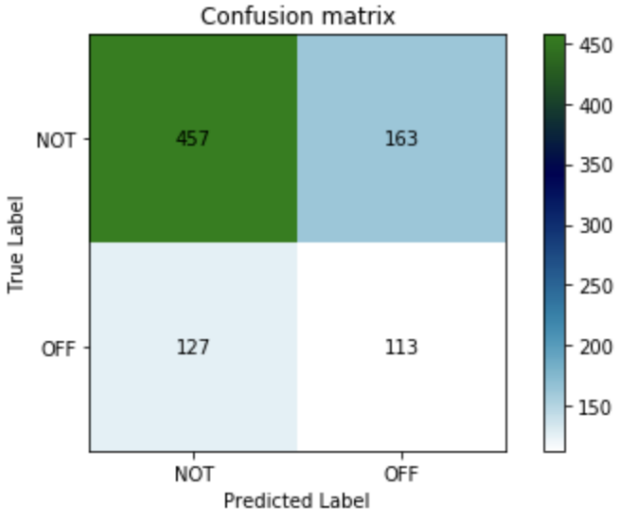
Model 1
Macro F1: 0.70
Accuracy: 0.77
Weighted F1: 0.76



Model 2
Macro F1: 0.72
Accuracy: 0.77
Weighted F1: 0.77



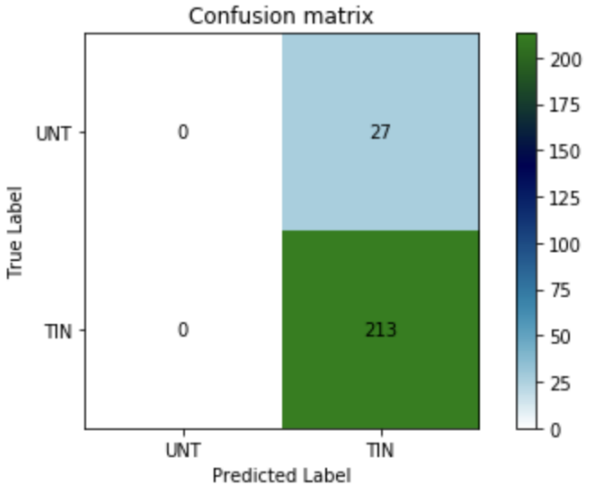
Model 3:
Macro F1: 0.60
Accuracy: 0.66
Weighted F1: 0.67

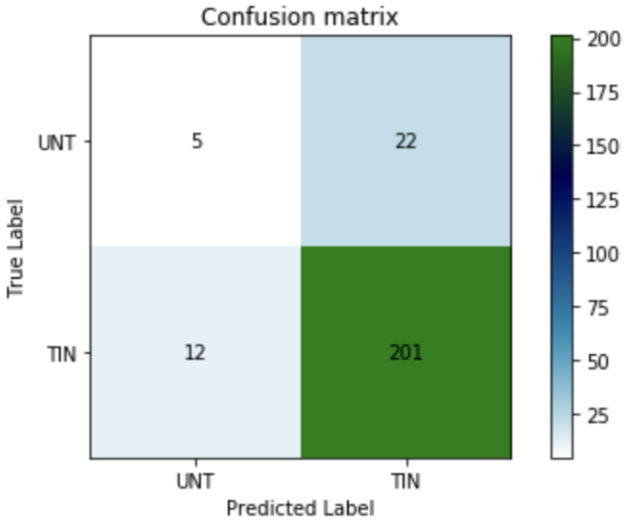


Best Performance: Model 2

B. Task B

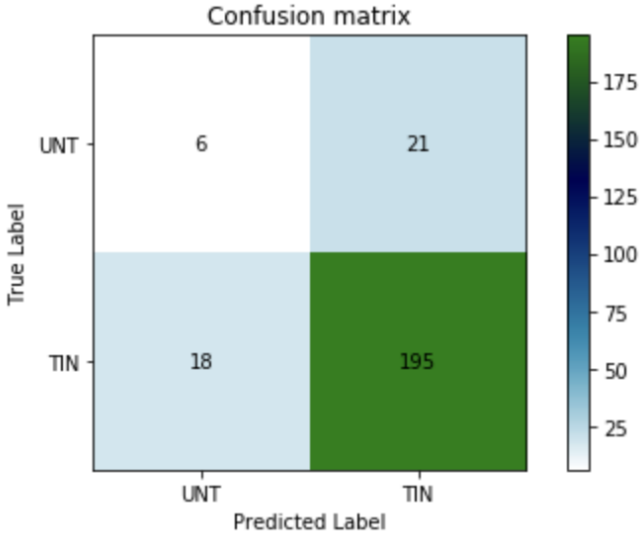
Model 1:
Macro F1: 0.47
Accuracy: 0.89
Weighted F1: 0.83





Best Performance: Model 3 but not significantly better

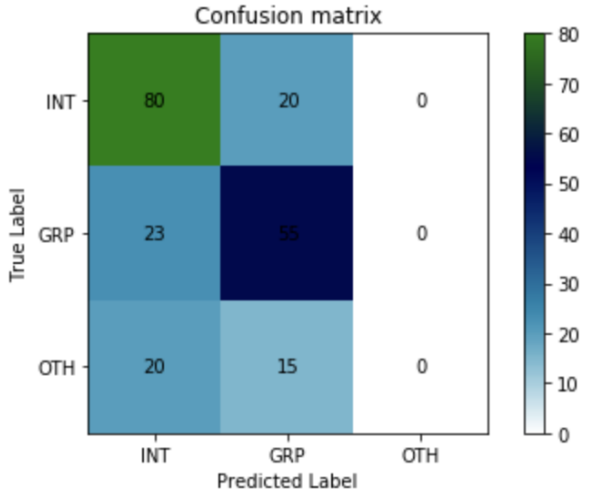
Model 2:
Macro F1: 0.57
Accuracy: 0.84
Weighted F1: 0.83



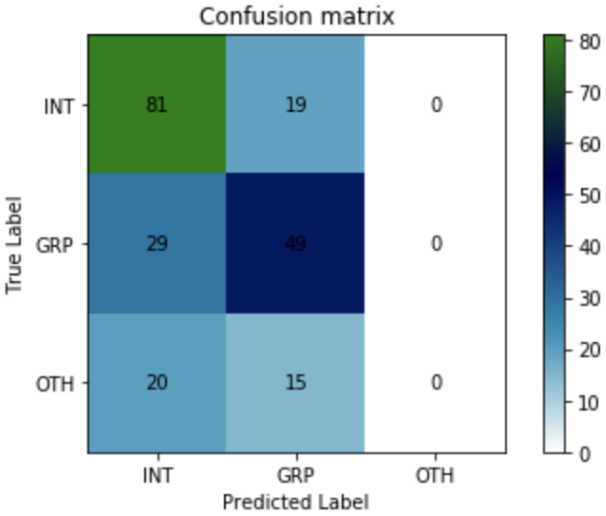
Model 3:
Macro F1: 0.57
Accuracy: 0.86
Weighted F1: 0.84

C. Task C

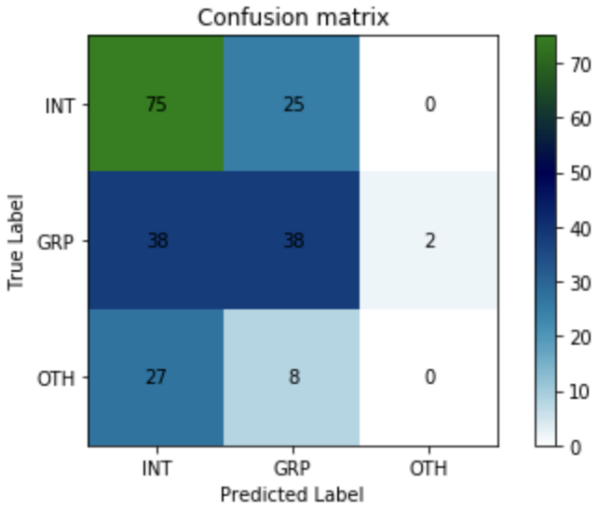
Model 1:
Macro F1: 0.46
Accuracy: 0.63
Weighted F1: 0.48



Model 2:
Macro F1: 0.44
Accuracy: 0.61
Weighted F1: 0.5



Model 3:
Macro F1: 0.38
Accuracy: 0.53
Weighted F1: 0.48



Best Performance: Model 1

VIII. EVALUATION

Results are inconclusive. We can observe that LSTM models performed well in subtask A. However, in subtask B and C, all of our models perform at the same level. During the training, we monitored the loss metric and observed it fluctuated throughout the training. We think that the reason for this is that the model got pushed into several directions and there is no enough data to generalise this conclusion. We believe if we feed better quality data into the LSTM models in subtask B and C, better results could be produced.
Due to the imbalance of data, the over-represented class was often over-predicted. The cause for this can be in the cross-entropy loss function which comes across some local minimum point. The result of the first model on subtask B clearly demonstrates how it can relate the test data to one class. The solution we tried is class weighting, it tries to soften the aforementioned situation. It actually was very helpful in the case with the simpler LSTM model. The normalized confusion matrix for the second model shows how class weighting mitigates the problem.

IX. CONCLUSION AND DISCUSSION

Being complete newcomers to NLP problems, we learned that not so complicated models could achieve surprising results.
Designing the architecture and setting the right parameters manually was very tedious and inefficient. Applying grid search over hyperparameters could be very beneficial in such tasks. Regarding the model, the top performer so far for this task were BERT models. In the future, we hope to use that in our findings and improve onx. Also, the quality and quantity of training sample have a very significant influence on the learning model. Models show very clear over-fitting on subtask B, so the variance is also a key issue. As such, enhancing the dataset by using other sources of labelled data would have helped more. To conclude, a deeper understanding of NLP is required to get better results. On our current level, we were led by our intuition on parameters tuning. We have several leads on what else to pursue for increasing our model's performance and will continue to work on this.

X. REFERENCES

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). In Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval).

Karthik Dinakar, Roi Reichart, and Henry Lieberman. 2011. Modeling the detection of textual cyberbullying. In The Social Mobile Web, pages 11–17

Pete Burnap and Matthew L. Williams. 2015. Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making

Bjorn Gamback and Utpal Kumar Sikdar. 2017. Us- ing convolutional neural networks to classify hatespeech. In Proceedings of the First Workshop on Abusive Language Online.

Paula Fortuna and Sergio Nunes. 2018. A Survey on Automatic Detection of Hate Speech in Text. ACM Computing Surveys (CSUR), 51(4):85.

Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. 2017. Understanding abuse: A typology of abusive language detection subtasks. arXiv preprint arXiv:1705.09899.

Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python. O'Reilly Media.

Ritesh Kumar, Atul Kr Ojha, Shervin Malmasi, and Marcos Zampieri. 2018b. Benchmarking aggression identification in social media. In Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018), pages 1–11.

Mai ElSherief, Vivek Kulkarni, Dana Nguyen, William Yang Wang, and Elizabeth Belding. 2018. Hate Lingo: A Target-based Linguistic Analysis of Hate Speech in Social Media. arXiv preprint arXiv:1804.04257.

N. Djuric, J. Zhou, R. Morris, M. Grbovic, V. Radosavljevic, and N. Bhamidipati. 2015. Hate speech detection with comment embeddings. In ICWWW, pages 29–30.