Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, pages 91–96
with a Shared Task on Offensive Language Detection.
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Combining Character and Word Embeddings for the Detection of Offensive Language in Arabic

Abdullah I. Alharbi[1,2], **Mark Lee**[1]
[1]School of Computer Science, University of Birmingham, Birmingham, UK
[2]Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, KSA
{aia784, m.g.lee}@cs.bham.ac.uk
aamalharbe@kau.edu.sa

Abstract

Twitter and other social media platforms offer users the chance to share their ideas via short posts. While the easy exchange of ideas has value, these microblogs can be leveraged by people who want to share hatred. and such individuals can share negative views about an individual, race, or group with millions of people at the click of a button. There is thus an urgent need to establish a method that can automatically identify hate speech and offensive language. To contribute to this development, during the OSACT4 workshop, a shared task was undertaken to detect offensive language in Arabic. A key challenge was the uniqueness of the language used on social media, prompting the out-of-vocabulary (OOV) problem. In addition, the use of different dialects in Arabic exacerbates this problem. To deal with the issues associated with OOV, we generated a character-level embeddings model, which was trained on a massive data collected carefully. This level of embeddings can work effectively in resolving the problem of OOV words through its ability to learn the vectors of character n-grams or parts of words. The proposed systems were ranked 7th and 8th for Subtasks A and B, respectively.

Keywords: character-level embeddings, word-level embeddings, Arabic offensive language detection

1. Introduction

Microblogging platforms, such as Twitter, offer users a channel through which they can share ideas and opinions via short messages. In the case of Twitter, these are known as tweets. While social media channels can be used for constructive purposes, they can also be exploited by people who wish to share their hatred. These people can share their negative views about an individual, a race or a group with millions of people at the click of a button. To counteract this, there is a significant need to develop an effective method that will automatically identify messages containing offensive language or hate speech. Much research has been undertaken on detecting offensive language in English, but as yet, little research has focused on the detection of offensive language in Arabic (Mubarak and Darwish, 2019).
To contribute to the development of this area, a shared task on 'Arabic offensive language detection' was conducted at the OSACT4 workshop (Mubarak et al., 2020). This shared task included two subtasks: Subtask A was to detect offensive language, and Subtask B was to detect hate speech. These subtasks shared a dataset of 10,000 tweets that comprised an Arabic offensive language dataset. The challenges of this shared task included the following:

(i) The distribution of the targeted classes was imbalanced in both subtasks. However, Subtask B is more challenging than Subtask A as only 5% of the tweets were labelled as hate speech and fell under Subtask B, 19% of the tweets are labelled as offensive and included in Subtask A.

(ii) The language used on social media has unique characteristics, such as sentences that are grammatically incorrect, and the use of symbols and emojis resulting in an out-of-vocabulary (OOV) problem.

(iii) This problem becomes even more challenging when considering that Arabic social media users employ various dialects and sub-dialects in their communication. In contrast to Modern Standard Arabic (MSA), the forms of dialectical Arabic vary widely, and there is a general lack of rules and standards (Salameh et al., 2018).

To deal with the issues associated with OOV, we generated a character-level (char-level) embeddings model, which was trained on a massive carefully collected dataset. This level of embeddings can work effectively in resolving the problem of OOV words through its ability to identify the vectors of character n-grams or parts of words. Our proposed systems were ranked 7th and 8th for Subtasks A and B, respectively.
The rest of this paper is organized in the following manner. Section two provides a brief overview of current literature on hate-speech and offensive speech detection. Section three includes an overview of the methodology, including the experimental setup for our proposed systems. Section four provides an evaluation and analysis of results and some discussion. Finally, section five concludes the paper with suggestions for future work.

2. Related Work

The use of offensive language and hate speech in the English language has been investigated widely, and various categories of hate speech have been identified, including sexism, religious hate speech and racial hate speech (Davidson et al., 2017; Malmasi and Zampieri, 2017; Kumar et al., 2018; Waseem et al., 2017; Zampieri et al., 2019). In contrast, limited studies have been done in this area in the Arabic language (Al-Hassan and Al-Dossari, 2019).

One of the earliest works on the detection of offensive language in Arabic was by Mubarak et al. (2017). They argued that some users have a higher likelihood of using offensive language than others. They used this insight to construct a list of Arabic words that are offensive. They subsequently developed an extensive corpus of Arabic tweets that were annotated manually into three categories: clean, obscene and offensive. Another contribution was made by Alakrot et al. (2018) who developed a corpus of offensive Arabic comments that had been shared on YouTube. This created a dataset that includes 16K Egyptian, Libyan and Iraqi comments, categorised into one of three classes: offensive, inoffensive and neutral. They trained a Support Vector Machine (SVM) classifier to detect the offensive comments. Based on their experiments, they concluded that using the N-gram feature improved the classifier's accuracy, while a combination of N-gram and stemming reduced the performance of the system. Albadi et al. (2018) focused on the detection of religious hate speech in Arabic but did not consider any other forms of hate speech. They constructed and scored a lexicon of the most frequently used religious hate terms and tested a variety of classifiers for their study.
More recently, Mubarak and Darwish (2019) extended the list of offensive words and used it to build a massive training corpus for automatic offensive tweet detection. They employed a character-level deep learning algorithm to classify each tweet as to whether or not it was offensive. In our work, we combined different levels of word embedding (character and word levels) and incorporated these into a supervised learning framework for the task of detecting offensive and hate speech tweets.

3. Data and Methodology

3.1. Data Description

The data released by the organisers included two sub-tasks: Subtask A (detecting offensive language) and Subtask B (detecting hate speech). The subtasks shared a common dataset of 10,000 tweets containing offensive language in Arabic. For Subtask A, the tweets were manually annotated using the term 'OFF' for offensive tweets and 'NOT_OFF' for tweets that were not offensive. In Subtask B, tweets were identified by 'HS' for hate speech and 'NOT_HS' for all other cases. An overview of the dataset is provided in Tables 1 and 2.

Dataset/Class	OFF	NOT_OFF	Total
Training	1410	5590	7000
Dev	179	821	1000
Test	402	1598	2000

Table 1: Distribution of classes in Subtask A

Dataset/Class	HS	NOT_HS	Total
Training	361	6639	7000
Dev	44	956	1000
Test	101	1899	2000

Table 2: Distribution of classes in Subtask B

3.2. Preprocessing

We followed the procedure described by a number of researchers (Abu Farha and Magdy, 2019; Duwairi and El-Orfali, 2014), which involves the following steps:

- **Cleaning:** All unknown symbols and other characters are eliminated. For example, other language letters, diacritics, punctuation, etc. However, emojis are not removed and each emoji is represented by a vector as same as words.
- **Normalisation of letters:**Letters which appeared in different forms in the original tweets were rendered into a single form. For example, the "hamza" on characters {أ,أ} was replaced with {ا}, and the 't marbouta' {ة} was replaced with {ه}.
- **Segmenting {يا} phrases:** One of the most common phrases used in Arabic offensive language is the phrase that begins with (ya), followed by an offensive word. A large number of writers on social media do not use a space between these two words, so they will be recognized as one word. This issue cannot be handled even by state-of-the-art tools such as MADAMIRA (Pasha et al., 2014) and Farasa (Abdelali et al., 2016). We therefore treated this situation by using RegEx to segment any strings starting with (ya) into two words. However, this approach needs to improved in our future works to treat words such as Yasser or Yassin.

3.3. Embedding Models

Word embedding is one of the most important methods that have been applied recently to many natural language processing tasks (Devlin et al., 2014; Zhang et al., 2014; Lin et al., 2015; Bordes et al., 2014). Word embeddings are learned representations of text, with words of similar meanings represented in similar ways. An essential element of this methodology is the concept of employing densely distributed representations for every word. Every word is encoded to a real-valued vector with a few hundred dimensions. We employed different levels of word embedding models, which are detailed in the following subsections. Table 3 presents a summary of important information about each of these models including their sizes and pre-trained corpus.

Model	Corpus	Size
Ara2Vec	General - Twitter	77M tweets
Mazajak	Sentiment - Twitter	250M tweets
Our Model	Emotion - Twitter	10M tweets

Table 3: Different pre-trained Arabic word embeddings used for our systems

3.3.1. Word-level Embeddings

We used two Arabic pre-trained word embeddings: Ara2Vec (Soliman et al., 2017) as well as Mazajak

(Abu Farha and Magdy, 2019). One of the largest open-source word embeddings is Ara2Vec, consisting of six different word embedding models for the Arabic language. The researchers derived the training data from three separate sources: the Wikipedia, Twitter and Common Crawl web-pages crawl data. They employed two word-level models to learn word representations for general NLP tasks. In addition, we used Mazajak, which is considered the largest word-level embeddings. They used 250M Arabic tweets to generate a language model. Although these models are trained on a large number of words, they cannot capture all words that can be encountered in the real world. Due to OOV, the inability to identify words is one of the main limitations of this word-level model.

3.3.2. Char-level Embeddings

As mentioned in the introduction, the form of dialectical Arabic words used varies widely, which leads to the OOV problem. Therefore, effective resources and tools are needed to better understand and treat these various linguistic forms when targeting offensive language in Arabic tweets. Our main intuition is that while word-Level embeddings seems to give more importance to the semantic similarity, char-level embeddings are more likely to encode all variants of a word's morphology closer in the embedded space. Table 4 shows an example of offensive Arabic word, where the similarity of these words is mostly based on morphology for the char-level and semantics for the word-level. Therefore, combining these two different levels of embeddings into a supervised learning framework for the task of detecting offensive tweets can improve the results.
To learn the morphological features found within each word, we utilised FastText, a character n-grams model (Bojanowski et al., 2017). FastText can learn the vectors of character n-grams or word parts. Therefore, this feature enables the model to capture words that have similar meanings but have different morphological word formations. To train this model, we used an in-house unlabelled Arabic dataset (consisting of 10 million tweets) [1]. This large dataset contains varied sentiment and emotional words expressed in different Arabic dialects. We used these data not only because of their variety of Arabic dialects but also because we believe a correlation exists between negative emotions and offensive language words. We used the Gensim library [2] to implement the FastText method. Gensim is an open source Python library for natural language processing, which supports an implementation of different word embeddings, including FastText. The input for this char-level model was a composed of n-grams for each word in a given tweet. For example: the word (mnHTyn) will be treated as composed of 3-grams: '<mn','mnh',...,'yn>'. The '<' and '>' are special symbols which are appended to indicate the token start and end. We used the following parameters: 300 for size, five for the windows context and three to ignore words that had a total frequency of less than three. In addition, to control the length of character n-grams, we used three and six. We released our generated char-level model to be used as a pre-trained language model for applications and research relying on Arabic NLP.

Example of an offensive query term: (mnHT)منحط	
Mazajak	Our char-level model
(wmnHT) ومنحط	(wmnHT) ومنحط
(q*r) قذر	(mnHTh) منحطه
(wq*r) وقذر	(wmnHTh) ومنحطه
(mtxlf) متخلف	(AlmnHT) المنحط
(wwqH) ووقح	(mnHTyn) منحطين

Table 4: The top five most similar words to a given query term using char and word level embeddings.

3.4. Classification Models

3.4.1. Logistic Regression

We selected logistic regression (LR) as our baseline model in order to investigate the lower bound performance that we should compare. We performed a number of experiments to compare the impact of the preprocessing techniques mentioned in section 3.2. We also used LR to compare the impact of using different features such as emojis, URL and user tags, and the combination of these features. We then reported the results with the highest scores to consider using them with other training models.

3.4.2. XGBoost Classifier

The XGBoost learning model (Chen and Guestrin, 2016) is frequently employed in different situations because it performs extremely well despite substantial challenges. This is an algorithm of decision trees in which new trees correct errors of those trees which are already part of the model. Trees are added to the model until no further changes can be made. We inputted tweet vector representations obtained from an average of real-value word vectors for every word with matching vector representations derived from the pre-trained embeddings.

3.4.3. Deep Learning Model

For the deep learning model, we utilized the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) which is an enhanced form of the recurrent neural network. It is able to tackle various problems and provide robust solutions, for example, to the vanishing gradient problem. The internal structure of the LSTM consists of four layers that interact with each other. These four layers can be described as Forget Gate, Input Gate, Modulation Gate and Output Gate. In order to achieve superior performance, we combined all word embedding models (see section 3.3), and these were then used to initialize the weights of the embedding layer. The weights of the embedding layer were then updated during training to be fine-tuned to each subtask. This was then connected to the rest of the layers in the networks. This embedding layer was followed by two layers each of 1-D convolutions with kernel size 3 and 128 filters. This was followed by two layers of the LSTM network, with 256 and 128 filters, respectively. In

[1]https://github.com/aialharbi/ACWE
[2]https://radimrehurek.com/gensim/models/fasttext.html

the LSTM layers, 0.2 dropouts were induced, and 0.2 recurrent dropouts were employed. Finally, a dense layer with one output was introduced by exploiting sigmoid as an activation function. For all other layers of the network, the ReLU activation function was utilized. We used the Adam optimizer as an optimization function for the network.

4. Experiment Results

The evaluation metrics for this shared task is evaluated by using Macro-F1. From our experiments on the Dev dataset of Subtask A, we selected the deep learning model as the first system and XGBoost algorithm as the second system. However, in Subtask B, the deep learning model performed poorly compared to XGBoost algorithm, therefore we swapped the proposed systems for this Subtask.One possible reason for this low performance is due to the significant imbalanced classes in Subtask B.
The results of subtask A are presented in Table 5. It can be seen that our model 1 provided an F1-score of 0.89 for the Dev dataset and 0.87 for the Test dataset with high precision and recall rates of 0.90 and 0.87, and 0.90 and 0.85, respectively. Similarly, model 2 produced an F1-score of 0.87 with 0.89 precision and 0.85 recall for the Dev dataset, and an F1 score of 0.85, with a precision of 0.88, and a recall of 0.84 for the Test dataset.

System	F1	Accuracy	Precision	Recall
Dev dataset				
1	0.885	0.935	0.901	0.871
2	0.870	0.928	0.895	0.849
Test dataset				
1	0.868	0.920	0.896	0.847
2	0.857	0.913	0.877	0.841

Table 5: Results for both systems with Dev and Test dataset for the Subtask A

The results of Subtask B are presented in Table 5. It can be seen that our model 1 provided an F1 score of 0.71 for the Dev dataset and a 0.74 F1 score for Test dataset with high precision and recall rates of 0.81 and 0.65, and 0.86 and 0.68, respectively. Similarly, model 2 produced an F1-score of 0.49 which is a little less, with 0.48 for precision and 0.50 for recall for the Dev dataset, while with the Test dataset, the F1 score was 0.49, with 0.47 precision, and 0.5 for recall. The confusion matrix of three sub-tasks are shown in fig 1 and 2, which is another way to explain the results discussed above.
Moreover, we evaluated the use of three pre-trained word embeddings: two open-source word-level models and our generated char-level model. We compared the performance of these models individually and by combining all of them. Our char-level model and Mazajak obtained an F1-score of 0.87, outperforming Ara2Vec (0.86). Although our generated model trained only on 10 million tweets, it achieved the same result as Mazajak, which trained on 250 million tweets. However, combining all these different levels of models improved the results by about 2%. We believe that with this combination, we take advantage of these large

System	F1	Accuracy	Precision	Recall
Dev dataset				
1	0.706	0.963	0.818	0.655
2	0.489	0.956	0.478	0.5
Test dataset				
1	0.742	0.963	0.864	0.685
2	0.487	0.950	0.475	0.5

Table 6: Results for both systems with Dev and Test dataset for the Subtask B

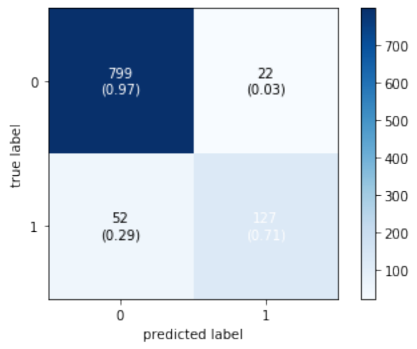

Figure 1: Confusion Matrix for system 1 for Sub-task A.

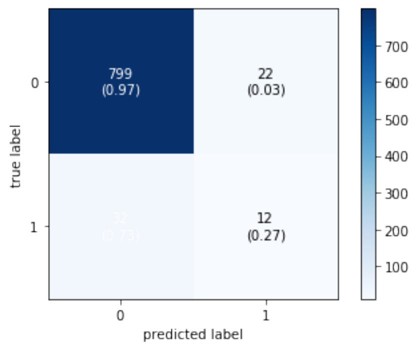

Figure 2: Confusion Matrix for system 1 for Sub-task B.

pre-trained word embeddings (Mazajak and Ara2Vec), and we also overcome their limitation by incorporating our char model to deal with the OOV problem. An example of OOV taken from the dataset of this shared task, an offensive word (الكلبوبه - Alklbwbh), meaning the small female dog, could not be realised by both aforementioned pre-trained word embeddings. However, our char-level model was able to capture its meaning by encoding this word close to other related words that either have the same semantic meaning or mostly a different form of this word.

5. Conclusion

In this work, we generated a character-level embeddings model, which was trained on a massive carefully collected dataset. We incorporated this model with other pretrained word embeddings into a supervised learning framework for the task of detecting offensive and hate speech tweets.

While the macro averaged F1 score for the majority baseline was 0.444 (given by the organisers), we achieved almost double this score. In future works, we hope to improve our results by applying more preprocessing techniques and exploiting a list of offensive language words. Additionally, we will investigate different methods to augment data into our training datasets to make them more robust.

6. Bibliographical References

Abdelali, A., Darwish, K., Durrani, N., and Mubarak, H. (2016). Farasa: A fast and furious segmenter for Arabic. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 11–16, San Diego, California, June. Association for Computational Linguistics.

Abu Farha, I. and Magdy, W. (2019). Mazajak: An online Arabic sentiment analyser. In *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, pages 192–198, Florence, Italy, August. Association for Computational Linguistics.

Al-Hassan, A. and Al-Dossari, H. (2019). Detection of hate speech in social networks: a survey on multilingual corpus. In *6th International Conference on Computer Science and Information Technology*.

Alakrot, A., Murray, L., and Nikolov, N. S. (2018). Towards accurate detection of offensive language in online communication in arabic. *Procedia Computer Science*, 142:315 – 320. Arabic Computational Linguistics.

Albadi, N., Kurdi, M., and Mishra, S. (2018). Are they our brothers? analysis and detection of religious hate speech in the arabic twittersphere. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 69–76, Aug.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Bordes, A., Chopra, S., and Weston, J. (2014). Question Answering with Subgraph Embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Davidson, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Eleventh international aaai conference on web and social media*.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1370–1380.

Duwairi, R. and El-Orfali, M. (2014). A study of the effects of preprocessing strategies on sentiment analysis for arabic text. *Journal of Information Science*, 40(4):501–513.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November.

Kumar, R., Ojha, A. K., Malmasi, S., and Zampieri, M. (2018). Benchmarking aggression identification in social media. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 1–11, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Lin, C.-C., Ammar, W., Dyer, C., and Levin, L. (2015). Unsupervised POS Induction with Word Embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1311–1316.

Malmasi, S. and Zampieri, M. (2017). Detecting hate speech in social media. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 467–472, Varna, Bulgaria, September. INCOMA Ltd.

Mubarak, H. and Darwish, K. (2019). Arabic offensive language classification on twitter. In *International Conference on Social Informatics*, pages 269–276. Springer.

Mubarak, H., Darwish, K., and Magdy, W. (2017). Abusive language detection on arabic social media. In *Proceedings of the First Workshop on Abusive Language Online*, pages 52–56.

Mubarak, H., Darwish, K., Magdy, W., Elsayed, T., and Al-Khalifa, H. (2020). Overview of osact4 arabic offensive language detection shared task. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools (OSACT)*, volume 4.

Pasha, A., Al-Badrashiny, M., Diab, M., El Kholy, A., Eskander, R., Habash, N., Pooleery, M., Rambow, O., and Roth, R. (2014). MADAMIRA: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1094–1101, Reykjavik, Iceland, May. European Language Resources Association (ELRA).

Salameh, M., Bouamor, H., and Habash, N. (2018). Fine-grained Arabic dialect identification. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1332–1344, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Soliman, A. B., Eissa, K., and El-Beltagy, S. R. (2017). Aravec: A set of arabic word embedding models for use in arabic nlp. *Procedia Computer Science*, 117:256–265.

Waseem, Z., Davidson, T., Warmsley, D., and Weber, I. (2017). Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pages 78–84, Vancouver, BC, Canada, August. Association for Computational Linguistics.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., and Kumar, R. (2019). SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86, Minneapolis, Minnesota, USA, June. Association for Computational Linguistics.

Zhang, J., Liu, S., Li, M., Zhou, M., and Zong, C. (2014). Bilingually-constrained phrase embeddings for machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 111–121.