Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, pages 61–65
with a Shared Task on Offensive Language Detection.
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020
© European Language Resources Association (ELRA), licensed under CC-BY-NC

ALT Submission for OSACT Shared Task on Offensive Language Detection

Sabit Hassan[1], Younes Samih[1], Hamdy Mubarak[1], Ahmed Abdelali[1]
Ammar Rashed[2], Shammur Chowdhury[1]
[1]Qatar Computing Research Institute, [2] Özyeğin University
{sahassan2, ysamih, hmubarak, aabdelali}@hbku.edu.qa
ammar.rasid@ozu.edu.tr, shchowdhury@hbku.edu.qa

Abstract

In this paper, we describe our efforts at OSACT Shared Task on Offensive Language Detection. The shared task consists of two subtasks: offensive language detection (Subtask A) and hate speech detection (Subtask B). For offensive language detection, a system combination of Support Vector Machines (SVMs) and Deep Neural Networks (DNNs) achieved the best results on development set, which ranked 1st in the official results for Subtask A with F1-score of 90.51% on the test set. For hate speech detection, DNNs were less effective and a system combination of multiple SVMs with different parameters achieved the best results on development set, which ranked 4th in official results for Subtask B with F1-macro score of 80.63% on the test set.

Keywords: Offensive Language Detection, Hate Speech Detection, Arabic Shared Task

1. Introduction

Detecting offensive language or hate speech on social media has gained a lot of interest recently. Use of offensive language and hate speech on social media can be an indication of hate crimes, toxic environment or level of antagonism against individuals or particular groups. Detecting offensive language and hate speech can also help in filtering out inappropriate content for users. Although there is a lot of research on detecting offensive language and hate speech (Agrawal and Awekar, 2018; Djuric et al., 2015; Davidson et al., 2017), work on Arabic offensive language detection is still in its early stages with very few notable works (Mubarak and Darwish, 2019; Mubarak et al., 2017; Albadi et al., 2018; Alakrot et al., 2018). Mubarak and Darwish (2019) report that only 1-2% of the tweets are offensive. The highly skewed distribution of data makes it extremely difficult to build useful datasets and effective systems. OSACT4 shared task (Mubarak et al., 2020) presents the problem of detecting offensive language and hate speech in Arabic tweets to the community. The shared task consists of 2 subtasks: offensive language detection (subtask A), and hate speech detection (subtask B).

This paper describes the systems submitted for OSACT4 shared task on Offensive Language Detection by the team ALT. First, we experimented with classical machine learning classifiers such as Support Vector Machines (SVMs) that are trained on character and word-level features. Then, we experimented with Deep Neural Networks (DNNs) and Bidirectional Encoder Representations from Transformers (BERT). SVMs were seen to outperform the DNNs and BERT. Since we expect the different kinds of classifier to make different kinds of errors, we take the most promising and diverse individual classifiers and perform voting to decide the final output. Majority voting on SVMs, DNNs and BERT yielded better results than individual systems for subtask A on the development set. The best results on development set for subtask B were obtained by combining the output of different SVMs and considering an instance to be hate speech if any of the classifiers voted it to be hate speech.

In section 2, we describe the dataset and the tasks, in section 3, we describe our approach and compare results for subtask A, in section 4, we describe our approach and compare results for subtask B and in section 5, we provide conclusion of our work.

2. Dataset and Task Description

In this section, we describe the dataset provided to the participants and the two subtasks.

2.1. Dataset

The dataset for OSACT Shared task consists of 10,000 Arabic tweets that are tagged for offensiveness and hate speech. The 10,000 tweets are split into training, development and testing sets as shown in Table 1.

Train	Dev	Test
7,000	1,000	2,000

Table 1: Training, development and testing data split

The organizers note that the data is highly skewed. Only 19% of the tweets are tagged as offensive and 5% of the tweets are tagged as hate speech. Table 2 shows examples from the training set. If a tweet has offensive language (insults or threats) targeting a group of people based on their origin (nationality, race, or ethnicity), their ideology (religion, political affiliation, etc.), gender or any other common characteristics, this is considered as hate speech, so all hate speech tweets are offensive according to this definition.

2.2. Task Description

The participants were required to produce labels indicating if a tweet is offensive (subtask A) or hate speech (subtask B). Each tweet took one of these labels for subtask A: ”OFF” (offensive) or ”NOT_OFF” (not offensive), and ”HS” (Hate Speech) or ”NOT_HS” (not Hate Speech) for subtask B. The full train and development data were made

Tweet	Offensive?	Hate Speech?
فدوه يا بخت فدوه يا زمن واحد منكم يحببه	NO	NO
يا شيخ المعرضين يا آفاق يا نتن !! ربنا ينتقم منك ...	YES	NO
..يا وقح يا كلب الصهاينة يا عميل الفرس أنت عبد من عبيد نعال الحكام الظلمة	YES	YES

Table 2: Examples from the dataset

available to the participants at the beginning of the shared task. At a later stage, only the tweets from test split were made available to the participants via Codalab. The test labels remained unseen by the participants throughout the shared task. The participants were evaluated on the test labels produced by their systems in a blind test phase on Codalab — without access to the participants' own scores or other participants' scores. As the labels in each subtask are not balanced, the macro-averaged F1 scores were used for official ranking.

3. Subtask A: Offensive Language Detection

In this section, we discuss our approach in Subtask A. First, we describe the preprocessing step. Then, we describe the different models we experimented on and compare results of different models. Lastly, we perform an error analysis to understand the limitations of our models and the dataset.

3.1. Preprocessing Tweets

Preprocessing the tweets is an important step as the data from social media can be quite noisy as they contain a lot of emojis, text in mixed languages, excessive use of punctuation etc. It is important to note that some of our models (described in the next subsection) use pretrained word embeddings as feature. In order to reduce noise and be able to find more words in the embeddings, we perform the following steps for preprocessing the tweets.
Step 1: Remove all words that contain non-Arabic characters.
Step 2: Remove all diacritics.
Step 3: Remove all punctuation.
Step 4: Replace repeated characters with only one.

In our initial experiments, we noticed that the settings listed above produces the best results. Therefore, we keep the same preprocessing settings for all experiments.

3.2. System Descriptions

We experimented on a myriad of classifiers including classical machine learning classifiers such as Support Vector Machines (SVMs), Logistic Regression (LR), Multinomial Naive Bayes (MNB), and deep learning classifiers such as Feed-forward Neural Network (FFNN), Convolutional Neural Network (CNN), Bidirectional Long Short Term Memory (BiLSTM), and Bidirectional Encoder Representations from Transformers (BERT). LR and MNB performed quite poorly, and therefore, excluded from the discussion that follows. The results on development set for rest of the classifiers are discussed next and are summarized in table 3. Note that the values for precision, recall and F1 are macro averaged and F1-macro is the official metric.

3.2.1. SVMs

As the features for SVMs, we transform the tweets into bag-of-n-grams vector weighted with logarithmic term frequencies (tf) multiplied with inverse document frequencies (idf). We created both character and word n-grams this way. We experimented with different ranges of character and word n-grams. We also experimented on using Mazajak embeddings (Abu Farha and Magdy, 2019) as features to SVM. Mazajak embeddings were trained on Twitter data, which matches the domain of data in our task. Therefore, we expect it to be more useful compared to other embeddings that are trained on different domain of data (BERT-Multilingual, for example). From table 3, we can see that the best results were obtained when character $[1,5]$ gram and word $[1-3]$ gram features were combined with pretrained Mazajak word embeddings.

3.2.2. FastText

FastText is an efficient deep-learning based system for learning embeddings and performing text classification (Joulin et al., 2016). Since the task of offensive language detection is a text classification problem, we experimented with FastText, but as we can see from Table 3, FastText was outperformed by other systems.

3.2.3. FFNN

For the FFNN architecture, we use four hidden layers with a different number of units (1000, 500, 500, 100) and a sigmoid activation function in each layer, followed by an softmax output layer. To train the network, we use a batch size of 256, with maximum epochs of 50 and early stopping using a 10% of the training set with similar distribution of the labels. The models are then optimized using rmsprop optimization function. The parameter were initialized with small random numbers, sampled from a uniform distribution. Since other systems outperform FFNN, we have not tuned the architecture for different parameters such as number of hidden units or the learning rates, among others.

3.2.4. CNN-BiLSTM

We use two sets of features for this model. First, we have pretrained word embedding (Mazajak embeddings) features for each word. Second, we use CNN as character-level feature extractor. First layer of the network is used to project the input string to character embeddings, which is then passed through a convolutional layer and max-over-time pooling is applied to obtain fixed length representation of words. Character-level representations have been shown to capture morphology of words (Kim et al., 2015). If we were to use only word level features obtained from pretrained word embeddings, we would lose out information when a word does not appear in the vocabulary of the pre-

No.	Model	Features	Acc.	Prec.	Recall	F1
1	SVM	Word [1-3] Gram	80.1	71.6	82.4	73.7
2		Char [1-5] Gram	91.5	78.9	90.8	83.3
3		Mazajak SG-250	92.2	85.5	90.2	87.6
4		Mazajak SG-100 + char[1-5] + word[1-3]	93	88.3	87.7	88
5		Mazajak SG-250 + char[1-5] + word[1-3]	94.3	89.8	91	90.4
6	FastText		89.5	85.5	75.7	79.4
7	FFNN	Char [1-8] + Word [1-4]	91.9	87.1	84	85.1
8	CNN	Mazajak SG-250	92.1	86.7	86.2	86.5
9	CNN-BiLSTM	Mazajak SG-100	92.9	87.6	88.1	87.8
10	M-BERT		90.1	83	83.7	83.3
11	**Ensemble(4+5+7+8)**		**94.3**	**89.9**	**91.1**	**90.5**

Table 3: Comparison of different systems submitted to subtask A

Tweet	True Label	Majority Label	Reason
چايمي خط احمر يا دينيريس يا بنت الاحبة انتي#GameofThrones	OFF	NOT	Out of context
# غلاق_حسابات_البدون_في_البنوك الحل رصاصه يا فيك يا فيه نقطه اخر السطر .	OFF	NOT	Out of vocab
يا جبران يا ابن جرجي وقف عند حدّك بقى كتير زوّدتا بطل في شي تسرقو شكّيت على جيبة المواطن الفقير يلي بيفلح كل لنهار ليطلع	NOT	OFF	Ambiguity

Table 4: Error analysis of subtask A

trained embedding. The character-level features obtained this way will provide us with information in such cases.
The character level representation of words are then merged with the word embedding features and passed to the BiLSTM. Then a softmax layer is used for projecting the probability distribution of the target classes. Note that the CNN learns the character-level features jointly with the BiLSTM, minimizing the cross entropy loss. Stochastic Gradient Descent (SGD) optimizer was used during the training.

3.2.5. BERT

Deep contextualized language models such as BERT. (Devlin et al., 2019) have been shown to perform really well in many NLP tasks. We fine-tuned BERT-multilingual (referred to as M-BERT) for offensive language detection. M-BERT is pretrained on Wikipedia text from 104 languages that include Arabic. From Table 3, we can see that M-BERT is outperformed by CNN-BiLSTM. We speculate that this is because the domain of data M-BERT is trained on, Wikipedia text, is quite different from the Twitter data used in the shared task. Articles on Wikipedia are typically written in a formal way and follow the structure and rules of grammar. Text on social media platforms such as Twitter, on the other hand, can be very informal and chaotic.

3.2.6. Ensemble Method

From table 3, we can see that several system are promising and perform quite close to each other. Since these systems are quite diverse (SVMs are quite different from BERT, for example), they are likely to make different types of errors. In order to improve our results, we experiment on combining multiple systems. We took the output of two SVM classifiers (No.4 and No.5), the CNN-BiLSTM (No.9) and M-BERT (No.10) and performed majority voting to decide on the label for each input. From table 3, we can see that this ensemble method is the best system on the development data and this is the system we submitted for official ranking. The official scores for this system on the test set is shown in table 5.

Official Rank	Acc.	Prec.	Recall	F1
1st	93.85%	90.18%	90.85%	90.51%

Table 5: Official results on subtask A test set

	NOT OFF	OFF
NOT OFF	789	32
OFF	25	154

Table 6: Confusion matrix on subtask A dev set

3.3. Error analysis

To have a better understanding of where our system is failing will help us improve our system in future and understand the limitation of the data. We examine 100 samples from the development data and attempt to identify *where* and *why* our best system (No. 11 from table 3) is failing. Examining the specific tweets provides us with some interesting insights. Table 4 lists examples of errors made by our best performing system. The first entry in

No.	Model	Features	Acc.	Prec.	Recall	F1
1	SVM	Mazajak SG-250 + char[1-5] + word[1-3]	96.2	72	78.2	74.7
2		Char [2-6] Gram	**97.1**	72.5	**88.8**	78.2
3	Bagged SVM	Mazajak SG-250 + char[1-5] + word[1-3]	96.4	68.9	81.3	73.4
4	CNN-BiLSTM	Mazajak SG-100 + CNN Feature Extractor	95.9	76.1	67.5	70.9
5	M-BERT		95.7	72.4	74.9	73
11	**Ensemble(1+2+3)**		96.6	**80**	78.7	**79.3**

Table 7: Comparison of different systems submitted to subtask B

Tweet	True Label	Majority Label	Reason
اقطع واخس يا الداعشي يا بو حساب نتن ..	HS	NOT	?
الله لا يوفقك يا خرا يا برازيلي يا تبن	HS	NOT	?
بيلعب يا توتهام يا الستي والاقرب توتهام وبيفوز وبيلعب النهائي قدام الليفر الاقرب واحتمال كبير يفوز بالبطولة الخامسة اياكس	NOT	HS	?

Table 8: Errors analysis of subtask B

the table contains reference to Game of Thrones. This is offensive only if context of Game of Thrones is taken into account. We cannot expect the classifier to be correct on such instances. The second entry provides us with an example of error that we can tackle in the future. We see that the hashtag غلاق_حسابات_البدون_في_البنوك # is offensive toward immigrants. But it's out of vocabulary for our systems. In the future, we can attempt to parse the hashtags into its constituent words and see if it improves the performance. The third entry in the table is quite interesting. The sentence uses an offensive word, تسرقو, which means "stealing", but the sentence itself is not offensive. It's likely our system picked up the offensive word but failed to take into account the context in which the word was used. In future, we can particularly target resolving issues of ambiguous use of words.

Table 6 shows the confusion matrix of our best system. We can see that the classifier is much more likely to misclassify offensive instance as not offensive compared to misclassifying not offensive instances as offensive. This is to be expected as the data is highly skewed with only 19% of the instances being offensive.

4. Subtask B: Hate Speech Detection

For subtask B, The preprocessing is the same as section 3. In this section, we describe the different models we experimented on for subtask B, present the results for the different models and discuss errors made by the models.

4.1. System Descriptions

Since same tweets are used for both the subtasks, for subtask B, we focused only on those systems that were promising in subtask A. These systems include SVMs, CNN-BiLSTM and M-BERT. The accuracy, precision, recall and F1 score are reported in Table 7. The precision, recall and F1 scores are macro averaged.

4.1.1. SVMs

As the features for SVMs, we use the same features as subtask A. In addition, we also experiment with bagged SVM (5 estimators). The SVMs outperformed CNN-BiLSTM and M-BERT in this subtask as well.

4.1.2. CNN-BiLSTM

We keep the same structure and settings of the CNN-BiLSTM used in subtask A.

4.1.3. M-BERT

Once again, we follow the same methodology as subtask A. The only difference is that the M-BERT is now fine-tuned on hate speech detection.

4.1.4. Ensemble Method

For subtask B, we opt for a slightly different ensemble method. The best ensemble on the development set was obtained by only considering the three SVMs (Nos. 1,2,3 from table 7). We also change the voting mechanism such that it's no longer majority voting. We consider an instance to be hate speech if any of the three SVMs vote it to be hate speech. This voting scheme was outperforming majority voting scheme on the development set. The official result on this ensemble is shown in table 9.

Official Rank	Acc.	Prec.	Recall	F1
4th	96.6%	83.8%	78.1%	80.6%

Table 9: Official results on subtask B test set

	NOT HS	HS
NOT HS	940	16
HS	18	26

Table 10: Confusion matrix on subtask B dev set

4.2. Error analysis

We attempt to perform error analysis similar to subtask A by collecting 100 samples from development set and examining them. Unfortunately, we could not identify any particular reasons for the system to fail. We speculate this to be because of the data for subtask B being *extremely* skewed (with only 5% of the data being hate speech). Table 8 contains examples of errors made by the best system (no. 6 from table 7).
Table 10 shows the confusion matrix of our best system on subtask 2. As expected, because of the extreme imbalance of the data, we can see that the classifier is prone to misclassifying hate speech instances as non hate speech instances.

5. Conclusion and Future Work

To conclude, we experimented heavily with classical machine learning and deep learning approaches to detect if a tweet is offensive or contains hate speech. We achieve state of the art results for offensive language detection by combination of SVMs, CNN-BiLSTM and Multilingual BERT. We achieve competitive results on hate speech detection with a system combination of SVMs. Our error analysis indicates certain types of errors for offensive language detection that can be addressed in future. In future, we aim to explore augmentation of hate speech data to build better systems.

6. Bibliographical References

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.

Agrawal, S. and Awekar, A. (2018). Deep learning for detecting cyberbullying across multiple social media platforms. *CoRR*, abs/1801.06482.

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.

Davidson, T., Warmsley, D., Macy, M. W., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. *CoRR*, abs/1703.04009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (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, June. Association for Computational Linguistics.

Djuric, N., Zhou, J., Morris, R., Grbovic, M., Radosavljevic, V., and Bhamidipati, N. (2015). Hate speech detection with comment embeddings. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 29–30, New York, NY, USA. Association for Computing Machinery.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Kim, Y., Jernite, Y., Sontag, D. A., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615.

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, Vancouver, BC, Canada, August. Association for Computational Linguistics.

Mubarak, H., Darwish, K., Magdy, W., Elsayed, T., and Al-Khalifa, H. (2020). Overview of osact4 arabic offensive language detection shared task. 4.