Investigating the Effect of Preprocessing Arabic Text on Offensive Language and Hate Speech Detection

FATEMAH HUSAIN, Kuwait University, Kuwait
OZLEM UZUNER, George Mason University, USA

Preprocessing of input text can play a key role in text classification by reducing dimensionality and removing unnecessary content. This study aims to investigate the impact of preprocessing on Arabic offensive language classification. We explore five preprocessing techniques: conversion of emojis to Arabic textual labels, normalization of different forms of Arabic letters, normalization of selected nouns from dialectal Arabic to Modern Standard Arabic (MSA), conversion of selected hyponyms to hypernyms, hashtag segmentation, and basic cleaning such as removing numbers, kashidas, diacritics, and HTML tags. We also experiment with raw text and a combination of all five preprocessing techniques. We apply different types of classifiers in our experiments including traditional machine learning, ensemble machine learning, Artificial Neural Networks (ANN), and Bidirectional Encoder Representations from Transformers (BERT)-based models to analyze the impact of preprocessing. Our results demonstrate significant variations in the effects of preprocessing on each classifier type and on each dataset. Classifiers that are based on BERT do not benefit from preprocessing, while traditional machine learning classifiers do. However, these results can benefit from validation on larger datasets that cover broader domains and dialects.

CCS Concepts: • **Computing methodologies** → **Information extraction**; **Natural language processing**.

Additional Key Words and Phrases: artificial neural networks, offensive language detection, natural language processing, Arabic language, machine learning

ACM Reference Format:
Fatemah Husain and Ozlem Uzuner. 2020. Investigating the Effect of Preprocessing Arabic Text on Offensive Language and Hate Speech Detection. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* 1, 1, Article 1 (January 2020), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Preprocessing is one of the critical steps in most text classification pipelines. The goal of preprocessing is to prepare textual input for accurate interpretation by a computational method. Earlier studies investigating the impact of preprocessing on text classification report different results for different preprocessing techniques and in different datasets[HaCohen-Kerner et al. 2020; Woo et al. 2020]. These findings indicate a need to better understand the impact of preprocessing on text classification. Moreover, the role of preprocessing in Arabic Natural Language Processing (NLP)

Authors' addresses: Fatemah Husain, f.husain@ku.edu.kw, Kuwait University, Kuwait City, Kuwait; Ozlem Uzuner, George Mason University, Fairfax, USA, ouzuner@gmu.edu.

Manuscript submitted to ACM

has received limited attention[Duwairi and El-Orfali 2014; Saad 2010]. The Arabic language has unique characteristics that need to be considered during preprocessing.

This study extends our previous work on offensive language detection and hate speech detection published within the scope of the Fourth workshop on Open-Source Arabic Corpora and Corpora Processing Tools (OSACT)[Husain 2020b]. Our former study demonstrated the significant impact of intensive preprocessing on the performance of a Support Vector Machine (SVM) classifier. Intensive preprocessing consisted of conversion of emojis to textual labels, normalization of multiple shapes of Arabic letters to one shape for each letter, normalization of Arabic dialects to Modern Standard Arabic (MSA) for a selected set of nouns, conversion of hyponyms to hypernyms for a selected set of nouns, formatting hashtags to include textual content only, and basic cleaning such as removal of punctuation, numbers, HTML tags, non-Arabic characters, etc. Our system ranked third in offensive language detection and ranked first in hate speech detection, with macro-averaged F1 scores of 89% and 95%, respectively.

Similar intensive preprocessing has been adopted by other studies for offensive language detection but without further investigating the contribution of different preprocessing techniques to the performance of classifiers. In [Husain 2020a], the author developed multiple ensemble traditional machine learning classifiers using all preprocessing techniques defined by Husain [Husain 2020b] without reporting the change in performance before and after preprocessing. Husain et al.[Husain et al. 2020] applied multiple preprocessing techniques with various deep learning classifiers without showing the effect on performance.

This paper provides an in-depth investigation and evaluation of preprocessing on automatic offensive language detection in Arabic. We utlize multiple classifiers and two different data sets. We assess the effect of various preprocessing techniques individually and in combination. This study includes (1) traditional machine learning classifiers: SVM and Logistic Regression (LR), (2) ensemble machine learning classifiers: Bagging and Random Forest (RF), (3) Artificial Neural Network (ANN) classifiers: Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM), and (4) Bidirectional Encoder Representations from Transformers (BERT) based models: AraBERT[Antoun et al. 2020] and Arabic-BERT[Safaya et al. 2020].

This paper addresses the following questions:

(1) What are the most significant preprocessing techniques for Arabic offensive language detection?
(2) How does preprocessing affect traditional machine learning classifiers and the more advanced classifiers?
(3) Do the results of the advanced classifiers imply a need for change in traditional text classification methods?

The following section starts with an introduction to Arabic language, followed by a related work section. Then, we discuss the methodology and the settings of the experiments, which include a description of the datasets, preprocessing techniques, features, and classification models. The results of the experiments are presented after the methodology section. We provide a detailed discussion and interpretation of our results in the discussion section. We include a conclusion section to summarize our recommendations and findings. The paper ends with a presentation of future work that highlights some possible ways to extend our work.

2 THE ARABIC LANGUAGE

Arabic is the official language for the majority of countries in the Middle East. It is also the original language of the Quran and the Hadith, thus, it is widely studied by Muslims around the world. There are three main forms of the Arabic language: (1) classical Arabic, which is the language used in Islamic manuscripts such as the Quran; (2) MSA, which is the official language of schools, news outlets, Arabic books, etc.; and (3) dialectal Arabic, which is the actual spoken

Arabic that varies from one area to another and from one society to another. According to [Habash 2010], the Arabic dialects can be divided into seven categories: Egyptian, Levantine, Gulf, North African, Iraqi, Yemeni, and Maltese.

Arabic is part of the Semitic language family. Scripts in Arabic are read and written in the opposite direction to English, i.e., from right to left. Arabic alphabet consists of 28 letters. It contains only two vowels: (أ/alif and ي/yaa'). Each letter in Arabic can be written in multiple shapes depending on the location within a word. For example, the letter "ف/faa'" can be written as "ـفـ/ ف / ـف" based on whether it is located at the beginning, in the middle, or at the end of a word[Gayar and Suen 2018; Habash 2010]. Figure 1 shows an example.

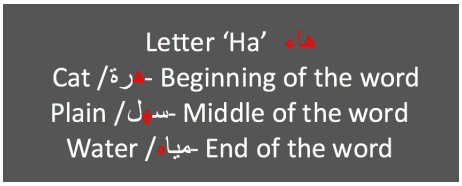


Fig. 1. Variations in the shape of Arabic letters based on the location of the letter within a word

The use of diacritics is very common in Arabic. Diacritics are often called Tashkīl or Harakāt in Arabic, and they are used as a phonetic guide. In some cases, diacritics help reduce ambiguity as multiple words in Arabic can share the same spelling but have different pronunciation and meaning. For example, the Arabic word "شعر", "شَعْرٌ" means hair, "شَعَرَ " means feel, and "شِعْرٌ" means poetry[Alhumoud et al. 2015; Boudad et al. 2018]. Figure 2 shows another example of the effect of diacritics.

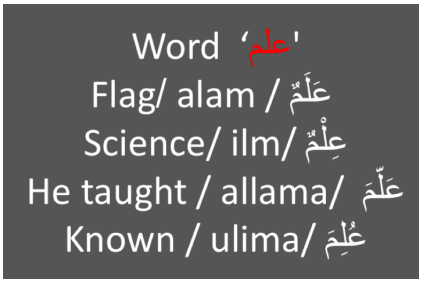


Fig. 2. The effect of diacritics on the meaning of Arabic words

Arabic words are justified by using elongation or Tatweel also called Kashida. Figure 3 shows an example of the use of elongation in Arabic.

The Arabic language has two genders for nouns: feminine and masculine. The base form is the masculine form. For example, the word "Qetta/قطة" refers to female cat and the word "Qett/قط" refers to male cat. In addition, Arabic has singular, dual, and plural forms of nouns, verbs, pronouns, and adjectives. For instance, for referring to two female cats, the word "Qettatan/قطتان" is used, for two male cats, the word "Qettan/قطان" is used, and for plural cats, the word "Qettat/قططا" is used.

The aforementioned properties describe the general properties of MSA and dialectal Arabic. However, Arabic text in user-generated content is more diverse and has other properties that are important to consider as well. User-generated Arabic content is often transliterated in Latin characters, numbers, and punctuation. For example, the number "2" can

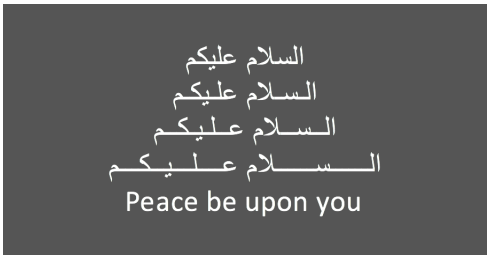


Fig. 3. Examples of elongation in Arabic

represent the letter "أ" (that sounds like "a" as in apple) and the number "3" can represent the letter "ع" (that is a guttural "aa'") [Darwish 2014]. Examples of words written in Arabizi, a form of user-generated Arabic, are "3a2albik/عالبك/on your heart" and "raw3a/روعة/wonderful" [Husain 2020b]. Moreover, code-switching among the various forms of Arabic and sometime English can be observed in the same post in user-generated content. Figure 4 demonstrates the diversity of user-generated content in Arabic language.

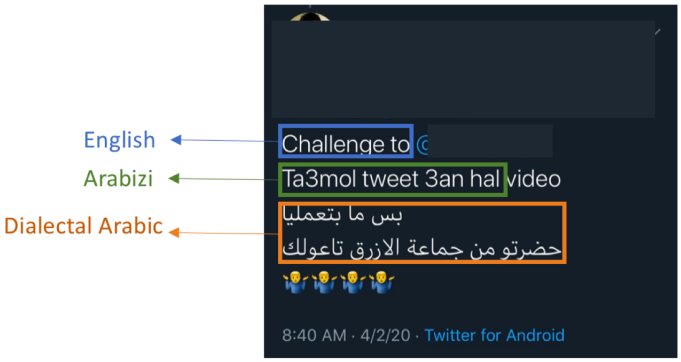


Fig. 4. An example user-generated post in Arabic

Some English terms are commonly used among Arabic speakers. Some Arabic text might show English text represented using Arabic letters. We show an example of a tweet for English text written in Arabic letters in Figure 5.

In some Arabic dialects, the pronunciation of the Arabic letters differs. Thus, users make use of letters from other languages to represent the dialectal sound of the letter rather than the standard Arabic letter. In this scenario, Urdu and Farsi alphabets are commonly used. Figure 6 shows an example of a tweet that is written in Iraqi Arabic, which replaces the letter "ك/kaa'" in Arabic with the letter "گ/Gaa" in Farsi and Urdu.

3 RELATED WORK

There are few studies that focus on analyzing the effect of preprocessing on Arabic text classification. A pioneering study in this area was delivered by Saad in 2010 [Saad 2010]. Saad [Saad 2010] compared the effect of Arabic text preprocessing on the performance of different traditional machine learning classifiers, including: Decision Trees (DT), K-Nearest Neighbors (KNN), SVMs, Naïve Bayes (NB) and its variations (Multinomial Naïve Bayes (MNB), Complement (CNB), and Discriminative Multinomial Naïve Bayes (DMNB)). Preprocessing techniques that were applied

Fig. 5. An example for a post representing English in Arabic letters

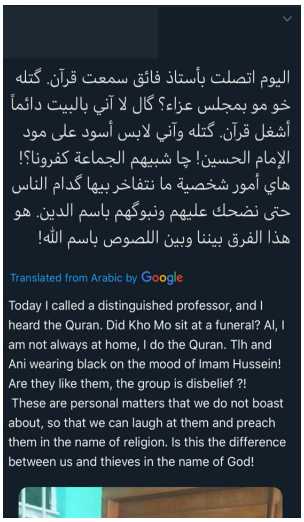


Fig. 6. An example of the use of non-Arabic letters in dialectal Arabic

include tokenizing strings to words, normalizing tokenized words, removing stop-words, different term weighting schemes, and Arabic morphological analysis. The weighting schemes consisted of the Boolean model that used 0 to refer to absence of a word or 1 to its presence, word count, normalized word count, term pruning, Term Frequency (TF), and Term Frequency-Inverse Document Frequency (TF-IDF). Arabic morphological analysis contained two stemming techniques: stemming and light stemming. Stemming substitutes words with their stems, while light stemming eliminates common affixes from words without substituting them with their stems. In [Saad 2010], seven Arabic datasets (BBC Arabic website bbcarabic.com corpus, CNN Arabic website cnnarabic.com corpus, Open Source Arabic Corpus (OSAC), Contemporary Corpus of Arabic (CCA), Aljazeera News corpus, Khaleej newspaper of the year 2004 corpus, and a corpus from multiple online Arabic newspapers) were used to evaluate the effect of preprocessing using Bag-of-Words (BOW). Results highlighted the significance of preprocessing Arabic text for feature reduction, which reduces

complexity of classifiers, consumes less storage, and saves processing time. Findings also showed the following: (1) light stemming with term pruning achieves the highest performance score; (2) SVM and variations of NB perform better than other algorithms; and (3) weighting schemes influence the performance of distance-based classifiers.

In another study [Duwairi and El-Orfali 2014], as part of building a sentiment analysis system for Arabic text, the authors analyzed the effect of removing stop-words and stemming on the performance of classifiers. They used two datasets: the first one contains reviews that are collected from political articles at Aljazeera Arabic news website and the second one contains movie reviews written in MSA. They used BOW to represent their features and applied SVM, NB, and KNN classifiers. The results showed that preprocessing improves performance over no preprocessing on the Aljazeera dataset. However, for the movie reviews, stemming hurt performance. Authors attributed this result to the use of the Rapidminer's stemmer, which has a high error rate. The authors also pointed out the fact that the movie reviews dataset is larger than the Aljazeera dataset, thus, the high error rate of the stemmer is more obvious and shows sharper effects on the movie reviews dataset.

Effect of preprocessing on text classification has been investigated more extensively in English than in Arabic. In this section, we also discuss two recent studies that focus on the effect of preprocessing on English text classification. HaCohen-Kerner, Miller, and Yigal [HaCohen-Kerner et al. 2020] studied six preprocessing techniques: spelling correction, HTML tag removal, converting uppercase letters into lowercase letters, punctuation mark removal, reduction of repeated characters, and stop-word removal. The experiments were performed on four datasets: the 4 Universities Data Set (WebKB), the multi-labeled Reuters-21578 dataset (R8), the SMS Spam Collection dataset, and the Sentiment Labelled Sentences dataset. They utilized BOW and three classifiers: a Bayes Networks (BN), a variant of SVM (SMO), and a Random Forest (RF). The results demonstrates the importance of experimenting with various preprocessing techniques in systematic combinations when developing a text classification system as different datasets show different effects of preprocessing. For example, stop-word removal shows significant improvement in accuracy score when applied to WebKB, R8, and SMS Spam Collection, but not when applied to Sentiment Labelled Sentences. Thus, the main finding is that there are no preprocessing techniques which can always improve text classification performance regardless of the dataset.

Woo, Kim, and Lee [Woo et al. 2020] also analyzed the effect of preprocessing on text classification. Their study applied two main types of preprocessing techniques: special character elimination, lemmatization, lowering upper case, and punctuation splitting or merging were categorized as typical preprocessing techniques. The second type of preprocessing techniques were more advanced and consisted of technical terminology preprocessing and sorting sentences based on their complexity. Technical terminology preprocessing used a customized rule-based algorithm with a technical term corpus. Rules were used to extract technical terminology and Wikipedia API was used to verify the accuracy of the identified terminologies. When a technical terminology was identified, the terminology was reformed to ensure proper segmentation during preprocessing to reduce any semantic change. The second advanced preprocessing technique calculated the entropy of each sentence based on the distribution of syllables. Entropy was used as the sentence complexity score. The Stanford Natural Language Inference (SNLI) corpus was used for training and testing the model. In this study, the authors considered modern neural network models only: a Convolutional Neural Network (CNN) model, Siamese LSTM model, and a transformer model. The experiments evaluated combinations of preprocessing techniques. Results showed higher accuracy scores when two preprocessing techniques are applied, specifically, lemmatization and punctuation splitting, lemmatization and lowering upper case letters, or lowering upper case letters and punctuation splitting. If only one technique is applied, the authors recommend using lemmatization, lowering upper case letters, or punctuation splitting. The results also demonstrate that preprocessing techniques which shorten

the lengths of sentences are not recommended. Special character elimination, normalization techniques, and ordering sentences based on their complexities do not improve accuracy.

This paper covers the gap in previous studies by providing an in-depth analysis of various Arabic text preprocessing techniques using several classifiers, including state-of-the-art language models, for text classification.

4 METHODOLOGY

Generally, we follow a traditional text classification pipeline in our experimental study, consisting of preprocessing, feature extraction, classification, and performance evaluation. We borrow the preprocessing procedures from Husain [Husain 2020b]. We apply each preprocessing technique separately and all preprocessing techniques together to create multiple versions of the input datasets in addition to the raw data. We apply multiple classifiers to each version of the datasets. We compare the results of classifiers that utilize input text preprocessed using different techniques and their combination against the raw data baseline in order to understand the contribution of each preprocessing technique to the end result.

4.1 Datasets

The datasets used in this study contain tweets. One dataset contains multiple dialects while the other is focused on the Levantine dialect.

The first dataset is a multi-dialect OSACT dataset [Mubarak et al. 2020], which supports two offensive language classification tasks: (a) detecting if a post is offensive or not, and (b) identifying if a post is hate speech or not. The providers of the OSACT dataset performed some preprocessing to ensure the privacy of users: Twitter user mentions were substituted by "@USER", URLs were substituted by "URL", and empty lines were replaced by "<LF>". Moreover, the OSACT dataset is highly imbalanced, 19% of the entire dataset is offensive tweets, and 5% contains hate speech.

The second offensive language dataset is a Levantine Twitter Dataset for Hate Speech and Abusive Language (L-HSAB) [1], which contains 5,846 tweets labeled as hate speech (468 tweets), abusive language (1,728 tweets), or normal (3,650 tweets) [Mulki et al. 2019]. L-HSAB is the first Levantine hate speech Arabic dataset. The dataset is manually filtered to ensure the collection of only Syrian and Lebanese tweets without duplication. Then, the tweets are cleaned to remove some characters (such as @, RT, and #) before presenting them to the annotators.

For both datasets, we shuffle them and then randomly split them into 80% training dataset and 20% testing dataset. Table 1 summarizes classification tasks related to each dataset. For all models, the training portion is used to develop the models and the testing portion is used to evaluate the models.

4.2 Preprocessing techniques

The following are the preprocessing techniques that were included in our experiments.

4.2.1 Emoji Conversion. Emojis are often used to convey feelings and attitudes, which could be very valuable to offensive language detection. Preprocessing text for emojis has been shown to improve aggression detection [Orăsan 2018]. We convert emojis to Arabic textual labels that describe the content of the emojis. For this, we use Beautifulsoup4 4.8.2[2] to extract the entire set of emojis defined by Unicode.org[3], which provides textual descriptions for emojis written in

[1] https://github.com/Hala-Mulki/L-HSAB-First-Arabic-Levantine-HateSpeech-Dataset
[2] https://pypi.org/project/beautifulsoup4/
[3] https://home.unicode.org

Table 1. Classification tasks

Dataset	Number of Classes	Class Labels	Training Dataset Size	Testing Dataset Size
OSACT	2	Offensive Not offensive	1,581 offensive 6,417 not offensive	408 offensive 1,590 not offensive
OSACT	2	Hate speech Not hate speech	408 hate speech 7,590 not hate speech	96 hate speech 1,902 not hate speech
L-HSAB	3	Abusive Hate Normal	1,399 abusive 365 hate 2,909 normal	327 abusive 101 hate 739 normal

Table 2. Examples of emojis and their Arabic labels with English translations

Emoji	Arabic Label	English Translation
🙂	وجه مبتسم قليلا	slightly smiling face
😃	وجه مبتسم بعيون كبيرة	grinning face with big eyes
🐵	وجه القرد	monkey face
🐒	قرد	monkey
🎂	كعكة عيد الميلاد	birthday cake

English language. We translate the descriptions to Arabic using Translate 1.0.7 python package[4]. The final extracted emoji list contains 1,374 emojis. Table 2 shows examples of emojis with their Arabic textual labels along with their English translations.

4.2.2 Arabic Dialect Normalization. To normalize dialectal nouns, we convert to MSA. For example, the word cat from multiple dialects — Egyptian "Otta/أطة", Levantine "Bisse/بسة", Gulf "Qatwa/قطوة", Moroccan "Qetta/قطة", Yemeni "demah/دمة", and Iraqi "Bazzuna/بزونة"— are converted to MSA form "قطة". The selected dialectal nouns are based on a manual inspection of the training set.

4.2.3 Word Categorization. This step converts hyponyms to hypernyms for a selected set of nouns. From manual inspection of a sample of tweets from the training dataset, we notice that it is very common to mention names of animals among hate speech tweets. Thus, we manually created a list of common animal names used in different Arabic dialects, such as "كلب/dog", "خنزير/pig", "حية/snake". The list considers the dialectal variations in animal names as well as the gender and quantity variations in animal names. We substitute all animal names with the Arabic word "حيوان", which means animal.

4.2.4 Letter Normalization. Some Arabic letters can be written in various forms depending on their location within a word. We normalize three letters to convert all their forms into one form — Alif (إ،آ،أ to ا), Alif Maqsura (ئ،ي to ى), and Ta Marbouta (ة to ه). In addition, letters repeated more than two times within a word are reduced to two times only.

4.2.5 Miscellaneous & Hashtag Segmentation. In addition to the previous techniques, we apply some miscellaneous preprocessing techniques that are commonly performed in the field. We remove numbers, Kashida, diacritics, HTML tags, more than one space, three or more repetitions of any character, and some symbols or terms (e.g., "", "...", "!", "?",

[4]https://pypi.org/project/translate/1.0.7/

Table 3. Examples of tweets using each preprocessing technique

Preprocessing technique	Example Before Preprocessing	Example After Preprocessing	Translation
Emoji Conversion conversion	فى حاجات مينفعش نلفت نظركوا ليها زى الاصول كده يا اتربيتوا عليها يا لأ😇	فى حاجات مينفعش نلفت نظركوا ليها زى الاصول كده يا اتربيتوا عليها يا لأ وجه مبتسم مع هالة	We do not want to bother you with things that you are not aware about like race
Dialect normalization	لا تعتب عليه هيدا اسمه ابو صرماي	لا تعتب عليه هيدا اسمه ابو حذاء	Don't worry his name is shoe's owner
Word categorization	ما كان ممكن تبدأ بيه بدل ما تخسر تبديل يا حمار	ما كان ممكن تبدأ بيه بدل ما تخسر تبديل يا حيوان	You need to start with him better than losing, donkey
Letter normalization	يا أنت يا حبيبي يا أنيق	يا انت يا حبيبي يا انيق	Lovely, you are elegant
Hashtag segmentation	نحبك يا تاج الرأس يا كل أيامنا أنت#يوم_الأم	نحبك يا تاج الرأس يا كل أيامنا أنت يوم الأم	We love you, you are our days mother's day
Miscellaneous	RT @USER: @USER @USER يا نموت احرار يا نمشي ماليزيا	نموت احرار نمشي ماليزيا	We either die free or we will leave to Malaysia

”:”, ”;”, ”:”, ”#”, ”@”, ”$”, ”&”). Moreover, we use the list of Arabic stop-words defined by Alrefaie Github[5], which contains 750 stop-words to filter the text of stop-words.

Finally, hashtags are commonly used in user-generated content to support sharing content and highlight important phrases within the posts. Hashtags in Arabic usually connect multiple words using '_', similar to "#hate_speech". We preprocess hashtags by removing the "#" symbol and replacing "_" by a space. For example, the hashtag "#الهلال_التعاون/#AlHillal_Altaawen" (name of a sport team) is converted to "الهلال التعاون/AlHillal Altaawen".

Table 3 shows an example of a tweet from the OSACT dataset before and after preprocessing by each technique.

4.3 Features

Given datasets consisting of tweets, with inconsistent writing formats, we consider using character-based features. We develop BOW features using n-grams of 2 to 5 characters implemented using the Sklearn Python library. To better assess the deep learning classifiers, we use AraVec word embeddings in addition to BOW features. AraVec is a pre-trained word embedding that is developed specifically for the Arabic language [Soliman et al. 2017]. Unlike BOW, it does not assume each word to be independent of others. For BERT-based models, we use the output from the BERT encoder as the feature. Below, we provide more details.

4.4 Classification Models

We evaluated the impact of preprocessing on multiple classifiers.

4.4.1 Traditional Machine Learning. LR and the SVM are used as representative traditional machine learning classifiers. They are developed using Python scikit-learn library. BOW with 2 to 5 characters is used in both LR and SVM models.

[5]https://github.com/mohataher/arabic-stop-words

4.4.2 Ensemble Machine Learning. Bagging and Random Forest represent ensemble machine learning classifiers. They were implemented using Python scikit-learn library. Similar to the traditional machine learning classifiers, BOW with 2 to 5 characters is used in both.

4.4.3 Artificial Neural Network (ANN). The ANN classifiers — RNN and LSTM — are also included in the study. We use Keras to develop ANN models using 3000 for maximum features, 0.25 for drop-out, 100 units, 1000 for patch size, and 500 epochs. Only the ANN classifiers are implemented using two features: BOW with 2 to 5 characters and AraVec word embeddings. Each feature is used separately from the other.

4.4.4 BERT-Based Models. We include BERT-based language models in our analysis. BERT is the current state-of-the-art in text classification. We use AraBERT [Antoun et al. 2020] (first version of AraBERT, also called AraBERTv1-base and bert-base-arabert) and Arabic-BERT [Safaya et al. 2020] to represent BERT-based classifiers. We apply each using maximum length = 128, patch size = 16, epochs = 2, epsilon = 1e-8, and learning rate = 2e-5. We do not use feature engineering; instead, we use the pooling layer from the encoder and feed it into a Feed Forward Neural Network (FFNN). We use the HuggingFace library[6] to apply both BERT models and develop our experiments using the PyTorch-Transformers library.

5 RESULTS

We consider macro-averaged measurements for all metrics: precision, recall, accuracy, and F1. The choice of macro-averaged metrics is motivated by the class imbalance in our datasets. Overall, results do not demonstrate any significant patterns for specific Arabic text prepossessing techniques across all tasks. The variations in performance achieved by different preprocessing techniques for each classifier are small.

Figure 7 and Figure 8 show results for offensive language detection using the OSACT dataset. As can be seen from Figures (a) to (d) for the traditional machine learning and ensemble classifiers, the SVM and the Random Forest perform better when applied to fully processed text, while the LR and Bagging perform better when applied to dialect normalized text. The ANN classifiers show different results when applied using BOW features versus AraVec features, as can be noticed from Figures (e) and (f). In general, preprocessing text reduces the performance of BOW-based ANN classifiers. For Figures (g) and (h), miscellaneous preprocessing and hashtag segmentation improve the performance of AraVec-based ANN classifiers. Both BERT-based models in Figures (i) and (j) show better performance when applied to dialect normalized text, while AraBERT also reports better performance when applied to emoji converted text.

Results for hate speech detection using the OSACT dataset are presented in Figure 9 and Figure 10. The SVM (Figure (a)) shows very high performance when applied to fully processed text; however, the LR reports the lowest performance when applied to fully processed text. As can be seen from Figure (b), the LR performs better with miscellaneous pre-processed text and hashtag segmentation. Figure (c) for the Random Forest demonstrates equal effects on performance by raw text and fully processed text, which also report the highest performance. The Bagging classifier (Figure (d)) and all ANN classifiers from Figures (e) to (h) improve in performance when applied with miscellaneous preprocessed text and with hashtag segmentation. BERT-based models (Figures (i) and (j)) do not show improvement on preprocessed text over raw text.

Figure 11 and Figure 12 highlight the results of abusive and hate speech detection using the L-HSAB dataset. Figure (a) for the SVM demonstrates that there is no improvement from preprocessing text, while Figure (b) for the LR reports

[6] https://huggingface.co/

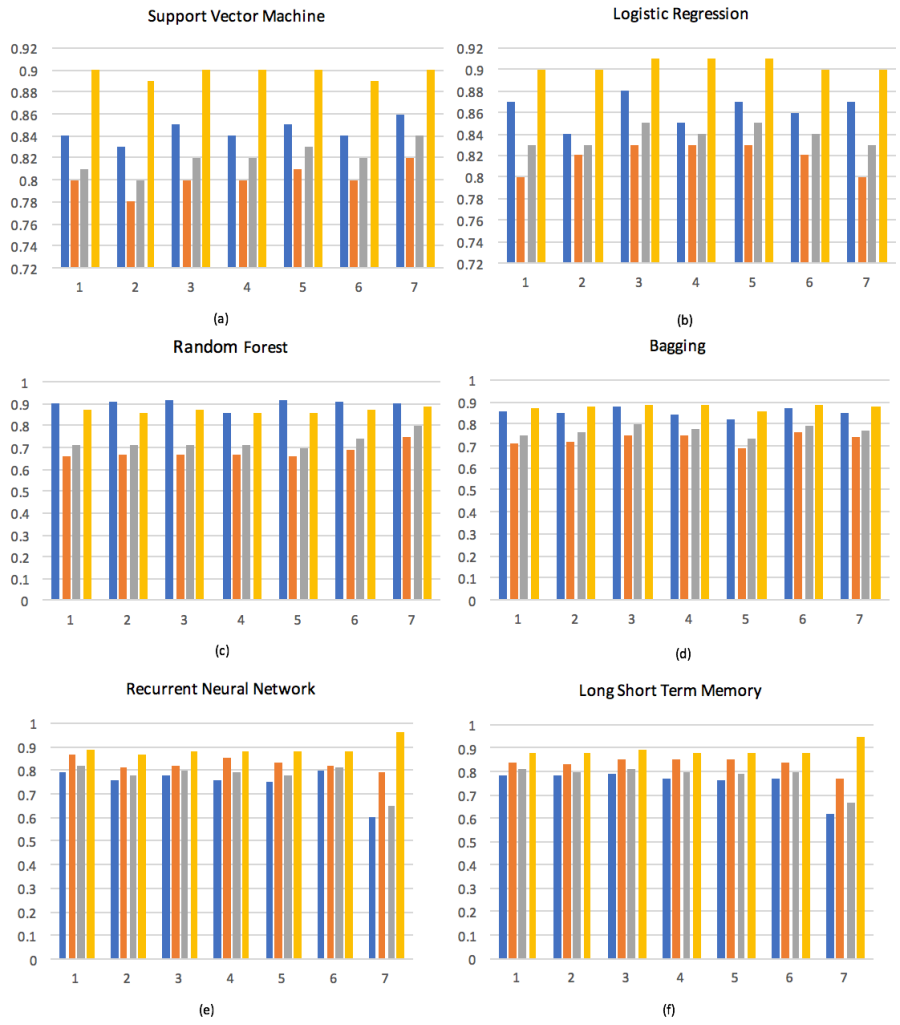
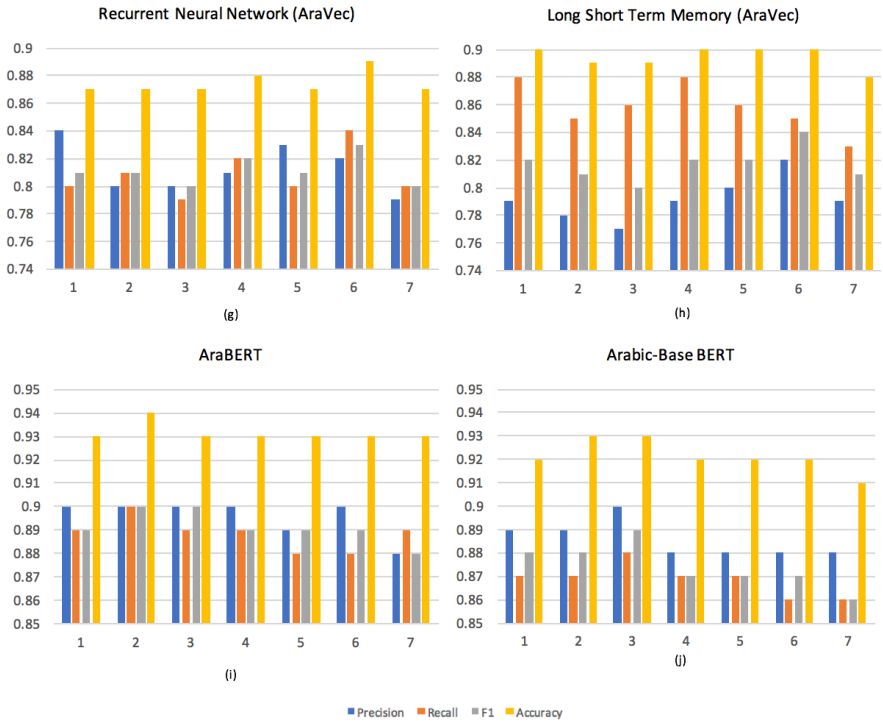


Fig. 7. Results of offensive language detection using the OSACT dataset

slightly better performance on dialect normalized text and letter normalized text over raw text. The Random Forest (Figure (c)) achieves better performance on dialect normalized text and fully processed text. On the other hand, the Bagging model in Figure (d) does not gain from preprocessing text. The RNN model with BOW (Figure (e)) does not gain from preprocessing as raw text records the highest performance. Figure (f) for the LSTM model with BOW shows the best performance on raw text, dialect normalized text, and miscellaneous preprocessed and hashtag segmented text. AraVec-based models show different results, in Figure (g) for the RNN, best performance is on miscellaneous preprocessed and hashtag segmented text, while in Figure (h) for the LSTM, the best performance is on raw text. Figure (i) for AraBERT shows the same highest performance score on four preprocessing techniques: emoji conversion, dialect normalization, word categorization, and miscellaneous preprocessing and hashtag segmentation. Arabic-BERT model

demonstrates slightly better performance on text preprocessed for emoji conversion and dialect normalization over raw text.



Step Number	Description
1	Baseline Model
2	Emoji and Emoticon Conversion
3	Dialects Normalization
4	Word Categorization
5	Letters Normalization
6	Miscellaneous and Hashtags Segmentation
7	All Preprocessing Steps

Fig. 8. Results for offensive language detection using the OSACT dataset (continue)

6 DISCUSSION AND FINDINGS

We perform a statistical analysis for the macro-averaged F-1 results to summarize results from the three offensive language detection tasks: offensive language detection on the OSACT dataset, hate speech detection on the OSACT dataset, and abusive and hate speech detection on the L-HSAB dataset. The statistical analysis includes minimum, maximum, mean, standard deviation, and range of macro-averaged F1 scores among the preprocessing techniques presented in the previous section, in addition to the raw text without preprocessing (baseline) and fully processed text using all preprocessing techniques on the same time. Tables 4, 5, and 6 show the statistical analysis for the results from the three tasks.

Generally, the results demonstrate the limited effect of Arabic text preprocessing on offensive language detection. From the statistical analysis, the traditional machine learning classifiers gain in performance more than the ensemble

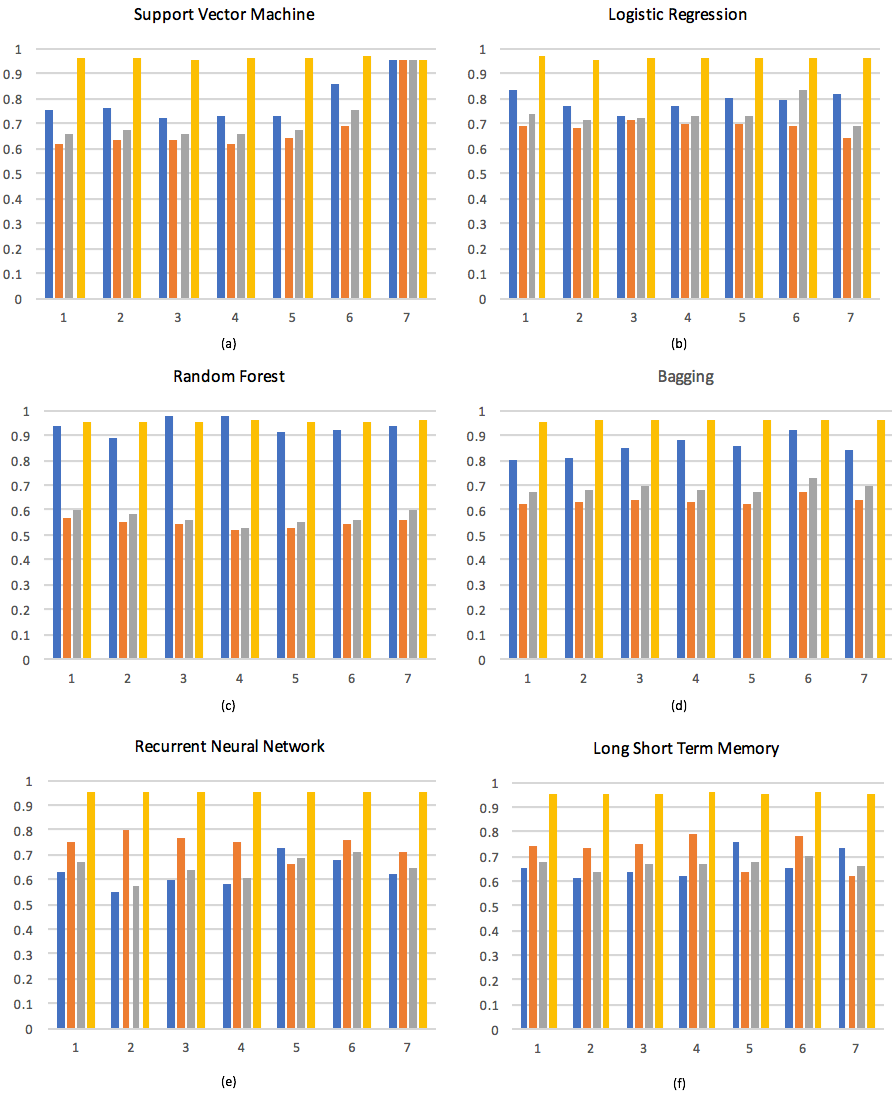
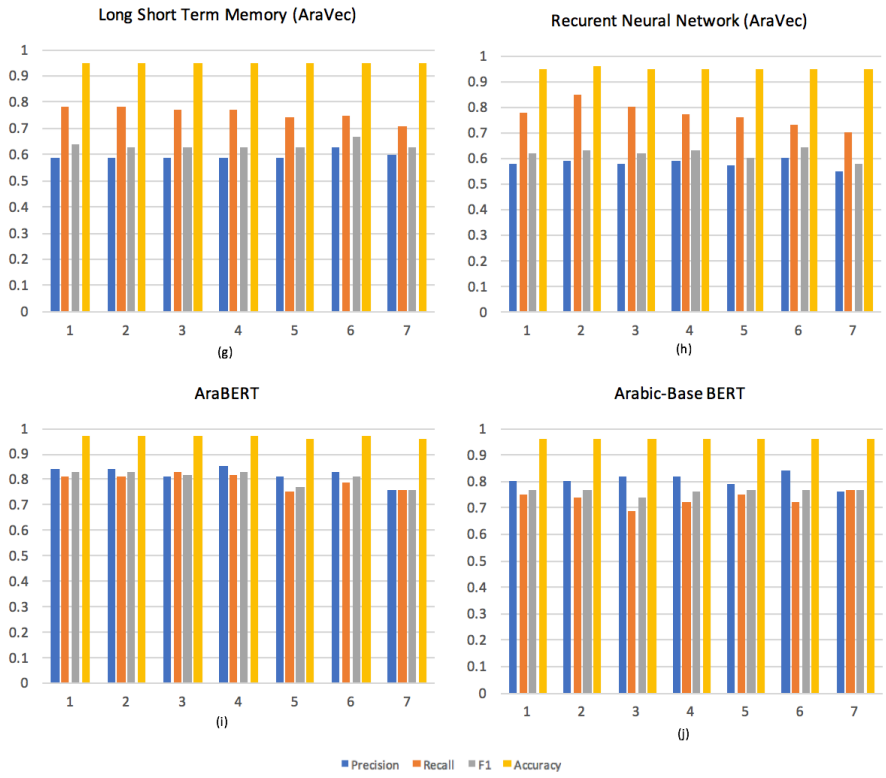


Fig. 9. Results for hate speech detection using the OSACT dataset

machine learning classifiers and deep learning classifiers. Only for hate speech detection on the OSACT dataset, the traditional machine learning, particularly the SVM, performs much better than all other classifiers including BERT-based classifiers.

The advanced BERT-based models gain the least from preprocessing. This finding implies that for systems that depend on advanced models, we can use the raw text without preprocessing. We expect that this is because BERT-based models learn the context rather than the individual words and their exact form, and are therefore able to effectively conflate the variations of words that appear in the same context. Preprocessing those words would be redundant in terms of conflating variants as the contextual representation which is already immune to the variations.



Step Number	Description
1	Baseline Model
2	Emoji and Emoticon Conversion
3	Dialects Normalization
4	Word Categorization
5	Letters Normalization
6	Miscellaneous and Hashtags Segmentation
7	All Preprocessing Steps

Fig. 10. Results for hate speech detection using the OSACT dataset (continue)

Examining the values of standard deviations from the statistical analysis tables show that the majority of experiments show very small values of 0.01, which indicates that macro-averaged F1 scores of the preprocessing techniques and the other two versions of the data — raw and fully processed using all techniques together — are clustered very closely around the mean. Among all tables, the largest standard deviation is 0.11, which is the result from the SVM classifier for hate speech detection. This finding reveals the high sensitivity of the SVM classifier to the various forms of the input text.

The statistical analysis tables also show that the lowest macro-averaged F1 scores for all tasks come from ensemble machine learning classifiers: Random Forest and Bagging. However, the classifier with the best performance varies among the tasks. For instance, AraBERT performs best on both offensive language detection on the OSACT dataset and on abusive and hate speech detection on the L-HSAB dataset, while SVM performs best on hate speech detection on the OSACT dataset.

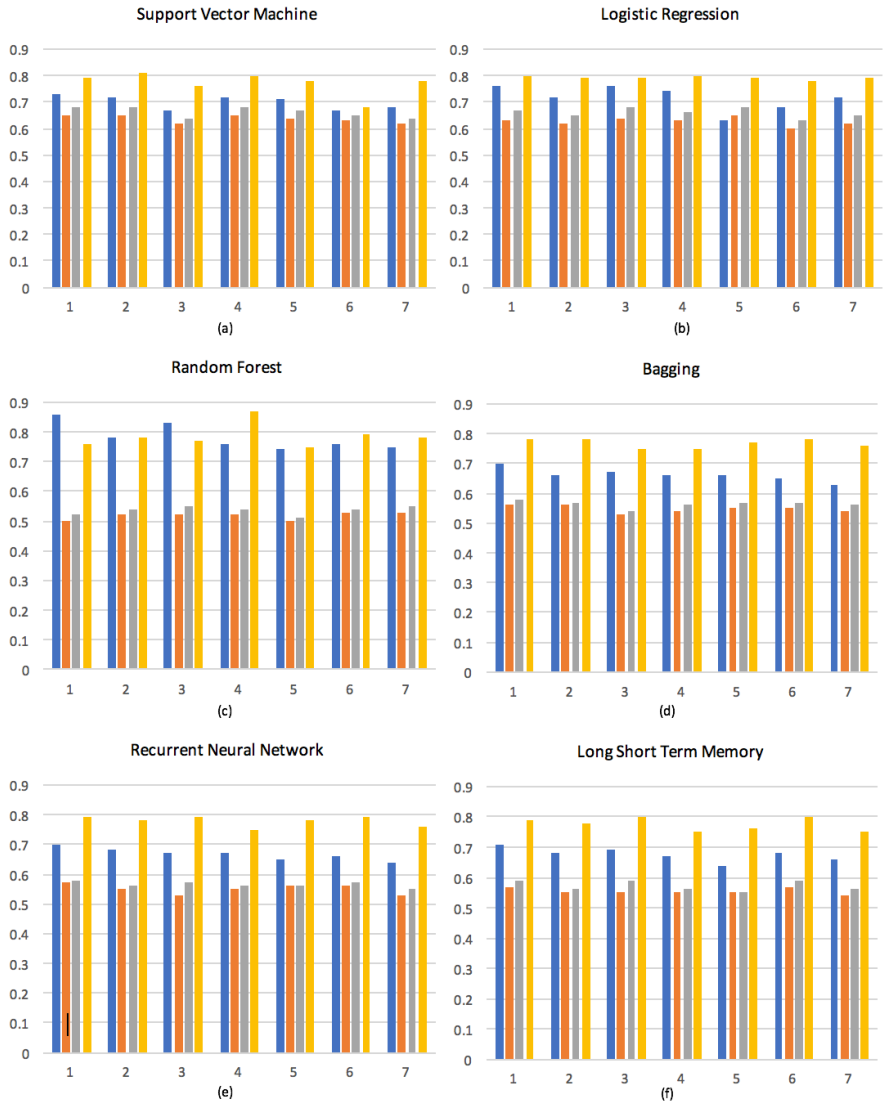
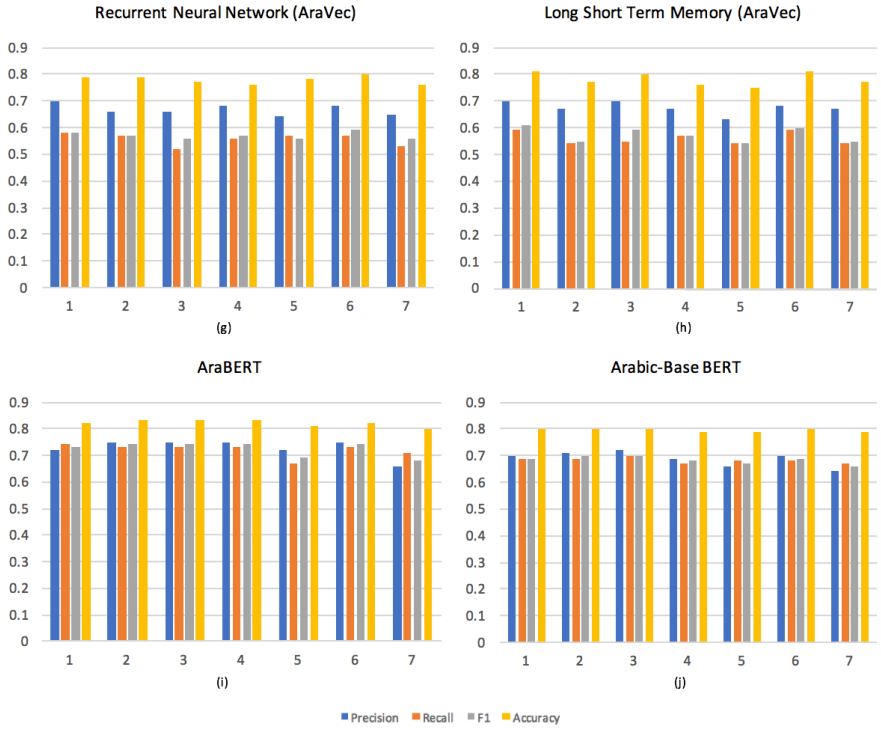


Fig. 11. Results for abusive and hate speech detection using the L-HSAB dataset

For offensive language detection on the OSACT dataset, the highest macro-averaged F1 score is 0.90, achieved by the AraBERT classifier. Table 4 shows that the highest variations among all forms of input text occur when applying the RNN and the LSTM with BOW features for offensive language detection. Table 5 presents the statistical analysis for hate speech detection on the OSACT dataset. It highlights the effectiveness of the SVM classifier. Moreover, the variation is relatively high among the input text forms for the SVM classifier in comparison to the rest of the classifiers. In Table 6 for abusive and hate speech detection on the L-HSAB dataset, the variations become more even, most of the classifiers have 0.01 standard deviation and BERT-based classifiers significantly exceed the others in performance.



Step Number	Description
1	Baseline Model
2	Emoji and Emoticon Conversion
3	Dialects Normalization
4	Word Categorization
5	Letters Normalization
6	Miscellaneous and Hashtags Segmentation
7	All Preprocessing Steps

Fig. 12. Results for abusive and hate speech detection using the L-HSAB dataset (continue)

This finding could be related to the diversity of the Arabic language forms used in the OSACT dataset versus the L-HSAB dataset, as the OSACT dataset covers broader dialects. Consequently, the preprocessing techniques we developed significantly affect performance. The L-HSAB dataset has only Levantine text; thus, preprocessing has less impact on the performance.

To further examine the generalizability and robustness of the classifiers on preprocessed text, we aggregate macro-averaged F1 scores from all tasks into Table 7. AraBERT model shows the highest minimum macro-averaged F1, the highest maximum macro-averaged F1, and the highest mean macro-averaged F1 scores. These results indicate the power of AraBERT in detecting offensive content regardless of preprocessing.

Table 8 summarizes the results from all experiments, including all tasks depending on macro-averaged F1 score per preprocessing technique to analyze the results for each technique separately. Results show the highest minimum and maximum macro-averaged F1 scores when using all preprocessing techniques at the same time. Text that has been preprocessed using miscellaneous and hashtag segmentation shows the highest average macro-averaged F1. All techniques report very similar standard deviation scores, which means that the variation in macro-averaged F1 among the

Table 4. Statistical analysis of macro-averaged F1 results for offensive language detection using the OSACT dataset

Classifier	Minimum	Maximum	Mean	Standard Deviation	Range
SVM	0.80	0.84	0.82	0.01	0.04
LR	0.83	0.85	0.84	0.00	0.02
Random Forest	0.70	0.80	0.72	0.03	0.10
Bagging	0.73	0.80	0.77	0.02	0.07
RNN (BOW)	0.65	0.82	0.77	0.06	0.17
LSTM (BOW)	0.67	0.81	0.78	0.05	0.14
RNN (AraVec)	0.80	0.83	0.81	0.01	0.03
LSTM (AraVec)	0.80	0.84	0.82	0.01	0.04
AraBERT	0.88	0.90	0.89	0.00	0.02
Arabic-BERT	0.86	0.89	0.87	0.00	0.03

Table 5. Statistical analysis of macro-averaged F1 results for hate speech detection using the OSACT dataset

Classifier	Minimum	Maximum	Mean	Standard Deviation	Range
SVM	0.66	0.95	0.72	0.11	0.29
LR	0.69	0.83	0.73	0.04	0.14
Random Forest	0.53	0.60	0.57	0.03	0.07
Bagging	0.67	0.73	0.69	0.02	0.06
RNN (BOW)	0.57	0.71	0.65	0.05	0.14
LSTM (BOW)	0.64	0.70	0.67	0.02	0.06
RNN (AraVec)	0.58	0.64	0.62	0.02	0.06
LSTM (AraVec)	0.63	0.67	0.64	0.01	0.04
AraBERT	0.76	0.83	0.80	0.03	0.07
Arabic-BERT	0.74	0.77	0.76	0.01	0.03

tasks is very small regardless of the preprocessing technique. The lowest range is 0.35, which is achieved by several preprocessing techniques, including emoji conversion, dialect normalization, and miscellaneous and hashtag segmentation. Overall, the findings from this table demonstrate a better performance (highest maximum macro-averaged F1) among all tasks when applying all preprocessing techniques at the same time.

7 CONCLUSION

In this study, we examined the impact of several preprocessing techniques on Arabic offensive language detection. We focus on five preprocessing techniques: conversion of emojis to Arabic textual labels, normalization of different forms of Arabic letters, normalization of selected nouns from Arabic dialects to MSA, conversion of selected hyponyms to hypernyms, hashtag segmentation, and basic cleaning processes, such as removing numbers, Kashida, diacritics, and

Table 6. Statistical analysis of macro-averaged F1 results for abusive and hate speech detection using the L-HSAB dataset

Classifier	Minimum	Maximum	Mean	Standard Deviation	Range
SVM	0.64	0.68	0.66	0.02	0.04
LR	0.63	0.68	0.66	0.02	0.05
Random Forest	0.51	0.55	0.53	0.01	0.04
Bagging	0.54	0.58	0.56	0.01	0.04
RNN (BOW)	0.55	0.58	0.56	0.01	0.03
LSTM (BOW)	0.55	0.59	0.57	0.02	0.04
RNN (AraVec)	0.56	0.59	0.57	0.01	0.03
LSTM (AraVec)	0.54	0.61	0.57	0.03	0.07
AraBERT	0.68	0.74	0.72	0.03	0.06
Arabic-BERT	0.66	0.70	0.68	0.01	0.04

Table 7. Statistical analysis of macro-averaged F1 results from all tasks

Classifier	Minimum	Maximum	Mean	Standard Deviation	Range
SVM	0.70	0.82	0.73	0.05	0.12
LR	0.72	0.79	0.74	0.02	0.07
Random Forest	0.58	0.65	0.65	0.02	0.07
Bagging	0.65	0.70	0.70	0.02	0.06
RNN (BOW)	0.59	0.70	0.66	0.04	0.11
LSTM (BOW)	0.62	0.70	0.67	0.02	0.08
RNN (AraVec)	0.65	0.69	0.67	0.01	0.04
LSTM (AraVec)	0.66	0.71	0.68	0.02	0.05
AraBERT	0.77	0.82	0.80	0.02	0.05
Arabic-BERT	0.75	0.79	0.77	0.01	0.03

HTML tags. The experiments utilize several classifiers, including traditional machine learning classifiers, ensemble machine learning classifiers, deep learning classifiers with BOW and AraVec, and BERT-based classifiers. The results show unequal effects of preprocessing techniques based on the classifier and demonstrate limited gains from preprocessing on offensive language detection. Specifically for the advanced BERT-based classifiers, preprocessing shows limited gains and can be omitted from text classification pipelines.

The findings from this study indicate that certain possible combinations of preprocessing techniques might improve the performance for some classifiers, such as deep learning systems with BOW features or SVMs for hate speech. This finding demonstrates the importance of further investigating the impact of preprocessing Arabic text for offensive language detection by conducting more comprehensive studies that include multiple dialects and social media platforms.

Table 8. Statistical analysis of macro-averaged F1 results from each preprocessing technique among all tasks

Preprocessing technique	Minimum	Maximum	Mean	Standard Deviation	Range
No preprocessing	0.52	0.89	0.71	0.10	0.41
Emoji Conversion conversion	0.54	0.90	0.70	0.11	0.35
Dialect normalization	0.54	0.90	0.70	0.11	0.35
Word categorization	0.53	0.89	0.70	0.11	0.36
Letter normalization	0.51	0.89	0.70	0.11	0.38
Miscellaneous and hashtag segmentation	0.54	0.89	0.72	0.10	0.35
All preprocessing	0.55	0.95	0.70	0.11	0.40

Our experiments utilize relatively small datasets with small number of offensive samples; thus, studying the effects with larger and more balanced datasets would support the generalizability of our findings.

8 FUTURE WORK

While preprocessing and its effects are important to understand, future work would benefit from inclusion of semantic information in addressing Arabic natural language processing tasks.

REFERENCES

Sarah Alhumoud, Mawaheb Altuwaijri, Tarfa Albuhairi, and Wejdan Alohaideb. 2015. Survey on Arabic Sentiment Analysis in Twitter, In World Academy of Science, Engineering and Technology. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 9, 1, 364–378.

Wissam Antoun, Fady Baly, and Hazem Hajj. 2020. AraBERT: Transformer-based Model for Arabic Language Understanding. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*. European Language Resource Association, Marseille, France, 9–15. https://www.aclweb.org/anthology/2020.osact-1.2

Naaima Boudad, Rdouan Faizi, Rachid Oulad Haj Thami, and Raddouane Chiheb. 2018. Sentiment analysis in Arabic: A review of the literature. *Ain Shams Engineering Journal* 9, 4 (2018), 2479 – 2490. https://doi.org/10.1016/j.asej.2017.04.007

Kareem Darwish. 2014. Arabizi Detection and Conversion to Arabic. In *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)*. Association for Computational Linguistics, Doha, Qatar, 217–224. https://doi.org/10.3115/v1/W14-3629

Rehab Duwairi and Mahmoud El-Orfali. 2014. A study of the effects of preprocessing strategies on sentiment analysis for Arabic text. *Journal of Information Science* 40, 4 (2014), 501–513. https://doi.org/10.1177/0165551514534143 arXiv:https://doi.org/10.1177/0165551514534143

Neamat El Gayar and Ching Suen. 2018. . Series on Language Processing, Pattern Recognition, and Intelligent Systems, Vol. 4. World Scientific. 1– 288 pages. https://doi.org/10.1142/10693

Nizar Y. Habash. 2010. (1 ed.). Synthesis Lectures on Human Language Technologies, Vol. 3. Morgan Claypool Publishers. 1–187 pages. https://doi.org/10.2200/S00277ED1V01Y201008HLT010

Yaakov HaCohen-Kerner, D. Miller, and Yair Yigal. 2020. The influence of preprocessing on text classification using a bag-of-words representation. *PLoS ONE* 15 (2020).

Fatemah Husain. 2020a. Arabic Offensive Language Detection Using Machine Learning and Ensemble Machine Learning Approaches. arXiv:2005.08946 [cs.CL]

Fatemah Husain. 2020b. OSACT4 Shared Task on Offensive Language Detection: Intensive Preprocessing-Based Approach. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*. European Language Resource Association, Marseille, France, 53–60. https://www.aclweb.org/anthology/2020.osact-1.8

Fatemah Husain, Jooyeon Lee, Samuel Henry, and Ozlem Uzuner. 2020. SalamNET at SemEval-2020 Task12: Deep Learning Approach for Arabic Offensive Language Detection. In *Proceedings of the 14th International Workshop on Semantic Evaluation*. Barcelona, Spain, 2133–2139.

Hamdy Mubarak, Kareem Darwish, Walid Magdy, Tamer Elsayed, and Hend Al-Khalifa. 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), with a Shared Task on Offensive Language Detection*, Vol. 4. European Language Resource Association, Marseille, France. https://www.aclweb.org/anthology/2020.osact-1.2

Hala Mulki, Hatem Haddad, Chedi Bechikh Ali, and Halima Alshabani. 2019. L-HSAB: A Levantine Twitter Dataset for Hate Speech and Abusive Language. In *Proceedings of the Third Workshop on Abusive Language Online*. Association for Computational Linguistics, Florence, Italy, 111–118. https://doi.org/10.18653/v1/W19-3512

Constantin Orăsan. 2018. Aggressive Language Identification Using Word Embeddings and Sentiment Features. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*. Association for Computational Linguistics, Santa Fe, New Mexico, USA, 113–119. https://www.aclweb.org/anthology/W18-4414

Motaz Saad. 2010. *The Impact of Text Preprocessing and Term Weighting on Arabic Text Classification*. Ph.D. Dissertation. https://doi.org/10.13140/2.1.4677.2164

Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. KUISAIL at SemEval-2020 Task 12: BERT-CNN for Offensive Speech Identification in Social Media. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. International Committee for Computational Linguistics, Barcelona (online), 2054–2059. https://www.aclweb.org/anthology/2020.semeval-1.271

Abu Bakr Soliman, Kareem Eissa, and Samhaa R. El-Beltagy. 2017. AraVec: A set of Arabic Word Embedding Models for use in Arabic NLP. *Procedia Computer Science* 117 (2017), 256 – 265. https://doi.org/10.1016/j.procs.2017.10.117 Arabic Computational Linguistics.

HoSung Woo, JaMee Kim, and WonGyu Lee. 2020. Validation of Text Data Preprocessing Using a Neural Network Model. *Mathematical Problems in Engineering* 2020 (2020), 1958149. https://doi.org/10.1155/2020/1958149