Influence of Offensive Words on the Machine Learning Detection Performance of Offensive Tweets

Arjan Dekker
student number: s3726169
a.j.dekker.5@student.rug.nl
Marios Souroulla
student number: s4765125
m.souroulla@student.rug.nl
Filippos Andreadis
student number: s4942906
f.andreadis@student.rug.nl

Abstract

As personal opinions on social media become more and more influential towards the general public, the pervasiveness of negative and offensive comments grows as well. As a result, the interest in the research field of automatically identifying offensive language has seen a noticeable increase. In this work we use the Offensive Language Identification Dataset (OLID), a dataset consisting of annotated tweets classified as either offensive or not, in order to train a variety of machine and deep learning models and observe how their detection performance is influenced by the existence of offensive terminology in the text. Our findings suggest that there is an important drop in performance when offensive words are entirely removed from the dataset, even for state-of-the art pretrained models such as BERT.

1 Introduction

Nowadays, many people use social media to connect with other people. It is a place where people can post stories, share opinions and react on other people's posts.

While there are many positive aspects of social media, it has also become a place where users can offend other users or people. As a social media platform, you may want to detect such behavior and remove these posts, comments or even users from your platform.

(Zampieri et al., 2019a) have collected tweets and classified them as offensive or non-offensive. They used Support Vector Machine (SVM), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models to classify the tweets into the two categories. CNN had promising results: 0.80 macro f1-score. The dataset created was also made available for a public competition between researchers (Zampieri et al., 2019b). Here researchers were able to push the macro f1-score for this task to 82.9% using the BERT language model (Devlin et al., 2018).

Since approximately 40% of the offensive tweets contain offensive words (see Section 3), we are interested in the influence of offensive words on the classification performance of machine learning (ML) models. In order to determine this influence, we perform three experiments. One where we keep the offensive words (keep), one where we replace the offensive words with a marker (replace), and one where we remove the offensive words entirely (remove). Our research question is therefore:

RQ What is the influence of offensive words on the offensive tweet detection for ML models?

This research questions is answered using the following three sub questions:

1. How accurate can ML models detect offensive tweets when keeping offensive words?
2. How accurate can ML models detect offensive tweets when replacing offensive words with a marker?
3. How accurate can ML models detect offensive tweets when removing offensive words entirely?

In order to see how different types of models react to the replacement or removal of offensive words, we perform the experiments with a variety of models. First, to get baseline scores we use traditional ML models including K-Nearest-Neighbors (KNN), Naive Bayes (NB), and Linear

Support Vector Machine (LSVM). Next, we experiment with a more sophisticated LSTM model, which should provide better performance since it takes into account sequences of words (i.e. context). Lastly, we experiment with recent language models (LMs) including BERT and RoBERTa (Liu et al., 2019b), as these should provide an even better understanding of language. The code for these models can be found in our GitHub repository[1].

We obtained the best performances using the language model and specifically BERT. On the test set we achieved f-scores of 0.815 when keeping offensive words, 0.790 when replacing offensive words with a marker word and 0.687 when removing offensive words entirely. Hence, we see that ML models do take into account the offensive words, but they also do not rely on them entirely to make correct predictions.

Similar work to our research can be found in the next Section. More details about our methodology can be found in Section 3. In Section 4 we show the results we obtained, also for the other models we tested including traditional ML methods and LSTM models. Finally, in Section 5 we discuss our results and conclude this research by answering our RQ.

2 Related Work

Hate speech recognition has been studied broadly in the last years, especially with the rise of social media platforms.

However, this task is proven to be non-trivial, as it has a number of challenges (Kovács et al., 2021). In a recent work (Pawar et al., 2022), the authors explain three basic approaches to that problem. The first approach is by using keywords, an intuitive method, but with a big disadvantage, it heavy relies on particular words and ignores the context. The other approach is to take advantage of the user data, like demographics, location, and some other information. However, this rises some ethical concerns about the user's privacy. The last approach, which we also follow in our task, is the Machine Learning classifiers and NLP models. This method bypasses the problems imposed by the other two approaches, and it achieves good results in various NLP problems.

In 2019, Zampieri et al. started a competition between researchers for a shared task. The shared task is the classification of annotated tweets they provide as 'offensive' or 'not offensive'. They used SVM, LSTM, and CNN models, out of these three, CNN was able to score the best results, achieving an f-score of 0.80. However, other researchers participating in the competition were able to achieve even better results (f-score 0.829) by using the BERT language model. In our work, we aim to evaluate different ML models, LSTM models, and fine-tuned language models, on the shared task.

Moreover, we aim to analyze the impact of offensive words in this classification task. To do this, we compare the performance of the models when keeping the offensive words, when replacing the offensive words by a marker, and removing them completely.

For the non-deterministic models (LSTM and the LMs), we trained and evaluated the models three times. Since we cannot fix the seeds, the seeds can be considered random. In the end we averaged the results and calculated the standard deviations. These results can be found in 4.

3 Method

This section includes a detailed description of the OLID and the pre-processing applied to the tweets. Additionally, we provide the methodology followed for each one of the machine learning, deep learning and pre-trained models we have experimented with. For the evaluation of our models, we use the macro f1-score due to class imbalance, precision, recall and accuracy.

3.1 Dataset

The dataset we used was created by (Zampieri et al., 2019a) and is essentially a collection of tweets in English annotated as offensive, or not offensive.

There are a total of 14,100 tweets, split into 3 sets: a training set, a development set, and a test set. The training set is used to train the algorithm, the development set is used to evaluate the trained model and select the best-performing parameters, and test set it used to evaluate the performance of the model on unseen data. The samples of each class for the three sets are shown on Table 1.

It is important to also check how many of the tweets actually contain offensive words. To do this, we use the better-profanity library [2] to find and replace the offensive words. Table 2 shows the

[1]https://github.com/GWNLekkah/LfD-final-assignment

[2]https://pypi.org/project/better-profanity/

	Training	Development	Testing
Offensive	4048	352	240
Not offensive	8192	648	620
Total	12240	1000	860

Table 1: Number of samples per set.

percentage of tweets that contain offensive words per class, for each set. We notice that the test set contains much more offensive tweets that have offensive words compared to the training and development set.

Set	Non-offensive tweets	Offensive tweets
Training	8%	39%
Development	9%	41%
Test	11%	48%

Table 2: Percentage of tweets that contain offensive words.

Some examples of the raw data are shown in Table 3.

Label	Sample
OFF	IM FREEEEE!!!! WORST EXPERIENCE OF MY FUCKING LIFE
OFF	@USER Fuk this fat cock sucker
OFF	@USER Figures! What is wrong with these idiots? Thank God for @USER
NOT	@USER He is so generous with his offers.

Table 3: Example samples in the dataset

The data is already preprocessed for anonymization by the creators, they have replaced the usernames with a '@USER' tag, and also replaced all the URLs with a 'URL' tag.

3.2 Preprocessing

During the pre-processing, there are some steps that we apply to the whole dataset, regardless of the algorithm used. These preprocessing steps are based on the steps from team NULI (Liu et al., 2019a), since they performed best on the shared task. The first step, is to remove the '@USER' tags, because we believe that it just introduces noise and unnecessary complexity to our models. Then we replace the 'URL' tag with the string 'http' because this was found to work better for the LM models and it does not make any difference for the other models. After that, we segment the hashtags using the 'wordsegment' library[3], and we lowercased the tweets. Lastly, we replace emojis with their description using the 'demoji' library[4]. For example, the fire emoji would be replaced by the word 'fire'.

3.3 Machine Learning Models

The first models we experimented with, are some classic ML models: NB, KNN and LSVM. For the implementation of these, we used the scikit-learn library [5].

These algorithms use features that are extracted from the text data. Before we try to optimize the models on each task, we first evaluate their performance without any feature preprocessing and parameter optimization, in order to obtain some baselines scores. The first step towards optimization is the optimization of the feature extraction process to improve our results. The following features were sequentially tested on the development set. This means that we first select the vectorizer (between BOW and TFIDF), and then add stemming, if that helps then we keep it and try adding POS-tags. Lastly, we experiment with word-level n-grams, in order to get a look at the context. The best setup for each model and task was used for the hyper-parameter tuning phase.

- *CountVectorizer*: Bag of words representation of the corpus.
- *TfIdfVectorizer*: Short for term frequency–inverse document frequency, assigns a value to each word representing its importance in the corpus based on how often it re-occurs in different documents of the corpus.
- *Stemming*: Represent each word with its stem.
- *POS-tags*: Assign a part of speech tag as an extra attribute for each word.
- *N-gram sizes*: We experimented with word-level 1,2 and 3-gram sizes.

[3]https://pypi.org/project/wordsegment/
[4]https://pypi.org/project/demoji/
[5]https://scikit-learn.org/stable/

Lastly, after the best feature setup has been found for each model, Random search (with maximum 800 parameter sets) is applied using the train set, and evaluated on the development set. This is done in order to derive the best hyper-parameters, and is done separately for each task. The last step of the pipeline is to perform the final evaluation on the held-out test set, using the feature and hyper-parameter sets found previously. The remaining part of the section provides a detailed overview of the settings tested for each classifier, and a brief description of each algorithm.

3.3.1 Naive Bayes

The first classifier we evaluated was the NB, we choose NB because it was shown that it performs well, and is computationally cheap, relative to other algorithms (Colas and Brazdil, 2006). The algorithm is based on Baye's theorem about conditional probabilities.

The way this classifier performs the prediction is described by the following equation. $c_{pred} \;=_{c \in C}\; \log(p(i_1|c) + \log(p(i_2|c).. + \log(p(i_n|c) + \log(c)$

Where c_{pred} is the predicted class, C is the set of all possible classes, and $p(i_n|c)$ is the conditional probability of feature n, given the class c.

Its main disadvantage is that it assumes all features are independent, which is a strong assumption, especially for text classification. However, we mitigate this by experimenting with n-grams during the feature selection.

The baseline f-scores of NB are 0.701, 0.638, and 0.696, for keeping, removing, and replacing offensive words respectively.

Regarding the feature selection the best setup is shown in Table 4, this has increased the performance by 0.01 for keeping, 0.04 for removing, and did not help for replacing.

	Keep	Remove	Mark
TFIDF/BOW	BOW	BOW	BOW
POS	NO	YES	NO
stemming	YES	NO	NO
N-grams	1	1	1

Table 4: NB best features for each task.

For the implementation, we use the sklearn implementation of multinomial NB, and we have optimized the model for each task separately, the best-performing parameters are shown in Table 5. After optimizing the parameters, we have achieved 0.719 f-score for keeping, 0.677 for removing, and 0.701 for replacing. The following parameters were optimized:

- *alpha*{*float in range 0.1-3*}: It is the smoothing parameter added to each feature in order to avoid zero (or extremely small) probabilities.
- *fit_prior*{*True, False*}: Whether or not to use the prior classes probabilities.

	Keep	Remove	Mark
alpha	2	2	3
fit_prior	True	False	False

Table 5: NB best parameters for each task.

3.3.2 K-Nearest Neighbors

KNN is a distance-based estimator, suitable for both classification and regression problems. In the case of classification, each observation is assigned the label of the majority of the k closest remaining observations. It is considered as a 'lazy' learner as no actual computation takes place during the fitting of the data, but rather all the distances between instances are calculated during the prediction phase.

The baseline f-scores of NB are 0.518, 0.540, and 0.614, for keeping, removing, and replacing offensive words respectively.

Regarding the feature selection the best setup is shown in Table 6. This setup has increased the performance by 0.03 for keeping, 0.03 for removing, and 0.01 for replacing.

	Keep	Remove	Mark
TFIDF/BOW	BOW	BOW	BOW
POS	NO	NO	NO
stemming	YES	YES	YES
N-grams	1-3	1-2	1-2

Table 6: KNN best features for each task.

The different hyper-parameters explored are listed below:

- *n_neighbors* { *int in range 1-30* } : The number of closest observation that will be considered for the majority vote of the class label assigned to the instance at question.

- *weights* {*uniform, distance*}: Whether all observations are weighted equally or closer ones have higher influence.

- *algorithm* {*auto, kd_tree,ball_tree, brute*}: Algorithm used to compute the nearest neighbors.

- *p* { *int in range 1-2* } : This parameter control whether the manhattan or the euclidean distance is used.

- *leaf_size* { *int in range 10-50* } : Leaf size passed to BallTree or KDTree.

- *metric* {*minkowski,mahalanobis,cosine*}: Metric to use for distance computation.

For the implementation, we use the sklearn implementation of KNN classifier, and we have optimized the model for each task separately, the best-performing parameters are shown in Table 7. After optimizing the parameters, KNN has achieved 0.569 f-score for keeping, 0.630 for removing, and 0.651 for replacing.

	Keep	Remove	Mark
weights	distance	uniform	uniform
p	1	1	2
n_neighbors	1	8	6
metric	cosine	cosine	minkowski
leaf_size	10	20	30
algorithm	brute	brute	auto

Table 7: KNN best parameters for each task.

3.3.3 Linear Support Vector Machine

Since SVM classifier (Cortes and Vapnik, 1995) was proven to be suitable for text classification (Colas and Brazdil, 2006), we also experimented with this type of classifier.

This classifier tries to maximize the distance between two classes, so that it can make a decision boundary between the points. It can do this by mapping the points to higher dimensions if necessary.

Due to the reason that SVM was found to be computationally expensive, we decided to use a linear SVM instead (LSVM), which is an SVM, but with a linear kernel. This also allows us to extract the most important features. By obtaining the most important features, we can evaluate the completeness of the list with the offensive words.

The baseline f-scores of LSVM are 0.695, 0.609, and 0.691, for keeping, removing, and replacing offensive words respectively.

Regarding the feature selection, the best setup is shown in Table 8. This setup has increased the performance by 0.02 for keeping, 0.05 for removing, and 0.02 for replacing.

	Keep	Remove	Mark
TFIDF/BOW	BOW	BOW	BOW
POS	NO	YES	YES
stemming	YES	NO	NO
N-grams	1	1-3	1

Table 8: LSVM best features for each task.

For LSVM we experimented with the following hyperparameters:

- *loss* {*hinge, squared_hinge*}: LinearSVC allows experimenting with different loss functions.

- *dual* {*False, True*}: Select the algorithm to either solve the dual or primal optimization problem.

- *tol* {*floats in range 0.000001-1.0*}: Tolerance for the stopping criterion.

- *C* {*floats in range 0.5-1.5*}: Regularization parameter.

- *fit_intercept* {*False, True*}: Whether to calculate the intercept of the model or not.

- *intercept_scaling* {*floats in range 0.5-1.5*}: Higher intercept scaling reduces the effect of regularization.

- *class_weight* {*balanced, None*}: Add balanced weights to classed or not.

- *max_iter* {*integers in range 500-5000*}: Maximum number of iterations to run

The best parameters for each task are shown on Table 9. After optimizing the parameters, LSVM has achieved 0.723 f-score for keeping, 0.670 for removing, and 0.711 for replacing.

3.4 Long-Short Term Memory Models

LSTM models improve upon the Recursive Neural Network (RNN) model by adding the ability to remember information over a long sequence of data.

	Keep	Remove	Mark
loss	hinge	squared_hinge	hinge
dual	True	False	True
tol	1.0	0.1	1e-5
C	0.5	1.0	0.75
fit_intercept	True	True	True
intercept_scaling	1.25	1.25	1.0
class_weight	balanced	None	None
max_iter	2000	5000	500

Table 9: LSVM best parameters for each task.

Whereas RNN cells only have one output, LSTM cells have two outputs. The first output is similar to that of the normal RNN cell. The second output is used for remembering information for a long time. This output is determined by two gates, the forget gate and the input gate. The forget date determines the amount of information that is kept from the previous state. The input gate determines how much information is added to that from the new state. The second output of the LSTM cell is then obtained by summing the results of the forget gate and input gate. Thus, the ability of this type of architecture to maintain both long-term and short-term memory makes it well-suited to deal with the vanishing gradient problem that occurs frequently when training a RNN or a typical feed-forward network.

We experimented with a variety of hyperparameters and empirically concluded to an optimized setup for each one of the offensive word filtering cases individually. Specifically, we tested different numbers of epochs, batch sizes, values of patience and learning rate along with multiple optimizers. Additionally, we tried using different number of units for the LSTM layer ranging from 16 to 256, as well as adding an extra dense layer and experimenting with a bidirectional LSTM layer. Since the dataset is heavily imbalanced, it is important to mention that we tried both upsampling the minority class and employing class weights during training in order to prevent the network from overfitting to the majority class. The latter showed more promising results, hence we entirelly disregarded upsampling for our final tuning. Lastly, we use the pre-trained Glove word embeddings (Pennington et al., 2014) and specifically the Wikipedia 2014 version with 200-dimentional vectors per word[6].

The best setup for the task of keeping the offensive terms consists of 40 epochs, batch size of 16, patience of 5, non-bidirectional LSTM layer, no extra dense layer, 16 units and dropout of 0.1 on the LSTM layer and the Stochastic Gradient Decent (SGD) optimizer with a learning rate of 0.01. For the tasks of removing or marking the offensive words, the best performing setup was found to be exactly the same, except for having 128 LSTM units and training the model for 30 epochs instead. The resulting macro f1-scores for the development and test sets can be seen on Tables 12 and 13 respectively.

[6]https://nlp.stanford.edu/data/glove.6B.zip

	Keep	Remove	Mark
units	16	128	128
learning rate	0.01	0.01	0.01
batch size	16	16	16
epochs	40	30	30

Table 10: LSTM best parameters for each task.

3.5 Language Models

We experimented with the BERT language model since it shows good performance on many NLP tasks. Furthermore, in the competition between researchers it was found that BERT obtained the best performance of all models (Zampieri et al., 2019b). It can achieve those high scores by exploiting the idea of transfer learning. It is trained on huge amount of text data scraped from the web. Then after that it is fine-tuned on a specific task, in our case offensive language detection. For the detection it can then make use of the knowledge it obtained in the initial learning phase to get an understanding of human language (Devlin et al., 2018).

Recently, many models are introduced that are in some way based on BERT. One of those models is RoBERTa (Liu et al., 2019b). RoBERTA was claimed by the researchers that it is an improved version of BERT. They found that BERT is undertrained and the preprocessing could be enhanced. By tackling these problems they obtained the RoBERTa model.

Since we want to obtain the best possible performance with these models, we optimized both BERT and RoBERTa for all three tasks (keeping, replacing and removing offensive words). First we optimized the maximum length of the input texts (values between 32 and 128). Then we optimized the learning rate (values between 5e-5 and 5e-7) and we experimented with using a polynomial decay learning rate. Next we optimized the batch size (values between 4 and 32) used during the training phase and finally we experimented with different numbers of epochs (values between 1 and 3). The optimized parameters of BERT and RoBERTa for each task can be found in Table 11.

	Keep	Remove	Mark
BERT			
max len	64	256	256
learning rate	5e-5 5e-7	5e-5 5e-7	5e-5 5e-7
batch size	16	8	8
epochs	1	1	1
RoBERTa			
max len	256	64	128
learning rate	5e-5	5e-5 5e-7	5e-6
batch size	8	8	8
epochs	1	3	1

Table 11: LM best parameters for each task.

4 Results

Tables 12 and 13 contain the development and test set performances for each one of the optimized models and the baseline models. In the following section we provide our interpretation of these results and our final thoughts on how much the appearance of offensive terms affects the ability of models to detect offensive tweets.

We also obtained the most important features of the classification. These most important features are used to determine in which class a sample belongs. Table 14 contains the most important features for the LSVM model for each case.

5 Discussion and Conclusion

For the ML models, we can notice that the best model for each task was different (algorithm, feature selection, parameters), this justifies our decision to optimize them separately. For example, when keeping the offensive words, the best NB model, was different from the best NB model when removing the offensive words (both in feature selection and parameters). Between the three ML algorithms, KNN underperformed as it was not able to match the scores of the LSVM and NB in any of the tasks. However, the interesting thing about KNN, is that it was the only model that was able to achieve the worst performance when keeping the offensive words. One possible reason behind this, is that it heavily relies on a few words and is also achieving a low f-score when keeping the offensive words. KNN seems that is not suitable for this task.

The best ML model was found to be different depending on the task. When keeping the offensive words, the best (optimized) model was the LSVM with f-score of 0.723 on the test set. When marking the offensive words, the best model was again LSVM (with different parameters and feature setup), reaching an f-score of 0.727 on the test set. This is interesting, because it is slightly higher when compared to keeping the offensive words. This is only the case for the test set, as the score on the development set is higher when keeping the offensive words for the corresponding model. When removing the offensive words, the best ML model is NB, but with considerably lower performance, just 0.677 f-score on the test set. In general, the best-performing ML model was the LSVM, outperforming the other two in the majority of the tasks.

It is clear from the results that the LSTM model suffers the most from removing or marking offensive words. We can see a quite noticeable performance drop for the replace and keep tasks as the f1-score is even worse than the arguably less powerful ML models. A possible explanation for this is that LSTM layers greatly depend on the exact sequence of words in the text, as well as the substance of specific words, especially when we consider the use of pre-trained embeddings. For the task of maintaining offensive words their performance falls somewhere between traditional models and LM models and we could argue that an f-score of 74% is a quiet adequate and expected result for a networks of this complexity on the given problem.

When looking at the results of the language models (Table 12 and 13), we see that BERT consistently outperforms RoBERTa in terms of f-score. This is surprising since RoBERTa is often considered as the superior version of BERT. We also see that the performance on the test set is considerably higher than on the dev set. This could be explained by the fact that the test set contains more offensive tweets with offensive words compared to the dev set. This conclusion is likely true, since the performance on the test set is about the same as on the dev set when we remove the offensive words entirely. We also obtained results that are close to what is obtained in the shared task. We obtained an f-score on the test set that is only 1.4% lower (81.5% for us, 82.9% for the shared task). Since the point of our research is not to obtain the best

Offensive Words	**Model**	**F**	**P**	**R**	**A**
Keep	NB baseline	0.701	0.708	0.697	0.734
Keep	LSVM	0.723	0.737	0.716	0.758
Keep	LSTM	0.712 ± 0.016	0.717 ± 0.010	0.706 ± 0.014	0.733 ± 0.009
Keep	BERT	**0.790** ± **0.004**	0.792 ± 0.003	0.788 ± 0.006	0.810 ± 0.003
Keep	RoBERTa	0.774 ± 0.019	0.787 ± 0.008	0.767 ± 0.024	0.801 ± 0.012
Mark	NB baseline	0.696	0.701	0.693	0.728
Mark	LSVM	0.711	0.722	0.705	0.746
Mark	LSTM	0.63 ± 0.016	0.675 ± 0.011	0.623 ± 0.013	0.697 ± 0.003
Mark	BERT	**0.773** ± **0.001**	0.770 ± 0.003	0.778 ± 0.004	0.790 ± 0.003
Mark	RoBERTa	0.769 ± 0.006	0.768 ± 0.005	0.772 ± 0.007	0.788 ± 0.004
Remove	NB baseline	0.638	0.649	0.634	0.685
Remove	NB	0.677	0.674	0.682	0.698
Remove	LSTM	0.623 ± 0.014	0.619 ± 0.01	0.624 ± 0.017	0.692 ± 0.007
Remove	BERT	**0.699** ± **0.007**	0.749 ± 0.010	0.688 ± 0.006	0.754 ± 0.006
Remove	RoBERTa	0.689 ± 0.025	0.750 ± 0.020	0.679 ± 0.023	0.750 ± 0.016

Table 12: Model performance on the dev set. The first column contains the task: keeping offensive words, replacing offensive words with a marker, or removing offensive words entirely. The second column contains the model used to obtain the results. Refer to the tables in Section 3 for the optimized parameters for each model. Note that the baseline model is not optimized, i.e. it uses the default sklearn settings with a BOW embedding. All other models use the optimized settings. The other columns are macro f1-score (F), precision (P), recall (R) and accuracy (A). The two numbers are the mean and the standard deviation respectively. The results without the standard deviation are from deterministic models.

Offensive Words	**Architecture**	**F**	**P**	**R**	**A**
Keep	NB baseline	0.744	0.759	0.733	0.803
Keep	LSVM	0.723	0.736	0.713	0.787
Keep	LSTM	0.744 ± 0.019	0.782 ± 0.015	0.733 ± 0.016	0.81 ± 0.005
Keep	BERT	**0.815** ± **0.003**	0.826 ± 0.002	0.806 ± 0.004	0.855 ± 0.002
Keep	RoBERTa	0.785 ± 0.023	0.818 ± 0.011	0.768 ± 0.031	0.840 ± 0.008
Mark	NB baseline	0.722	0.736	0.712	0.787
Mark	LSVM	0.727	0.732	0.723	0.785
Mark	LSTM	0.622 ± 0.03	0.661 ± 0.021	0.619 ± 0.019	$0.703 \pm 0.01s3$
Mark	BERT	**0.790** ± **0.011**	0.789 ± 0.016	0.793 ± 0.004	0.830 ± 0.013
Mark	RoBERTa	0.781 ± 0.005	0.788 ± 0.006	0.776 ± 0.004	0.828 ± 0.005
Remove	NB baseline	0.647	0.674	0.636	0.744
Remove	NB	0.677	0.676	0.678	0.738
Remove	LSTM	0.633 ± 0.023	0.621 ± 0.014	0.645 ± 0.015	0.756 ± 0.013
Remove	BERT	**0.687** ± **0.003**	0.801 ± 0.002	0.664 ± 0.003	0.798 ± 0.001
Remove	RoBERTa	**0.687** ± **0.025**	0.780 ± 0.033	0.667 ± 0.022	0.793 ± 0.013

Table 13: Model performance on the test set. The first column contains the task: keeping offensive words, replacing offensive words with a marker, or removing offensive words entirely. The second column contains the model used to obtain the results. Refer to the tables in Section 3 for the optimized parameters for each model. Note that the baseline model is not optimized, i.e. it uses the default sklearn settings with a BOW embedding. All other models use the optimized settings. The other columns are macro f1-score (F), precision (P), recall (R) and accuracy (A). The two numbers are the mean and the standard deviation respectively. The results without the standard deviation are from deterministic models.

Keep	Replace	Remove
organis	church_nn	pressure_nn
symbol	voting_vbg	about_in a_dt
nike	ya_nn	blue_nn
doctor	baltimore_nn	msm_nn
shoe	bless_nn	so_rb good_jj
enter	feinstein_nn	wish_jj
fyi	permission_nn	girl_nn you_prp
space	save_vbp	answer_nn
staff	civilians_nns	film_nn
review	pressure_nn	you_prp know_vbp

Table 14: Most important features for the LSVM model

detection performance, we did not spend effort to optimize the models further for this. In fact, we are only interested in relative differences between the models. Still, it is good to have a model that has good performance. For the rest of the discussion we will focus on the best performing models for each task, which are the BERT models.

Replacing offensive words with a marker yields a 2% lower f-score compared to keeping the offensive words. We have two possible explanations for this. The first being that the better-profanity library does not detect all the offensive words and therefore harms the performance. The second (more likely) explanation is that the dataset contains inconsistent labels regarding certain offensive words (see Section 5.2). The result would be that the model correctly finds the offensive words and thus marks the tweets as offensive, but the label can sometimes be not offensive.

If we remove the offensive words entirely, we see that the f-score drops another 7% for the dev set and 10% for the test set. This is of course expected to happen. However, it still achieves a decent f-score of 69%. We can therefore conclude that the models do take offensive words into account a lot. However, it does not rely on them entirely either. The models can probably use other words or context to determine whether a tweet is offensive. Also, since not all offensive tweets contain offensive words, the model is forced during training to also consider other words and context.

5.1 Most Important Features

One of the advantages, and main reason we used LSVM, is that it allows us to explore the most important features, according to the classifier. When looking at Table 14, we can notice that the keywords differ for each task. We should keep in mind that these keywords are indicators for either one class, or the other.

The interesting thing we get from this analysis, is the absence of any offensive words from the list, we would expect some offensive words to show up for the 'keep' case. Especially the marker ('OFFOFFOFF') we used to replace the offensive words. However, the absence of offensive words for the other two cases (replace, remove) indicates the completeness of the list we use as offensive words.

Apart from that, the absence of offensive words from this list, is also showing that the model does not rely on any particular offensive word, neither on the set of offensive words.

There are some interesting words among that list, for example, fyi (For Your Information) probably indicates that the tweet is probably offensive. Moreover, 'voting' and 'feinstein' (American politician) indicate that the tweet is probably offensive, without the word themselves being offensive. Furthermore, there are some keywords and bigrams that indicate that the tweet is probably not offensive. For example, the bigram 'so good', and 'girl you' are usually shown on positive tweets.

5.2 Error Analysis

We are also interested in what mistakes our models make. Therefore, we used the best model for each of the tasks to make predictions for the test set and we then manually inspected which mistakes are made and selected a number of interesting examples. These examples can be found in Table 15.

This analysis is interesting in many ways. First of all we see that some offensive words are not considered offensive by the keep model (first example). However, when we replace this word with a marker, we see that it is considered offensive. Even more interesting is how the remove model considers this offensive without having the offensive word at all. It could be that the way this sentence is written makes it more likely to be offensive. On the contrary, the second example contains a word that could be considered offensive which is labeled as not offensive. Hence, there is much reason for the keep and replace models to predict this tweet to be offensive.

The third example is interesting because it is not clear whether this label is correct. In this example someone is called racist and exclusionary which could definitely be considered as offensive. Hence it is not that surprising that two of the models, the keep and replace model, consider this as offensive.

The last example is similar to the first two examples, as it contains a word that could be considered offensive. Therefore this tweet is probably labeled as offensive as well. Since the replace model detects the offensive marker it classified this tweet as offensive as well. However, the keep model does not find this word offensive apparently, which is interesting.

There are many examples similar to the ones explained in this section. In many cases we found that the tweets are likely to be mislabeled. The other large part of these examples can be seen as corner cases where it is not clear whether these should be labeled as offensive or non-offensive.

5.3 Future work

Regarding future work, there are some things we plan to investigate further. Firstly, we plan to perform more error analysis to understand the cases that our models fail.

Moreover, we can try to improve the given dataset by re-annotating some questionable labels. Another idea, is to expand the dataset, by combining it with other datasets about hate speech recognition.

Furthermore, we plan to extract the most important features using the BERT language model, this can be done by looking at the attention scores the model gives to each word. These features might be more relevant than the ones obtained from the LSVM, since the performance of BERT is much better.

Another interesting thing to explore, is the performance of these models, when replacing the offensive words with words that have the same meaning but are not offensive themselves. For example, we can replace the word 'idiot', with the words 'not smart'.

Lastly, we aim to perform some analysis to find out more on why (some) models are still able to perform when removing the offensive words, even with less performance.

References

Fabrice Colas and Pavel Brazdil. 2006. Comparison of svm and some older classification algorithms in text classification tasks. In Max Bramer, editor, *Artificial Intelligence in Theory and Practice*, pages 169–178, Boston, MA. Springer US.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

György Kovács, Pedro Alonso, and Rajkumar Saini. 2021. Challenges of hate speech detection in social media. *SN Computer Science*, 2(2):1–15.

Ping Liu, Wen Li, and Liang Zou. 2019a. NULI at SemEval-2019 task 6: Transfer learning for offensive language detection using bidirectional transformers. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 87–91, Minneapolis, Minnesota, USA, June. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach.

AB Pawar, Pranav Gawali, Mangesh Gite, MA Jawale, and P William. 2022. Challenges for hate speech recognition system: Approach based on solution. In *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pages 699–704. IEEE.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019a. Predicting the type and target of offensive posts in social media. In *Proceedings of the 2019 Conference of the North American Chapter of*

Tweet	**Offensive words**	**Keep**	**Replace**	**Remove**	**Ground Truth**
@USER @USER Oh noes! Tough shit.	shit	NOT	OFF	OFF	OFF
@USER No fucking way he said this!	fucking	OFF	OFF	NOT	NOT
@USER @USER Please explain what controlled opposition"" means. As for Bernier escaping, he split with the rest of the Conservatives because he's even more racist and exclusionary than they want to be.	None	OFF	OFF	NOT	NOT
@USER @USER @USER @USER LOL!!! Throwing the BULLSHIT Flag on such nonsense!! #PutUpOrShutUp #Kavanaugh #MAGA #CallTheVoteAlready URL	BULLSHIT	NOT	OFF	NOT	OFF

Table 15: Wrong prediction by the models. The first column contains the tweets and the second column contains the offensive words in each tweet. The column Keep contains the predictions for the model where offensive words are kept. Similarly the column Replace contains the predictions of the model where offensive words are replaced and the Remove column contains the predictions where the offensive words are removed. The last column contains the real label.

the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 1415–1420, Minneapolis, Minnesota, June. Association for Computational Linguistics.

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

A How to run the code

The code we used can be found in our GitHub repository[7]. For running the best models for each of the tasks, use the following commands, where `program.py` should be `create_features.py` for creating the features, `train_model.py` for creating and training the model and `evaluate_model.py` for evaluating the model on the dev and test sets. The first command is for the keep task, the second for the replacing with a marker task and the last command is for removing offensive words entirely.

-
  ```
  python program.py --vectorizer=LM --model_type=LM
  --features_file=bert-keep --model_file=bert-keep --max_len=64
  --batch_size=16 --epochs=1
  ```

-
  ```
  python program.py --vectorizer=LM --model_type=LM
  --features_file=bert-replace --model_file=bert-replace
  --offensive_replacement=mark --max_len=256 --batch_size=8
  ```

-
  ```
  python program.py --vectorizer=LM --model_type=LM
  --features_file=bert-remove --model_file=bert-remove
  --offensive_replacement=remove --max_len=256 --batch_size=8
  ```

Table 16 contains all the arguments you can specify.

B Work Distribution

Marios Souroulla focused on the ML models, Filippos Andreadis focused on the LSTM model, and Arjan Dekker focused on the pre-trained LM models. We worked equally for the production of this report.

[7]https://github.com/GWNLekkah/LfD-final-assignment

Argument	Type	Description
–train_file	string	Filename of the train set
–dev_file	string	Filename of the dev set
–test_file	string	Filename of the test set
–feature_file	string	Filename to save the features
–model_file	string	Filename of save the model
–offensive_word_count	flag	Enable to obtain offensive word counts. Only works when using –offensive_replacement=mark
–algorithm	string	Specify the algorithm to use (NB, KNN, LSVM)
–pos_tag	flag	Enables part-of-speech tagging
–stem	flag	Enable word stemming
–offensive_replacement	string	Select what to do with offensive words (None, mark, remove)
–ngram_min	int	Set the lower bound of the ngram range
–ngram_max	int	Set the upper bound of the ngram range
–char_ngram	flag	Enable character level ngrams
–random_search	flag	Enable random search of hyperparameters (for ML models)
–grid_search	flag	Enable grid search of hyperparameters (for ML models)
–most_important_features	flag	Show the most important features (for LSVM)
–vectorizer	string	Select which vectorizer to use (BOW, TFIDF, LSTM, LM)
–max_len	int	Maximum length of the text samples
–batch_size	int	Specify the batch size used during training
–epochs	int	Specify the number of epochs used during training
–patience	int	Specify the patience for early stopping
–learning_rate	float	Specify the learning rate for the optimizer
–polynomial_decay	bool	Enable/disable polynomial decay learning rate
–pd_start	float	Starting value of the polynomial decay
–pd_end	float	Ending value of the polynomial decay
–pd_power	int	Specify the polynomial power for the polynomial decay
–pd_steps	int	Number of steps used for the polynomial decay
–embeddings	string	Embedding file used
–lstm_units	int	Set the number of units for the LSTM layer(s)
–dense	int	Set the number of nodes for the dense layer. If 0, no dense layer is used (relevant only to the LSTM model)
–trainable_embedding	flag	Enable or disable a trainable embedding layer for the LSTM model
–pretrained_embedding	flag	Enable or disable a pretrained embedding for the LSTM model
–lstm_layers	int	Set the number of LSTM layers
–lstm_dropout	float	Set the amount of dropout for the LSTM layers
–dropout	float	Set the amount of normal dropout between the LSTM layers
–bidirectional_lstm	flag	Set to true for bidirectional LSTM layers
–optimizer	string	Set the optimizer for the LSTM model
–class_weights	flag	Enable or disable class weights for training a LSTM model
–upsampling	flag	Enable upsampling of minority class
–model_type	string	Select the type of model (ML, LSTM, LM)
–language_model	string	Specify the language model to use
–params	key=value format	Specify the parameters of the ML algorithm

Table 16: Program arguments