Offensive Language Detection on Twitter

Mejbah Uddin Shameem
Saarland University
Student Id : 2577739
s8mesham@stud.uni-saarland.de

Dominik Weber
Saarland University
Student Id : 2548553
s9dowebe@stud.uni-saarland.de

1 Introduction

Twitter had 330 million users in the first half of 2019 and these users tweet 500 million tweets each day. But since Twitter is on the internet and everyone thinks they are anonymous, they post whatever comes to their mind. Some are angry and others just want to troll, but there are a lot of tweets which are offensive to others. Since there are 500 million tweets a day the tweets can not be checked manually before they are released and therefore offensive tweets are red by a lot of people before they got deleted based on the reports from other users. This is the motivation behind this project. When we get a classifier with satisfying accuracy we can flag offensive tweets and they can be checked manually or even deleted automatically if the classifier is accurate. The goal would be to make social media like twitter free of offensive posts, which would lead to a nicer community in the end. This project is an interesting challenge since it shows a real problem where the methods learned in the lecture can be used and tested what can be archived. The challenging part will be to classify offensive tweets which work with sarcasm or context to the main tweet which we do not have.

2 Support Vector Machine

Since we do most of our experiments with SVM and it is our solution for the final classifier we want to say something about the basics of SVM. SVM plots the data and tries to calculate a function that separates the data as good as possible into the different classes. With two classes, the SVM would calculate the border f and then predicts $P(Y|X)$ with the help with a look at the distance of X to the Hyperplane f. A negative distance means the first class positive distance the second. The hyperplane of a linear SVM can be written as for points x which satisfy the equation $< w * x + b = 0 >$ where w is a normal vector to the hyperplane and b the bias. Since we have only two classes we use a linear SVM and the only parameter we need to optimize is C. The C parameter tells the SVM how much you want to avoid misclassification. The SVM tries to find the Hyperplane with the maximum possible margin satisfying C.

3 Preprocessing

For preprocessing we used CMU TweetNLP first to create a new tsv file which included the tokens, POS tags, and the class. The next steps are we add whitespace to each emoji with the help of the emoji package to reduce the degree of sparsity in the data set. The tokens and emojis were the first features we used for our experiments. But to increase the accuracy of our classifier we tested different techniques. This include word and character ngrams, stop words, stemmer, lemmatization. This advanced techniques are described more closely in 6.3.

4 Our Basic Classifier

For our classifier, we tried different approaches. The first try was using different versions of Naive Bayer classifiers[1], where Complement Naive Bayes and Gaussian Naive Bayes get a similar accuracy which is close to 70% like the baseline classifier in the project description.

[1] https://scikit-learn.org/stable/modules/naive_bayes.html

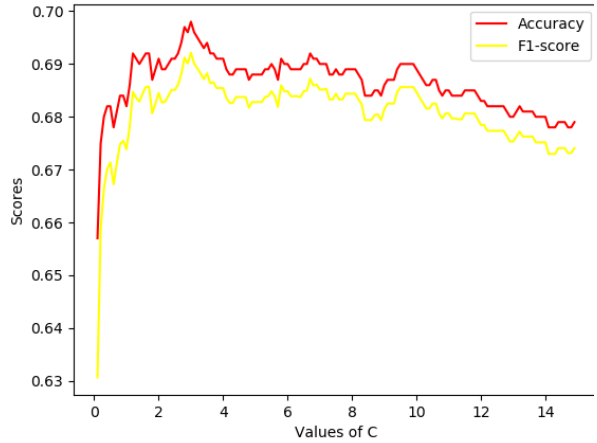

Figure 1: Count vectorized input

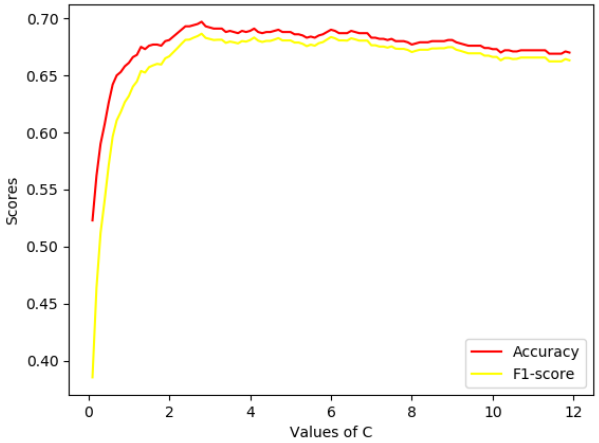

Figure 2: Tf-idf vectorized input

The second version we tried was SVM with the data inputted as count-vectorized matrices. Depending on the parameters, the accuracy went up to almost 70% without further feature processing.

From the above two plots, we observe the influence of different C values on the classifier and chose the best C value for the initial imbalanced data. The third method we tried was the neural network solution fasttext (Joulin et al., 2013) which was mentioned in the advanced analysis part of the project which also got about 70% accuracy but did not increase as much as the SVM with improved features, most likely caused by the small data set. The fasttext itself learned pretty fast and was easy to optimize but in the end, we thought that SVM would achieve a better result and therefor used SVM for the Advanced Analysis and our final classifier.

5 Error Analysis of Baseline

As far as we can see with the experiments on the basic classifier the biggest problems we have is to identify offensive tweets, while the classification of not offensive tweets works quite well. If we analyze the confusion matrix from figure 3 & 4 we find that for tf-idf vectorizer, the classification of not offensive tweets has an accuracy of 88% which is a good result for a basic classifier. The problem is that the accuracy for offensive tweets is 51% which is not a good result and is not far from tossing a coin.

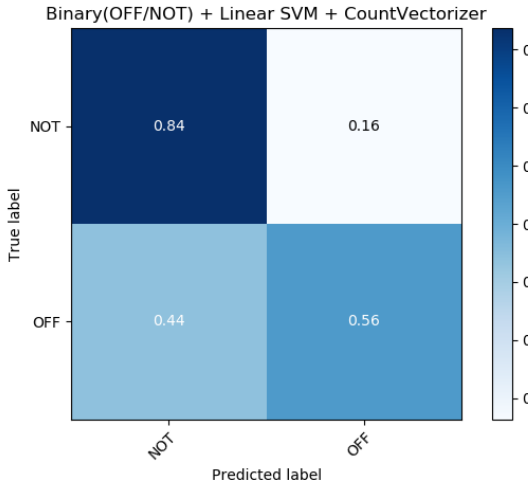

Figure 3: Confusion Matrix for Count vectorizer

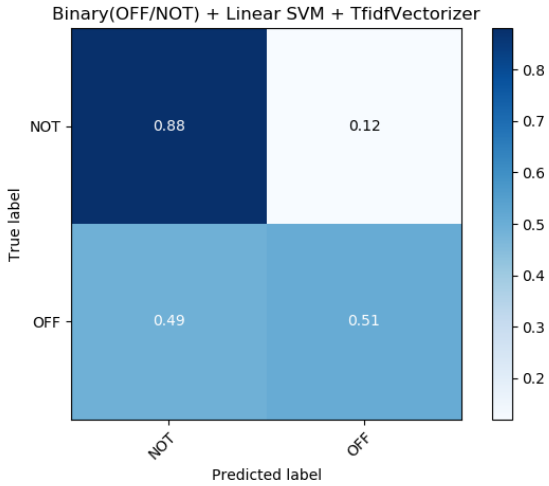

Figure 4: Confusion Matrix Tf-idf vectorizer

For this reason, we start our error analysis for the misclassified offensive tweets and in most cases,

it was misclassified for two reasons. The first reason is that the tweet contains words with a nice and positive meaning but used with a lot of sarcasm which could not be recognized by the classifier. The second reason would be the tweets that we see is mislabeled. These could be caused by a rule in the guidelines which was used to label them or they are only offensive in regard to the main tweet and/or the people linked users which are anonymous for us in this data set. Whatever the cause is, it is a problem since it is really hard to teach the classifier to classify something which we can not classify ourself because of missing data or context.

The misclassified not offensive tweets have similar problems. There are several tweets which contain negative words or emojis but are not offensive in the way the tweet was written. A deeper analysis of the errors and examples to these misclassified tweets can be found down below in the advanced analysis part (8).

6 Advanced Analysis and Improvement of Baseline SVM

6.1 Balancing Samples of Training Data

In the training dataset, we observed that the number of tweets which belongs to the NOT class is more than double from the OFF class. From the confusion matrix of our baseline classifier both for the count and tfidf vectorizer we see that the classifier is more successful to predict the NOT class (88%) but it misclassifies a lot of OFF samples (51%) of the development set. Because of the higher number of NOT class in training data, our baseline becomes more familiar with this particular class so on the development set it predicts well for this class. However, as the classifier gets less than half the number of OFF class to the other class it does not get all the context of this class, so it does not do well on the development set.

Therefore, we balanced the samples of the training data. We reduced the samples of NOT class and made it equal to the number of OFF class. By doing so we observed some mentionable changes in the confusion matrix. Though the prediction of NOT class is declined as expected, but the number of samples of OFF class which are classified correctly is raised significantly from 51% to 68%. This would make the classifier more stable and reliable, as in this particular problem we do not want anything like one class prediction is correct for 85% of the time and another class is wrongly predicted half of the time rather than we want something which makes some balancing like a percentage of 71% and 68% of accuracy for this two classes.

6.2 Fixing C Value for Balanced Data

We found that when we use SVM with count vectorizer in balanced training data we get highest accuracy for the C value 0.3 and for tfidf vectorizer the highest accuracy can be achieved with by setting the value of parameter C to 1.7. So we do our further analysis using this two different well tested value of C.

6.3 Feature Analysis

The features which were used in "Sentiment Analysis of Tweets" (Mohammad et al., 2013) are very promising and we also tried to implement some of them in our classification task. The features we tested are:

6.3.1 Word ngrams

We tested word ngrams ranging from uni to tri gram(1,3) with the balanced traning data. The accuracy improved when this feature was used with some other features.

6.3.2 Stop Word

Stop words which are most common and do not have any impact on classification can be preprocessed to make them uninfluential in the task. When these words are removed sometimes the classifier improves a bit. Our classifier also crosses the 70% mark of accuracy with count vectorizer when stop word feature is used with word ngrams and word stemmer.

6.3.3 Stemmer

Word and character stemmers are good features in this type of classification task which reduces the word verities to their root, base or stemmed form. In our task word stemmer and character stemmer with other features correctly predicts more than 70% of total samples in the development set. For tfidf vectorizer char stemmers work even more better touching almost the quality of our final proposed classifier.

6.3.4 Lemmatization

Unlike stemming, lemmatization finds out the parts of speech and meaning of a word in that context and group the forms of the word by the words lemma or dictionary form. In our classification task lemmatization with word ngrams gives accuracy below 70%.

6.3.5 POS

The number of occurrences of parts of speech tag was also examined in this task. It did not improve the classifier significantly.

6.3.6 Character ngrams

Character ngrams in balanced data with the ngram range 1 to 7 gave the best classifier for tfidf vectorizer when the value of C is 2.6. We got 71.4% accuracy and macro averaging F1-score of 71.3 with this feature which prediction is quite convincing regarding the percentage of correct class for each class. Our baseline was giving correct prediction for 85% of the time for one class and fails half of the time for another class but balanced data with character ngrams brings up some balance. From the confusion matrix we can observe that it gives prediction for OFF class for 76% tweets correctly whereas, the percentage was only 51% in the baseline. 67% percentage of the NOT tweets are predicted correctly which is not bad either while confirming high overall accuracy of the classifier.

7 Final Classifier

Table 1 lists the different feature set we experimented to get the best feature or feature set for the development set of data we are given. We found that some of the feature set are quite good in this classification task. Count vectorizer is almost as good as tfidf vectorizer when experimented with the feature set consists of word ngrams, word stemmer and stop word with the usage of balanced training data.

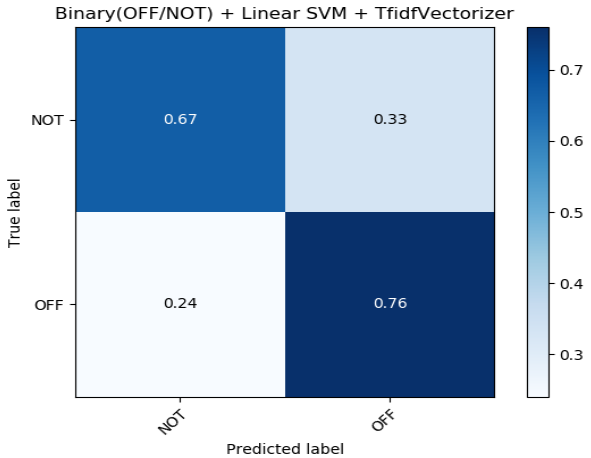

Figure 5: Confusion matrix of the prediction on development set

The accuracy of 71.3% which is just 0.1% less than the best feature set proves that this feature set is also pretty good and can be tested for future improvements. The feature set which consists of character ngrams, character stemmer and stop word is also very promising feature set for both tfidf and count vectorizer by confirming the both accuracy and F1-Score of more than 70%. Our proposed classifier which was implemented on the basis of character ngrams using SVM gave accuracy of more than 71%(71.4%) after meeting all the requirements of the project. The highest accuracy was achieved by setting the

Features	Count Vectorizer		TfidfVectorizer	
	Accuracy(%)	F1-Score (macro)	Accuracy(%)	F1-Score (macro)
Balanced Training Data (BTD)	68.8	68.6	69.4	69.3
BTD, Word ngrams	66.5	66.3	67.8	67.7
BTD, Word ngrams, Stop Word	65.7	65.5	66.2	66.1
BTD, Word ngrams, Word Stemmer, Stop Word	71.3	71.1	69.7	69.6
BTD, Word ngrams, Lemmatization	68.8	68.6	69.4	69.3
BTD, char ngrams	70.9	70.8	***71.4**	**71.3**
BTD, char ngrams, Stop Word	69.8	69.7	69.7	69.5
BTD, char ngrams, char Stemmer, Stop Word	70.1	70.0	71.3	71.2
BTD, POS	68.3	68.1	67.3	67.2

Table 1: Accuracy and F1-Score(macro) of different features on development set

value of the parameter C to 2.6 for tfidf vectorizer. The macro averaging F1-Score was also more than 71% which denotes the evidence of massive improvement of recall and precision of our baseline. The performance is significantly improved for the OFF class.

From Figure 5, we can observe that, 67% samples of the NOT class was correctly predicted by our final classifier whereas, prediction of the OFF class improved a lot from the baseline classifier by achieving the accuracy of 76%. After completing all the experiments on the development data we also merged the development data to the training data to train the final classifier to get better performance on the test data.

8 Challenges of Identifying Abusive Language & Error Analysis of final classifier

Categorizing abusive content is the first challenge that comes up with this type of classification problem. Though abusive terms are in general common but their meaning are not the same in all context. According to the analysis of (Vidgen et al., 2019) this is a social and also theoretical task and it is obviously true. Same language, exact same words can be offensive and non-offensive at the same time depending on who receives the message (tweets in our case) or who is targeted. This makes the task so challenging that only keeping a storage of bad and good words does not come handy.

Moreover, These texts are often unstructured and informal that make the task more challenging. For example, u r ma hon is used by some people to say you are my honey. The detection of abusive content is so varied that according to some it should be divided in subtasks consisting of individual directed abuse, Identity-directed abuse and concept directed abuse. Furthermore, the intention of the speaker, Humor, irony and sarcasm, Spelling variations, Polysemy (word with a single spelling but has multiple meanings), Long range dependencies, Incorrect Annotation, these all make the task even more difficult. Below we are presenting some tweets of our development set which were misclassified by our classifier:

* : Though for the balanced dataset if c=1.7 the classifier works best but for the character ngrams with Tfidf vectorizer we get 2.6 is the best value for C after testing it from the range of C values of [.1-15.0] with difference of 0.1 between neighbouring values

1. NOT @USER It would be so nice if the Trump supporters could tell us why they believe he is so great without comparing him to every criminal in the history of American politics. (predicted OFF)

2. NOT @USER I remember when the game just fucking started. (predicted OFF)

3. OFF @USER @USER Damn straight he is. Nothing but good vibes coming through November 💯💯💯👍👍👍 made it so. (predicted NOT)

4. OFF @USER Omg not even interested in his age but damn he is fine af 😜 (predicted NOT)

For the first tweet mentioned above, the annotated given class is NOT but our classifier predicts it to be in the OFF class. This has happened most probably due to the high weight of negative words than the positive or neutral words. In this tweet we observed that in the training data the words Trump, criminal, politics often used with the same tweet which contains some rough, bad words and slangs. This might be a reason why our classifier counting it as a sample of OFF class. Same goes with the second tweet. In the training set the word fucking comes with the offensive class quite always, hence the classifier predicts it to be in the OFF class while in this context the tweet does not aim to offend anyone or any groups.

We think for the third tweet there are so many words which are frequently used with the not offensive tweet on the training set. For example, words and emojis like good, vibes, 💯 (perfect), 👍 (thumbs up) these are often contained in the NOT class which are dominating here in this tweet. It might be the reason that the classifier does not get higher value for the true class OFF due to the use of the word damn.

The fourth tweet written above is also misclassified as NOT while its true label is OFF. However, we think this annotation might be incorrect. Because at the first look this tweet does not seem to be offensive at all. Then to become more confident to discuss this tweet in this report we showed this tweet to some people who are good at English language and understood this tweet quite well, they also discarded the possibility of this tweet being in the group of offensive class. So, this might be an example of incorrect annotation of the true class.

9 Further Analysis

Some other features could also be tested in our task. We did not consider tweet lengths while reducing the samples of the NOT class. The best length of tweets while reducing data can have some impacts on the classifier. In addition, the emojis presented in the tweets can be translated to verbal emotions via text string and grouping the emojis together with different type string may improve the classifier. For example, all type of sad, disgusted or bored emojis can be grouped together as a sad or irritated type. Moreover, cross validation method can also be applied to get more insights about the classifier.

10 Conclusion

We implemented SVM classifier for both balanced and imbalanced data for both count and tfidf vectorizer for offensive language detection on Twitter. In our implementation we experimented several features with different C values and came out with the best one which is based on character ngrams.

References

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. *Second Joint Conference on Lexical and Computational Semantics (*SEM)*, volume 2. Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), Pages:321-327.

Bertie Vidgen, Dong Nguyen, Rebekah Tromble, Alex Harris, Scott Hale, and Helen Margetts. Association for Computational Linguistics. 2019. *Proceedings of the Third Workshop on Abusive Language Online.* Florence, Italy. Pages:80-93

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification arXiv preprint arXiv:1607.01759