Detecting Offensive Language in Tweets Using Machine Learning Algorithm

1. Introduction

Online offensive language has become more frequent in recent years due to the rising use of social media platforms and easy access to the internet. This spread of offensive language on social media platforms leads to negative user experience online and hate crimes & violence in the real world. The key challenges in detecting offensive language are its context-dependent nature and the lack of agreement on what constitutes Offensive. Additionally, due to the scale of the internet, manual filtering is time and labor-intensive and, hence, impossible.

Thus, automated hate speech detection using machine learning algorithms is an important tool to detect offensive language at scale.

The rest of the paper is organized as Section 2 highlights the related works. Section 3 discusses the methodology, including Data Collection, Data Preprocessing, Data Splitting, Baseline Performance, Feature Enhancement, Hyperparameter tuning, and Final Model Construction and Evaluation. Sections 4 explain the Final settings, and results. Finally, Section 5 is discussion including the limitation, future work, and conclusion.

2. Related Work

Researchers in the past have proposed various machine learning-based text classification approaches to classify Offensive language content. They have implemented many feature extraction techniques, like Bag of Words, TF-IDF, Ngram, Word2Vec, and Doc2Vec. Machine learning algorithms often used for classification are logistic regression, Naive Bayes, decision trees, random forests, and linear SVMs (Support Vector Machine). Below we will discuss some of them briefly.

In the paper, Automatic offensive language detection from Twitter data using machine learning and feature selection of metadata[1], De Souza and Da Costa-Abreu have proposed a machine-learning approach based on the feature selection of metadata to deal with automatic offensive language detection on Twitter. The data used had a set of tweets in English prepared by Davidson et al. (2017) and manually labeled as either hateful, offensive, or normal through crowdsourcing. They discarded the hateful tweets due to a smaller sample size and then sampled offensive tweets to have a balanced data set. Then they split the data as 60% training and 40% testing set and built Linear SVM (LSVM) and Naïve Bayes classification models to detect offensive language. LSVM achieved 90% accuracy and 92% recall, whereas Naïve Bayes achieved 92% accuracy and 95% recall. LSVM was sensitive to data type and normalization and required proper parameters and a balanced input to attain good results. On the other hand, the Naïve Bayes classifier didn't have such issues and was easier to implement. Hence NB outperformed the SVM.

In another paper, Automatic hate speech detection using machine learning: A comparative study [2], authors compared the performance of three feature engineering techniques, namely, n-gram with TFIDF, Word2Vec, and Doc2Vec, and eight machine learning algorithms to evaluate the performance of the publicly available dataset compiled and labeled by CrowdFlower. The dataset had three types of tweets- Hate speech, Not Offensive, and Offensive but Not Hate Speech. In all 24 analyses, the lowest precision (0.58), recall (0.57), accuracy (57%), and F-measure (0.47) were found in MLP and KNN classifier using TFIDF features representation with bigram features. The highest recall (0.79), precision (0.77), accuracy (79%), and F-measure (0.77) were obtained by SVM using TFIDF features representation with bigram features.

Similarly, in another paper, Hate speech on Twitter: A pragmatic approach to collect hateful and offensive expressions and perform hate speech detection, Watanabe et al. [3] designed a J-48graft decision tree to detect hate speech in Twitter messages. He based the J-48graft approach on the unigrams feature representation model using the collected training dataset and used the extracted features as input into the J-48graft decision tree model for hate speech detection from Twitter messages. The evaluation test of the J48graft model showed better accuracy when compared to other baseline classifiers like Random Forest and SVM.

3. Methodology

The goal of this project is to differentiate between Offensive and non-Offensive tweets.

This section will outline the steps to classify tweets into Offensive and Not Offensive. The study approach illustrated in this figure 1 includes seven steps: Data Collection, Data preprocessing, Data splitting, Perform Baseline Experiments, Adding Additional Features, Hyperparameter Tuning, Final Model Construction, and Evaluation. I will go through each step-in detail in the subsequent sections:

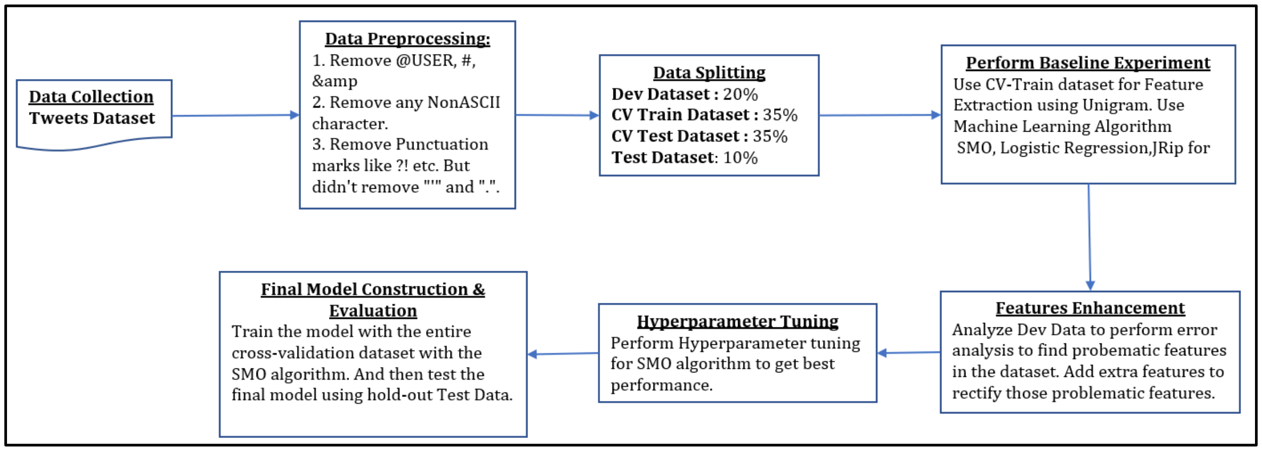

Data Collection
Tweets Dataset
Data Preprocessing:
1. Remove @USER, #, &amp
2. Remove any NonASCII character.
3. Remove Punctuation marks like ?! etc. But didn't remove "'" and ".".
Data Splitting
Dev Dataset : 20%
CV Train Dataset : 35%
CV Test Dataset : 35%
Test Dataset: 10%
Perform Baseline Experiment
Use CV-Train dataset for Feature Extraction using Unigram. Use Machine Learning Algorithm SMO, Logistic Regression,JRip for
Features Enhancement
Analyze Dev Data to perform error analysis to find probematic features in the dataset. Add extra features to rectify those problematic features.
Hyperparameter Tuning
Perform Hyperparameter tuning for SMO algorithm to get best performance.
Final Model Construction & Evaluation
Train the model with the entire cross-validation dataset with the SMO algorithm. And then test the final model using hold-out Test Data.

Figure 1: Steps Overview

Data Collection

For my analysis, I will use the Offensive Language Identification Dataset (OLID) compiled by Zampieri, Marcos, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar[4]. The dataset has 14,200 annotated English tweets using a fine-grained three-layer annotation scheme:

- A: Offensive Language Detection
- B: Categorization of Offensive Language
- C: Offensive Language Target Identification

OLID was the official dataset used in the OffensEval: Identifying and Categorizing Offensive Language in Social Media (SemEval 2019 - Task 6) shared task[5].

For annotation, the researcher annotated the data using crowdsourcing. They used Figure Eight and ensured data quality by (i) only hiring platform-experienced annotators and (ii) using test questions to discard annotations by individuals who did not reach a certain threshold. Further, the researcher acquired two annotations for each instance. In the case of disagreement, they requested a third annotation and then took a majority vote. Approximately 60% of the time, the two annotators agreed, and thus the researchers didn't need an additional annotation. But they requested a third annotation for the rest of the tweets. There was no instance when they had to ask for more than three annotations.

Data Pre-processing

Next, we will focus on data pre-processing to get clean data for analysis. The social media data does not follow standard writing practices. Thus, it often has issues like spelling mistakes and grammatical errors. Further, a tweet has a variety of characters, like usernames, hashtags, URLs, and emojis, that can confuse a text classifier. Hence, data pre-processing becomes an essential step before feeding the data into the machine-learning models.

For my analysis, I have primarily used Excel and VBA scripts in excel to cleanse the data. The steps are as follows:

- **Remove @USER, &amp, # symbol**: Words like @USER, &amp do not have any value, and hence removed.
- **Removal of NonASCII character**: Using Excel and VBA functions, I removed all the punctuation marks (except single quotes or periods) and special characters (like colon, semi-colon, ampersand, exclamation mark, and question mark) from the tweets.
- **Capitalization and word case**: Since I am using Lightside to do the classification, I didn't have to worry about the upper- or lower-case words. It is because Lightside automatically converts all the words to lowercase.

I removed most non-essential elements from the tweets with these pre-processing steps.

Data Splitting

Since my focus is to differentiate Offensive and Non-Offensive tweets, I will divide the data only into Offensive and Non-Offensive. The original dataset has ~14,000 tweets. Out of which, ~4,600 tweets are offensive, and ~8,800 tweets are non-offensive. Now, to have balanced data set, I have randomly selected a subset of non-offensive tweets of the same size as the set of offensive tweets. The resultant dataset has an equal number of offensive and non-offensive tweets, 4,640 tweets each and 9,280 tweets total.

Next, I will divide the data into Dev, Training, and Test datasets in the following proportions:

1. Dev Dataset: 20%
2. Cross-Validation Dataset: 70%
    a. Cross-Validation-Train Dataset: 35%
    b. Cross-Validation-Test Dataset: 35%
3. Test Dataset: 10%

Dataset statistics (Table 1) and features (Table 2) are as follows:

CLASS	Total Instances	Dev Instances	Cross-Validation Instances	Test Instances
Offensive	4640	940	3250	450
Not Offensive	4640	916	3246	478
Total	**9280**	**1856**	**6496**	**928**

Table 1: Dataset Statistics

COLUMNS	DESCRIPTION	POSSIBLE VALUES
ID	Unique ID column to identify every record	
Tweet	Tweet Posted by User	Text Values
Class	To classify tweet as Offensive or not	**OFF** : Offensive **NOT**: Not Offensive

Table 2: Dataset Features

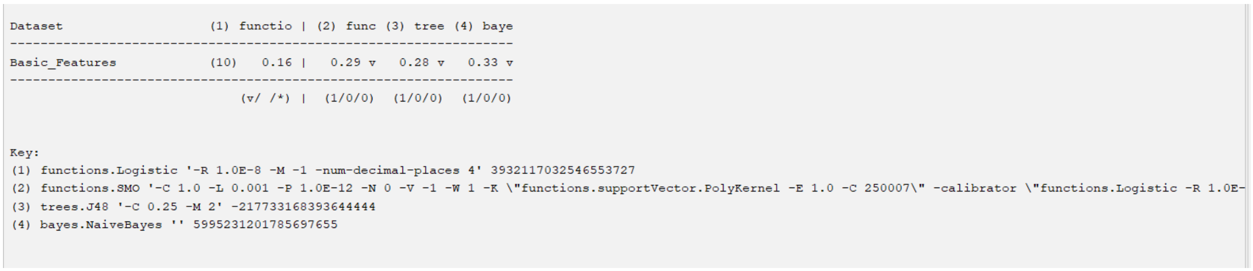
Perform Baseline Experiments

Next, I will perform the baseline experiment. For the Baseline Experiment, I will use the CV-Train dataset to extract basic features in Lightside, using Unigram as a feature extraction technique. Since no single classifier best performs on all datasets, I will evaluate a few different classifiers on a feature vector to identify the one that reaches the best results. I will evaluate three well-known text classifiers for the machine learning algorithm using a 10-fold cross-validation technique in Weka: Logistic Regression, Decision Tree,Naïve Bayes, and SVM (a.k.a SMO) with default parameters to find the best performance model. Table 3 shows the accuracy and Kappa of each model used.

Algorithm	Percent Correct	Kappa Statistics
Logistic Regression	58.251	0.1649
Decision Tree	64.1933	0.2841
SMO	64.4089	0.2888
Naïve Bayes	66.2254	0.3251

Table 3: Percent Correct and Kappa for different classifiers

Next, I repeated the experiment with the Experimenter to determine whether the difference in the performance of Logistic Regression, Decision Tree,Naïve Bayes, and SMO was statistically significant as shown in below mentione figure.



```
Dataset                   (1) functio | (2) func (3) tree (4) baye
------------------------------------------------------------------
Basic_Features            (10)    0.16 |   0.29 v   0.28 v   0.33 v
------------------------------------------------------------------
                              (v/ /*) |  (1/0/0)  (1/0/0)  (1/0/0)


Key:
(1) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
(2) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
(4) bayes.NaiveBayes '' 5995231201785697655
```

Figure 2: Comparison of Different Classifiers

Out of all Naïve Bayes performs the best. But since Naive Bayes doesnt not have a numeric parameter to tune. I am moving forward with the next best choice. In the Experimenter, the Decision Tree (0.28 kappa) and SMO (0.29 kappa) performed very similarly. SMO did well in predicting Not-Offensive tweets, whereas Decision Tree was great at predicting Offensive tweets. However, SMO took significantly less time to execute than the Decision Trees. Logistic Regression (0.16 kappa) performed the least among all. As a result, I've decided to go forward with SMO as the classifier.

Next, with the SMO as the algorithm, I trained the model with the CV-Train dataset and tested it on the CV-Test dataset on the light side. Table 4 shows the results:

Percent Correct	65.948276
Kappa	0.318553

Table 4: Baseline Percent Correct and Accuracy on CV Test Dataset

Note: *To find the best Machine learning algorithm and hyperparameter tuning I am using Weka. Whereas for feature selection and final model evaluation on test held-out data I have used Lightside. Hence, there will be a slight difference in performance.*

Features Enhancement

The Percent Correct and Kappa on CV-Train dataset is as follows:

Percent Correct	62.3152709
Kappa	0.24651569

Table 5: Baseline Percent Correct and Accuracy on CV Train Dataset

Next, I will run an error analysis on the Dev dataset using the SMO classifier. The idea is to identify features that might be confusing the classifier to misclassify. When using the Unigram feature extraction technique, 332 out of 940 cases are incorrectly labeled as Not-Offensive when they are Offensive. Some of the problematic features I have found in my analysis are as follows:

1. **Stopwords**: Most Stopwords have a positive weight and high frequency. These are common words that do not carry any meaning about the actual content of the tweet and can hamper the training process. Hence, we will remove them from the classification process.
2. **Stemming**: Some features are not in their base form and consider differently in different groups. For instance, words like Think and Thinks are classified differently, with one used in Offensive Instances and the other not.
3. **Regular Expression**: Lightside interprets "n't" separately from words like don't, doesn't, wasn't, etc. Once they are separated, it is impossible to tell the origin of "n't", which might have different connotations in tweets. Further, the features, like dont, doesnt, havent, wasnt, without a single quote are also treated separately.
4. **Unigram Features**: The Unigram only marks the presence of a single word within a text. It does not consider the contextual use of positive words in negative aspects and vice versa. For example, instances associated with the feature “antifa” has positive words like “friends”, “free”, “holiday”, and” Obama”. Therefore, despite the overall negative context of these instances, the prevalence of positive phrases may be a contributing factor in their labeling as Not-Offensive. It proves that employing Unigram as a feature extraction procedure fails to obtain the context. Hence, It is insufficient to classify the tweets.

To rectify above mentioned problematic features, I have selected new features extraction methods in the Lightside that are as follows:

1. **Ignore All-Stopwords N-Grams**: As mentioned above, Stopwords are irrelevant for Offensive language classification. Using this feature, we will remove all Unigram Stopword features from the feature set.
2. **Stem N-Grams:** In a tweet, words can be written in different forms like walk, walks, walking, etc. To reduce this divergence, I will use Stem N-Grams. With stemming, those words would be represented by a single “walk” feature, losing inflection but gaining generality.

3. **Regular Expression**: As seen earlier, Lightside splits word containing "n't" into two words and also treat the same word without quotes differently. To fix this, I will create a regular expression ".{2,4}n'?t" to consider words like don't, dont, wasn't, wasnt, etc, as the same.
4. **Character N-Grams & Word/POS Pair:** Due to Unigram's insensitivity to spelling and use variations, it fails to understand the context. To resolve this, I'm going to use Character N-grams in combination with Word/POS Pair to extract the contextual features of the text.

After implementing the above changes and extracting new features, the new Kappa and Percent Correct on the CV-Train dataset with SMO has improved compared to the baseline SMO model (performance on cross-validation train data is in Table 6).

Percent Correct	66.440887
Kappa	0.3288992

Table 6: Percent Correct and Kappa with new feature space on CV Train Dataset

However, the new model's performance on the test dataset was not better than the baseline performance. The percent correct and accuracy with the new model on the test dataset is as follows:

Percent Correct	65.8559113
Kappa	0.31704935

Table 7: Percent Correct and Kappa with new feature space on CV Test Dataset

Even though the new model's CV-Test performance is no better than the baseline, the new model improves prediction accuracy for Offensive situations. The baseline model predicted 1,009 Offensive instances correctly, whereas the new model correctly predicted 1,053 Offensive Instances. It is likely because the new features identified during the error analysis enabled the model to better detect the Offensive tweets but also led the model to misclassify Non-Offensive tweets as Offensive.

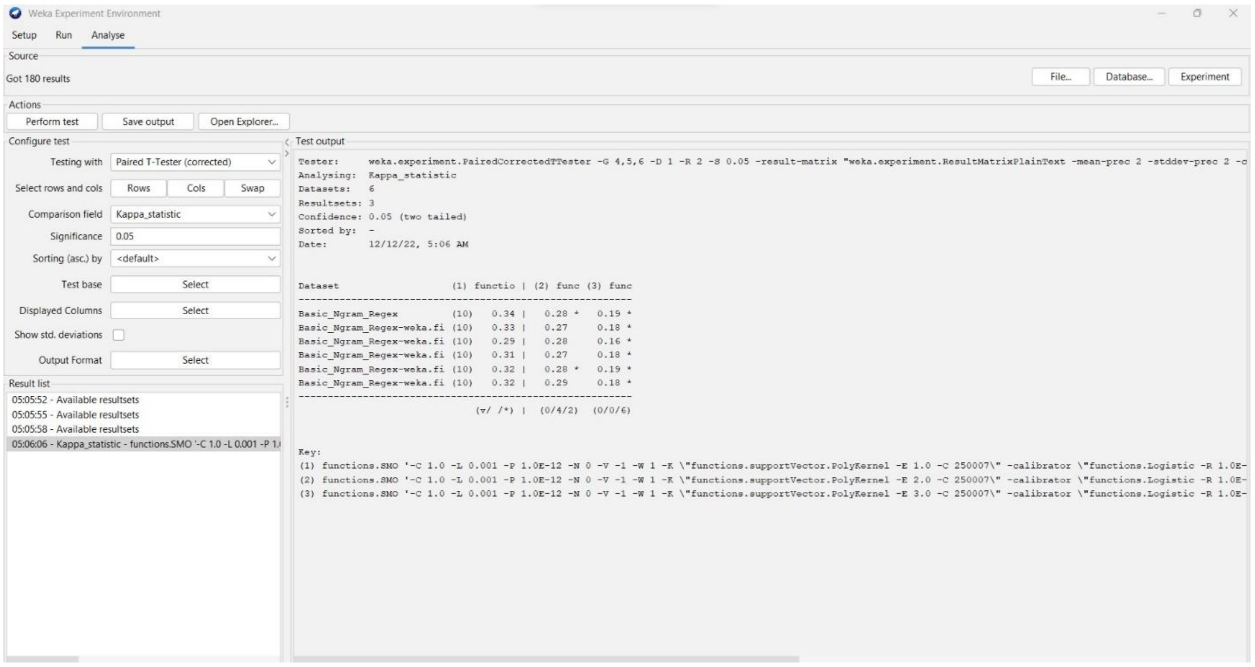
Hyperparameter Tuning

Now that I have the final feature set, my next step is to tune the SMO model to find the parameter setting that will give me the best performance for my dataset.

I have selected the parameter Exponent value in PolyKernal for hyperparameter tuning. I will experiment with three different exponent choices (1.0 (default), 2.0, and 3.0) to find the best value for the parameter. Since the parameter I have selected is in the nested option, I cannot tune it with the CV Parameter selection. Hence, I will use Train/test pair for tuning.

I have created train/test pair using Supervised Stratified Remove folds with five cross-validations. I have used the Weka experimenter to perform the steps of Hyperparameter Tuning. The steps I followed are as follows:

1. For stage 1, I will run cross-validation on the full cross-validation dataset for different Exponent settings. As shown in Fig2., the * next to the Kappa value for Exponent = 2.0 and Exponent = 3.0 tells us that it is significantly smaller than Exponent = 1.0. Hence, for stage 1, the best setting is Exponent = 1.0.
2. For Stage 3, the optimal setting for each inner fold is also E = 1.0. Figure 3. shows the Kappa value for each fold and every setting.



Source

Got 180 results

Configure test

Testing with: Paired T-Tester (corrected)

Comparison field: Kappa_statistic

Significance: 0.05

Sorting (asc.) by: <default>

Show std. deviations: ☐

Result list

05:05:52 - Available resultsets
05:05:55 - Available resultsets
05:05:58 - Available resultsets
05:06:06 - Kappa_statistic - functions.SMO '-C 1.0 -L 0.001 -P 1.0

Test output

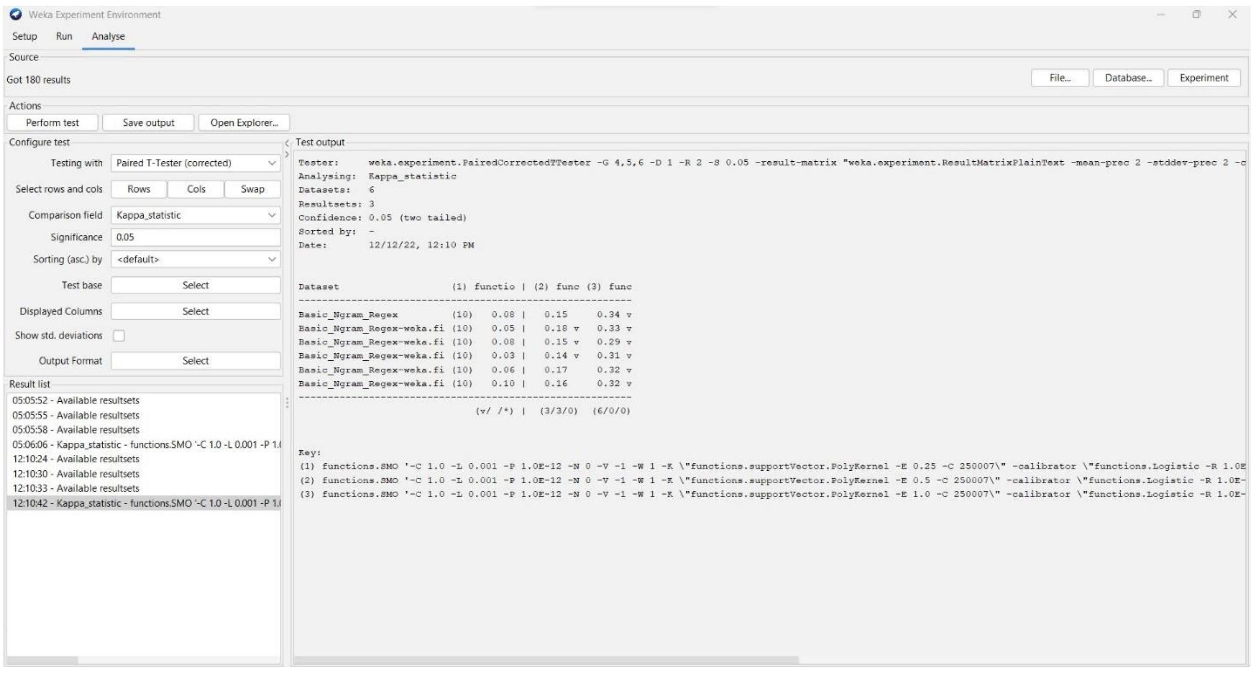
```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -c
Analysing:  Kappa_statistic
Datasets:   6
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       12/12/22, 5:06 AM


Dataset                   (1) functio | (2) func (3) func
----------------------------------------------------------
Basic_Ngram_Regex         (10)   0.34 |   0.28 *   0.19 *
Basic_Ngram_Regex-weka.fi (10)   0.33 |   0.27     0.18 *
Basic_Ngram_Regex-weka.fi (10)   0.29 |   0.28     0.16 *
Basic_Ngram_Regex-weka.fi (10)   0.31 |   0.27     0.18 *
Basic_Ngram_Regex-weka.fi (10)   0.32 |   0.28 *   0.19 *
Basic_Ngram_Regex-weka.fi (10)   0.32 |   0.29     0.18 *
----------------------------------------------------------
                              (v/ /*) |  (0/4/2)  (0/0/6)


Key:
(1) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
(2) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 2.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
(3) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 3.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
```

Figure 3: Analyze Tab Result for SMO Algorithm with E = 1.0,2.0,3.0

3. Since the optimal setting for each fold is the default value i.e., E = 1.0. I don't have enough evidence that the optimization will help improve the performance.
4. Since the low value of E = 1.0 is the default and best value for the SMO model for this dataset, I will further test the low range values for E = 0.25, E = 0.5, and E = 1.0 to check if the optimal value of E is <1.0 for this dataset.
5. As shown in Figure 3., the "v" next to the Kappa value for Exponent = 1.0 tells us that it is significantly higher than the other two settings. Also, for each inner loop, the optimal setting is E = 1.0. As a result, the optimal setting is again E = 1.0.



Source
Got 180 results

Configure test
Testing with: Paired T-Tester (corrected)
Comparison field: Kappa_statistic
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ☐

Result list
```
05:05:52 - Available resultsets
05:05:55 - Available resultsets
05:05:58 - Available resultsets
05:06:06 - Kappa_statistic - functions.SMO '-C 1.0 -L 0.001 -P 1.
12:10:24 - Available resultsets
12:10:30 - Available resultsets
12:10:33 - Available resultsets
12:10:42 - Kappa_statistic - functions.SMO '-C 1.0 -L 0.001 -P 1.
```

Test output
```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -c
Analysing:  Kappa_statistic
Datasets:   6
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       12/12/22, 12:10 PM


Dataset                     (1) functio | (2) func (3) func
------------------------------------------------------------
Basic_Ngram_Regex           (10)   0.08 |   0.15     0.34 v
Basic_Ngram_Regex-weka.fi (10)   0.05 |   0.18 v   0.33 v
Basic_Ngram_Regex-weka.fi (10)   0.08 |   0.15 v   0.29 v
Basic_Ngram_Regex-weka.fi (10)   0.03 |   0.14 v   0.31 v
Basic_Ngram_Regex-weka.fi (10)   0.06 |   0.17     0.32 v
Basic_Ngram_Regex-weka.fi (10)   0.10 |   0.16     0.32 v
------------------------------------------------------------
                            (v/ /*) |  (3/3/0)  (6/0/0)


Key:
(1) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 0.25 -C 250007\" -calibrator \"functions.Logistic -R 1.0E
(2) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 0.5 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
(3) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-
```

Figure 4:Analyze Tab Result for SMO Algorithm with E = 0.25, 0.5,1.0

Hence, the optimal parameter setting for the Exponent value is 1.0 for the SMO algorithm for this dataset.

4. FINAL RESULTS

Final Model Construction & Final Model Evaluation:

In the final step, I will train the model with the entire cross-validation dataset (CV-Train & CV-Test) with the SMO using E = 1.0 and then test the final model using hold-out Test Data.

The percent correct and Kappa for the Final held-out test data is as follows:

Percent Correct	66.27155172
Kappa	0.325650288

Table 8: Percent Correct and Kappa for held out Test Dataset

5. DISCUSSION

In this project, I explored a solution to detect Offensive Language in Tweets through Machine Learning Algorithm using Lightside as a Feature Extraction tool. I also performed a comparative analysis among multiple Machine Learning algorithms, like Logistic Regression, Decision Tree, Naïve Bayes, and SMO, using a basic set of features using Weka. The comparative analysis shows that the SMO algorithm outperforms Logistic Regression and Decision Tree in accuracy and execution time. Further, with SMO as the algorithm, I performed Hyperparameter Tuning for several Exponent value settings in SMO, and the outcome was that the default setting was the best. Upon evaluating the final model on held-out test data, I achieved 66.27% accuracy.

I also reviewed a few similar research papers, outlined in Section 2 - Related Work, and they used different feature extraction techniques like Unigram, TFIDF, Word2Vec, Doc2Vec, etc., to get the best prediction. However, none performed any error analysis to identify problematic features and engineer new features to correct the issues, as I did in this project. I started with Unigram as a feature extraction technique and later added four new features to improve the ability of the model to detect Offensive tweets. As a result, the performance to predict Offensive instances improved from 1,009 to 1,053.

My work also has a few limitations. Even though Naive Bayes provided the best performance, I chose SMO as my machine learning algorithm. I made this decision because Naive Bayes has no numeric parameter to tune in Weka. However, since Naive Bayes demonstrated better performance than SMO, I am anticipating using Naive Bayes would have given a better prediction accuracy.

For future work, I would like to explore other feature extraction techniques, like TF-IDF, Ngrams, and Deep Learning techniques, like Multilayer Perceptron, and compare their performance with Naive Bayes. Further, due to the time limitation, I had only tried sub-task A, classifying tweets as Offensive or Not-Offensive. In the future, I would like to explore the next sub-tasks, like identifying whether the tweets targeted specific individuals or groups. There has also been relatively little research in predicting Offensive Language outside the English language. That is another area that I would like to explore in the future.

6. REFERENCES

1. De Souza, Gabriel Araújo, and Márjory Da Costa-Abreu. "Automatic offensive language detection from Twitter data using machine learning and feature selection of metadata." In 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1-6. IEEE, 2020.
2. Abro, Sindhu, Sarang Shaikh, Zahid Hussain Khand, Ali Zafar, Sajid Khan, and Ghulam Mujtaba. "Automatic hate speech detection using machine learning: A comparative study." International Journal of Advanced Computer Science and Applications 11, no. 8 (2020).
3. Watanabe, Hajime, Mondher Bouazizi, and Tomoaki Ohtsuki. "Hate speech on twitter: A pragmatic approach to collect hateful and offensive expressions and perform hate speech detection." IEEE access 6 (2018): 13825-13835.
4. Zampieri, Marcos, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. "Predicting the type and target of offensive posts in social media." arXiv preprint arXiv:1902.09666 (2019).
5. https://sites.google.com/site/offensevalsharedtask/offenseval2019