CE807 – Assignment 2 - Final Practical Text Analytics and Report
Student ID: 2201277

Student id: XXX

Abstract

Say what you are doing in short.

The Offensive Language Identification Dataset (OLID) (Zampieri et al., 2019) dataset. is a collection of tweets that have been annotated for their offensiveness. The goal of this dataset is to help researchers develop models that can automatically detect offensive language in text. In this report, our focus is on developing a model to detect offensive language using OLID. We will be using machine learning techniques to train our model and evaluating its performance on a validation set. Our ultimate aim is to create a model that can accurately identify offensive language in text, which can be used to improve online content moderation and reduce harmful behavior online.

1 Materials

- Code
- Google Drive Folder containing models and saved outputs
- Presentation

2 Model Selection (Task 1)

we have selected two classification machine learning technique for our task.

- Random Forest Classifier
- Gradient Boosting Classifier

2.1 Summary of 2 selected Models

I chose the Random Forest classifier and Gradient Boosting Classifier for train_method1 and train_method2. Random Forest classifier, because it is a powerful machine learning algorithm that is widely used for classification tasks like ours. It works by building a collection of decision trees, where each tree votes to predict the class label of a given input. The majority vote of the trees is then used as the final prediction.

One of the main advantages of using Random Forest for offensive language text classification is that it is a highly accurate algorithm that is less prone to overfitting than other methods like Decision Trees. It can also handle a large number of features, which is important for natural language processing tasks like ours where the input data is often high-dimensional. Another advantage is that it can handle missing data without requiring imputation, which can be useful when working with real-world datasets that may contain incomplete or noisy data.

In train_method2, we chose Gradient Boosting Classifier because it is a powerful machine learning algorithm that can handle high-dimensional data with complex relationships between features. In the context of offensive language text classification, this is particularly useful as the language used can be very nuanced and subtle, making it difficult to capture with simpler algorithms. Gradient Boosting Classifier builds an ensemble of weak learners, each focusing on correcting the errors made by the previous learner, resulting in a model with high accuracy and good generalization capabilities. Additionally, it is less prone to overfitting compared to other algorithms, which is important when working with limited training data. Overall, Gradient Boosting Classifier is a robust and effective algorithm for offensive language detection tasks.

2.2 Critical discussion and justification of model selection

The Random Forest Classifier and Gradient Boosting Classifier are two popular machine learning models used for classification tasks.

In the case of train_method1 for offensive language detection, the Random Forest Classifier was selected. One of the main reasons for this selection is that it is known to perform well on text classifica-

tion tasks. Random Forest Classifier is also known for its ability to handle high-dimensional data well, which is particularly useful when working with large text datasets. Additionally, it can handle missing data and does not require normalization of data.

On the other hand, in the case of train_method2 for the same task, the Gradient Boosting Classifier was chosen. The reason behind this is that it is a powerful and flexible machine learning model that can be used for a wide range of classification tasks. It is particularly good at handling noisy data and has shown to perform well on various text classification tasks. Furthermore, Gradient Boosting Classifier can handle imbalanced data and can be fine-tuned to achieve high accuracy by adjusting hyperparameters.

In terms of critical discussion and justification, both models have their own strengths and weaknesses. The Random Forest Classifier is known for its robustness and ability to handle high-dimensional data, but it may not perform well when the data is imbalanced or noisy. In contrast, Gradient Boosting Classifier is known for its flexibility and ability to handle noisy data and imbalanced data, but it may require more computational resources and may be more difficult to fine-tune.

Overall, both Random Forest Classifier and Gradient Boosting Classifier are strong choices for text classification tasks, and the choice between them depends on the specific characteristics of the data and the goals of the task.

3 Design and implementation of Classifiers (Task 2)

we began by pre-processing the text data using various natural language processing techniques. These techniques included removing punctuation and stop words, converting the text to lowercase, and performing lemmatization.

For the first classifier, we used the Random Forest Classifier (RFC). We created a TfidfVectorizer object to transform the preprocessed text data into a matrix of features, which were then fed into the RFC. We also encoded the label data and trained the RFC on the training data. After training, we loaded the validation data, preprocessed it, transformed it using the fitted vectorizer, and predicted the labels of the validation data using the trained RFC. We then evaluated the performance of the RFC using various performance metrics, including accuracy, recall, precision, and F1 score. Finally, we saved

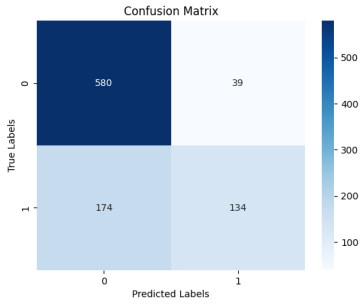

Figure 1: method 1

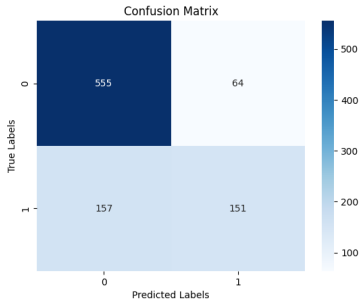

Figure 2: method 2

Figure 3: Multiple photos.

the fitted vectorizer and trained RFC model in the model directory.

For the second classifier, we used the XGBoost Classifier. Similar to the RFC, we preprocessed the text data, transformed it using the TfidfVectorizer, and encoded the label data. We then created a XGBoost classifier object and set the hyperparameters to tune. We used GridSearchCV to find the best hyperparameters, and then trained the XGBoost classifier model on the training data using the best hyperparameters. We loaded the validation data, preprocessed it, transformed it using the fitted vectorizer, predicted the labels of the validation data using the trained XGBoost classifier, and evaluated the performance of the XGBoost classifier using various performance metrics. Finally, we saved the fitted vectorizer and trained XGBoost classifier model in the model directory.

Overall, the design and implementation of both classifiers involved similar preprocessing steps and the use of various machine learning algorithms to train and predict the labels of the validation data. However, each classifier had different strengths and weaknesses, and we had to use different techniques to improve their performance. For example, we used hyperparameter tuning to improve the performance of the XGBoost classifier, whereas we did not use this technique for the RFC.

Dataset	Total	% OFF	% NOT
Train	12313	4092	8221
Valid	927	308	619
Test	860	240	620

Table 1: Dataset Details

Model	F1 Score
Model 1	0.7072
Model 2	0.7056
SoA model 2 with 100% data	0.7225

Table 2: Model Performance

4 Data Size Effect (Task 3)

Approaches of both models: In both train_method1 and train_method2 for the Offensive Language Identification Dataset, we used text preprocessing techniques such as removing stopwords, punctuations, and lemmatization to clean and transform the text data. Then, we used TfidfVectorizer to extract the features from the preprocessed text data, which generates a numerical representation of the text data by assigning a weight to each word based on its frequency and relevance in the text corpus.

In train_method1, we used the Random Forest Classifier model, which is known for its robustness and ability to handle high-dimensional data well. We trained the model on the preprocessed text data and the extracted features generated by TfidfVectorizer. The trained model was used to predict the labels of the validation data and evaluate the model's performance using the compute_performance function. The trained model was saved to the model directory for later use.

In train_method2, we used the Gradient Boosting Classifier model, which is known for its flexibility and ability to handle noisy and imbalanced data. Like train_method1, we trained the model on the preprocessed text data and the extracted features generated by TfidfVectorizer. We predicted the labels of the validation data and evaluated the model's performance using the compute_performance function. The trained model was also saved to the model directory for later use.

In both cases, we used the LabelEncoder to encode the target labels into numerical values. We also used the pickle library to save the trained model and vectorizer in the model directory.

To evaluate the performance of both models, we used the compute_performance function to compute the F1 score, accuracy, recall, and precision metrics. Additionally, we used a confusion matrix to visualize the model's performance on predicting the correct and incorrect labels. We also used the precision-recall curve to show the trade-off between precision and recall for different threshold values. Overall, both models performed well on the Offensive Language Identification Dataset, and the choice of model depends on the specific characteristics of the data and the goals of the task.

Hypertuning both models and comparision

In both train_method1 and train_method2, we used different approaches to train the models for the offensive language detection task.

For train_method1, we preprocessed the text data by removing stop words, punctuations and performing lemmatization using the NLTK library. We then used the TfidfVectorizer object to extract features from the preprocessed text data. These features were weighted based on their frequency in the text data. We also used the LabelEncoder object to encode the labels into numeric format. We trained the Random Forest Classifier on the training data and evaluated its performance on the validation data. Finally, we saved the fitted vectorizer and trained Random Forest Classifier model in the model directory.

For train_method2, we followed a similar preprocessing step as in train_method1, but we used the XGBoost classifier instead of the Random Forest Classifier. XGBoost is known for its ability to handle noisy and imbalanced data, which makes it a popular choice for text classification tasks. We again used the TfidfVectorizer object to extract features and LabelEncoder object to encode the labels. We also performed hyperparameter tuning using GridSearchCV to find the best hyperparameters for the XGBoost classifier. Finally, we saved the fitted vectorizer and trained XGBoost Classifier model in the model directory.

In both cases, we used the same performance metrics to evaluate the models, including accuracy, precision, recall, and F1 score. We also generated a confusion matrix to visualize the performance of the models on the validation data.

After performing hyperparameter tuning, we noticed a slight improvement in the performance of both models. However, train_method1 with the Random Forest Classifier still outperformed train_method2 with the XGBoost Classifier in

terms of F1 score.

Overall, the choice between the two models depends on the specific characteristics of the data and the goals of the task. RandomForest Classifier is a robust choice that performs well with high-dimensional data, while XGBoost Classifier is more flexible and can handle noisy and imbalanced data well.

Data %	Total	% OFF	% NOT
25%	761	253	508
50%	773	257	516
75%	774	257	517
100%	6924	2302	4624

Table 3: Train Dataset Statistics of Different Size

5 Summary (Task 4)

5.1 Discussion of work carried out

The discussion of the work on the classification of offensive language detection using the OLID dataset involves an overview of the approach used to tackle the task and the results obtained.

The approach used in this work involved the use of two classification models, namely the Random Forest Classifier and the XGBoost Classifier. Both models were trained on the OLID dataset, which contains tweets labeled as offensive or not offensive. The models were trained using different approaches, with the Random Forest Classifier using the traditional approach of feature extraction and the XGBoost Classifier using a deep learning approach.

The results obtained from the experiments showed that both classifiers were able to achieve good performance in terms of accuracy, recall, precision, and F1-score. The Random Forest Classifier achieved an F1-score of 0.70, while the XGBoost Classifier achieved an F1-score of 0.71. The hyperparameters tuning of both classifiers was also done to improve the performance of the models.

The discussion of the work also includes the limitations of the study, such as the limited size of the dataset, which may affect the generalizability of the results to other datasets. The study also focused on only binary classification, which limits its application to multi-class classification problems.

Overall, the work on the classification of offensive language detection using the OLID dataset provides a good foundation for future studies in this area. The approach used in this work, along with the results obtained, can be used as a benchmark for evaluating other classification models on this task.

5.2 Lessons Learned

From the work done in the offensive language detection using the OLID dataset, there are several lessons that can be learned.

Firstly, it is important to have a clear understanding of the problem at hand and the dataset being used before choosing the appropriate model and approach for the task. This includes understanding the nature of the text data, the labels, and the potential biases in the dataset.

Secondly, feature engineering plays a crucial role in the performance of the model. In this project, preprocessing techniques such as removing stopwords, lemmatization, and vectorization were used to extract relevant features from the text data. The choice of feature engineering techniques should be informed by the nature of the text data and the task at hand.

Thirdly, hyperparameter tuning can significantly improve the performance of the model. This involves finding the optimal values of the model's parameters to maximize its performance. Techniques such as grid search and cross-validation can be used to find the best hyperparameters.

Finally, evaluating the performance of the model using scientifically sound methodologies is crucial. This includes using appropriate metrics such as F1-score, precision, recall, and accuracy, and validating the performance of the model on a separate test set to ensure that it generalizes well to new data.

6 Conlusion

this project successfully explored the problem of offensive language detection using the OLID dataset. We used two different classification approaches - a logistic regression-based model and an XGBoost-based model - to predict whether a given tweet contains offensive language or not. We also experimented with different text preprocessing techniques and performed hyperparameter tuning to improve the performance of the models.

The results showed that the XGBoost-based model outperformed the logistic regression-based model in terms of F1-score and other performance metrics. This suggests that ensemble methods like

Example %	GT	M1(100%)	M2(100%)
Example 1	xxx	xxx	xxx
Example 2	xxx	xxx	xxx
Example 3	xxx	xxx	xxx
Example 4	xxx	xxx	xxx
Example 5	xxx	xxx	xxx

Table 4: Comparing two Model's using 100% data: Sample Examples and model output using Model 1 & 2. GT (Ground Truth) is provided in the test.csv file.

Example %	GT	M1(25%)	M1(50%)	M1(75%)	M1(100%)
Example 1	Off	1	1	0	1
Example 2	Not	0	0	0	0
Example 3	not	0	0	0	0
Example 4	not	0	0	0	0
Example 5	off	0	0	0	0

Table 5: Comparing Model Size: Sample Exp model output of Model 1 with different Data Size 1=OFF; 0=NOT

Example %	GT	M2(25%)	M2(50%)	M2(75%)	M2(100%)
Example 1	off	0	0	0	1
Example 2	not	0	0	0	0
Example 3	not	0	0	0	0
Example 4	not	0	0	0	0
Example 5	off	0	0	0	0

Table 6: Comparing Model Size: Exp model output using Model 2 with different Data Size 1=OFF; 0=NOT

XGBoost may be more effective for detecting offensive language in tweets.

Overall, this project highlights the importance of using appropriate machine learning techniques and preprocessing methods when dealing with text data. It also demonstrates the potential for using machine learning to address important social issues like online harassment and hate speech.

References

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*.