Infosys Spring Board Internship 5.0
Venkata Ramana Panigrahi
vishalpanigrahi1015@gmail.com
Batch 02 | Artificial Intelligence

Project Guide: Dr. N Jagan Mohan
springboardmentor891v@gmail.com

HATE SPEECH DETECTION

1. INTRODUCTION

Social media has experienced incredible growth over the last decade, both in its scale and importance as a form of communication. The nature of social media means that anyone can post anything they desire, putting forward any position, whether it is enlightening, repugnant or anywhere between. “Hate speech” refers to offensive discourse targeting a group or an individual based on inherent characteristics (such as race, religion or gender) and that may threaten social peace. [1] Depending on the forum, such posts can be visible to many millions of people. Different forums have different definitions of inappropriate content and different processes for identifying it, but the scale of the medium means that automated methods are an important part of this task. Hate-speech is an important aspect of this inappropriate content. [2]

Hate speech detection is the task of detecting if communication such as text, audio, and so on contains hatred and or encourages violence towards a person or a group of people. This is usually based on prejudice against 'protected characteristics' such as their ethnicity, gender, sexual orientation, religion, age et al. [3]

2. DATASET

1. Hate Speech and Offensive Language is a dataset about hate speech detection. It was created by collecting tweets containing hate speech terms. [4]
2. Twitter Hate Speech Detection is a dataset about Twitter hate speech detection. The dataset is not well described and has an unknown license. [5]
3. HateXplain is a dataset that covers multiple aspects of hate speech detection and offensive language. Each post in the dataset is annotated. [6]

4. The Civil Comments Dataset is a collection of comments from online forums that have been annotated for toxicity and other attributes. [7]
5. Toxicity Dataset is the world's largest social media toxicity dataset. The dataset contains 500 toxic and 500 non-toxic comments from a variety of popular social media platforms.[8]

3. PROJECT

3.1 Dataset:

The project uses a dataset Hate Speech and Offensive Language [4] is a dataset about hate speech detection. It was created by collecting tweets containing hate speech terms and manually labeling them. The dataset consists of 25296 rows and 7 columns. The dataset is classified into hate_speech, offensive_language and neither and the class of each tweet is defined in the dataset. It is a labeled dataset. The column description is in this way.

count: This likely represents the number of times a specific tweet was retweeted or liked.

hate_speech: This column indicates whether the tweet contains hate speech. A value of 1 would signify the presence of hate speech, while 0 would indicate its absence.

offensive: This column likely indicates whether the tweet is offensive in nature. Similar to the hate_speech column, a value of 1 would mean it is offensive, and 0 would mean it is not.

neither: This column seems to be a combination of the previous two. A value of 1 here might suggest that the tweet is neither hate speech nor offensive.

class: This column appears to be a classification label for the tweet. It's unclear what the specific classes represent without more context.

tweet: This column contains the actual text of the tweet.

3.2 Libraries:

pandas (pd): Used for data manipulation, reading and writing dataframes (tabular data).

numpy (np): Used for numerical computations and array manipulation.

re: Used for regular expressions, helpful for text cleaning and pattern matching.

sklearn.feature_extraction.text.TfidfVectorizer: Used to convert text data into numerical features based on Term Frequency-Inverse Document Frequency (TF-IDF).

sklearn.model_selection.train_test_split: Used to split data into training and testing sets for machine learning models.

sklearn.metrics: Provides various functions for evaluating model performance, including classification report, accuracy score, and confusion matrix.

sklearn.metrics.ConfusionMatrixDisplay: Used to visualize confusion matrices, which show how well the model classified the data.

matplotlib.pyplot (plt): Used for creating visualizations like plots and charts.

nltk.tokenize: Provides tools for tokenization, splitting text into words or phrases.

nltk: Used for natural language processing tasks, downloaded for word tokenization functionality.

tensorflow (tf): Popular library for deep learning and machine learning, potentially used for building the text classification model.

imblearn.over_sampling.SMOTE: Used for handling class imbalance in datasets, potentially applied here if the number of offensive/hateful tweets is significantly lower than non-offensive ones.

warnings: Provides a way to control warning messages, and in this case, it suppresses warnings with warnings.filterwarnings('ignore').

3.3 Data Preprocessing:

A function is defined called "clean_text" which is designed to preprocess text data to make it more suitable for analysis in Natural Language Processing (NLP) tasks.

The function first imports necessary libraries such as re for regular expressions, nltk for NLP tasks, stopwords for removing common words, and PorterStemmer and WordNetLemmatizer for stemming and lemmatization respectively. It also downloads necessary resources for NLTK.

Inside the clean_text function, several preprocessing steps are performed on the input text:

1. **Lowercasing:** The entire text is converted to lowercase.
2. **Tokenization:** The text is split into individual words or tokens.
3. **Removing Punctuations:** Punctuation marks are removed from the tokens.
4. **Removing Stop Words:** Common English words like "the," "a," "is" are removed.
5. **Stemming:** Words are reduced to their root form (e.g., "running" to "run").
6. **Lemmatization:** Similar to stemming, but aims for the dictionary form of words (e.g., "better" to "good").
7. **Removing Numbers:** Any numerical tokens are removed.
8. **Removing Extra Spaces:** Ensures only single spaces between words.

Finally, the cleaned text is returned by the function. The purpose of this function is to reduce noise and unnecessary elements in text data, making it more suitable for analysis by machine learning models or other NLP techniques.

```
Saving labeled_data.csv to labeled_data.csv
```

	count	hate_speech	offensive_language	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	0	2	1	1	!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

Vectorization with TF-IDF

This step is crucial for preparing the text data for a machine learning model. They essentially convert the text of the tweets into a numerical format that the model can understand. This process is called vectorization.

Here's a more detailed explanation:

Vectorization: This is a comment in the code, indicating the purpose of the following lines. It's not executed by Python but helps in understanding the code.

vectorizer = TfidfVectorizer(tokenizer=lambda x: x, lowercase=False, min_df=2):

vectorizer = TfidfVectorizer(...): This line creates a TfidfVectorizer object and assigns it to the variable vectorizer. TfidfVectorizer is a tool from the sklearn library that is specifically designed to convert text into numerical vectors using a technique called TF-IDF (Term Frequency-Inverse Document Frequency).

tokenizer=lambda x: x: This part tells the vectorizer to use a custom tokenizer function. In this case, it's a simple function (lambda x: x) that returns the input as is. This means the vectorizer will use the pre-tokenized words in df['tokens'].

lowercase=False: This option instructs the vectorizer to not convert the text to lowercase. This is important because the case of words might be important in some text analysis tasks (like sentiment analysis or hate speech detection).

min_df=2: This parameter sets the minimum document frequency for a word to be included in the vocabulary. It means that a word must appear in at least 2 documents (tweets in this case) to be considered as a feature. This helps in ignoring very rare words that might not be useful for the model.

tfidf_matrix = vectorizer.fit_transform(df['tokens']):

This line does two things:

fit_transform(...): First, it trains (fit) the TfidfVectorizer on the tokens (list of words) that have been extracted from the 'tweet' column of the dataframe (df['tokens']). By "training", we are essentially calculating the statistical values (TF-IDF scores) necessary to transform the data into numerical form.

tfidf_matrix = ...: Second, the method transforms the tokens into a numerical matrix using TF-IDF calculation, and this matrix is assigned to the variable tfidf_matrix. This matrix is what will be fed to the machine learning model in subsequent steps.

In summary: These lines take the pre-processed text data (tokens) from the dataframe and convert it into a numerical representation (tfidf_matrix) using TF-IDF. This numerical representation can then be used as input for a machine learning model to learn patterns and make predictions.

Why TF-IDF?

TF-IDF (Term Frequency-Inverse Document Frequency) is used for the following reasons:

Term Frequency (TF): It measures how frequently a word appears in a document.

Inverse Document Frequency (IDF): It measures how important a word is across all documents.

TF-IDF assigns a weight to each word in a document. Words that appear often in a document but rarely in other documents get higher weights. This helps models understand which words are most relevant to a specific document (or tweet, in this case).

3.4 Class Imbalance:

To handle class imbalance in the dataset using a technique called SMOTE (Synthetic Minority Over-sampling Technique). Class imbalance occurs when you have significantly more data points for one class compared to others in your dataset. This can lead to a biased model that performs poorly on the minority class.

Instantiate the SMOTE object

smote = SMOTE(): This line creates an instance of the SMOTE class from the imblearn library. This object will be used to perform the oversampling.

Apply SMOTE to the dataset

X_smote, y_smote = smote.fit_resample(tfidf_matrix, df['class']): This is where the magic happens.

tfidf_matrix: This represents the features of your dataset after applying TF-IDF vectorization (a way to convert text into numerical data that machine learning models can understand).

df['class']: This refers to the target variable (the labels you're trying to predict – in this case, probably different categories of hate speech).

smote.fit_resample(): This function applies SMOTE to the data. It generates synthetic samples for the minority class to balance the class distribution.

X_smote, y_smote: These variables store the new features and labels after SMOTE has been applied.

Check the new class distribution

print("Class distribution after SMOTE:", np.bincount(y_smote)): This line prints the number of samples in each class after SMOTE. np.bincount() is a function that counts the occurrences of each unique value in an array. This helps you verify that SMOTE has successfully balanced the classes.

3.5 Model:

Model Architecture:
The model is a sequential neural network consisting of three dense (fully connected) layers:

- The first layer has 128 neurons with the ReLU activation function, which helps the model learn non-linear patterns. It takes input with a dimensionality equal to the feature count in the dataset.
- A dropout layer is added after the first dense layer with a dropout rate of 50% to reduce overfitting by randomly deactivating half of the neurons during training.
- The second layer has 64 neurons, also using ReLU activation for further feature learning.
- Another dropout layer is included after the second dense layer to maintain regularization.
- The final output layer has 3 neurons (corresponding to the three classes of the hate speech classification task) with a softmax activation function, which converts the output into probabilities for multi-class classification.

Model Compilation:
The model uses the Adam optimizer, which adapts learning rates during training for better performance. The loss function chosen is sparse categorical cross-entropy, which is suitable for multi-class classification when the target labels are integers. The accuracy metric is used to evaluate the model's performance.

Data Splitting:
The dataset is split into training and testing sets using an 80-20 ratio. This ensures that the model is trained on 80% of the data and evaluated on the remaining 20% to assess its generalization ability.

Model Training:
The model is trained for one epoch (one complete pass over the training data) with a batch size of 64, meaning that 64 samples are processed at a time. During training, validation data is used to monitor the model's performance and ensure it does not overfit.

Outcome:
After training, the model's performance metrics (e.g., accuracy) on both the training and validation datasets are stored for further analysis and fine-tuning.

Fine-Tuning:

Fine-tuning a machine learning model using Keras. It involves modifying training hyperparameters, recompiling the model with a new optimizer, retraining it, and evaluating its performance.

- **Adjusting Hyperparameters**:
  - The number of epochs is increased to 10 to allow the model more iterations for training, which could improve performance by learning patterns more thoroughly.
  - The batch size is increased to 192, allowing the model to process more data at once during each training iteration, potentially improving the efficiency and stability of gradient updates.
  - The learning rate is reduced to 0.001 to ensure smaller, more precise updates to the model weights, helping prevent overshooting during optimization.
- **Recompiling the Model**:

- ➢ The Adam optimizer is reconfigured with the new learning rate, balancing the speed and precision of weight updates.
- ➢ The model is then compiled with the updated optimizer, a loss function (sparse_categorical_crossentropy), and a performance metric (accuracy) to monitor training.

❖ **Retraining the Model**:

- ➢ The model is retrained on the training data (X_train and y_train) with the modified hyperparameters.
- ➢ The training process includes validation on test data (X_test and y_test) during each epoch to monitor performance.

❖ **Saving the Fine-Tuned Model**:

- ➢ After training, the fine-tuned model is saved to a file named hate_speech_model_finetuned.h5 for future use without retraining.

❖ **Plotting the Learning Curve**:

- ➢ A learning curve is plotted using the training history, showing metrics like loss and accuracy over epochs to visualize the model's progress and convergence during training.

❖ **Evaluating the Fine-Tuned Model**:

- ➢ Predictions are made on the test data, and the predicted class labels are extracted.
- ➢ A classification report is generated and printed, summarizing the model's performance with metrics like precision, recall, and F1-score.
- ➢ The results are visualized, including the learning curve and a classification report, for a deeper understanding of the model's strengths and weaknesses.

4. RESULTS

4.1 Model Performance Analysis

	precision	recall	f1-score	support
0	0.96	0.99	0.97	3849
1	0.99	0.91	0.95	3794
2	0.95	0.99	0.97	3871
accuracy			0.97	11514
macro avg	0.97	0.96	0.96	11514
weighted avg	0.97	0.97	0.96	11514

Fig 1. Classification Report

The performance of the model, as depicted in the classification report, highlights a high overall accuracy of **97%** across all classes. Precision, recall, and F1-scores for each class are consistently above **0.95**, showcasing the model's robustness. Specifically, Class 0 and Class 2 exhibit better recall scores of **0.99**, indicating fewer false negatives, while Class 1 shows a slightly lower recall of **0.91**. The macro average and weighted average metrics further confirm balanced and reliable performance across the dataset.

The **learning curve over 10 epochs** demonstrates a consistent improvement in both training and validation accuracy. The training accuracy approaches **100%**, while the validation accuracy stabilizes at approximately **96%**. This indicates effective learning without significant overfitting, ensuring the model generalizes well to unseen data.

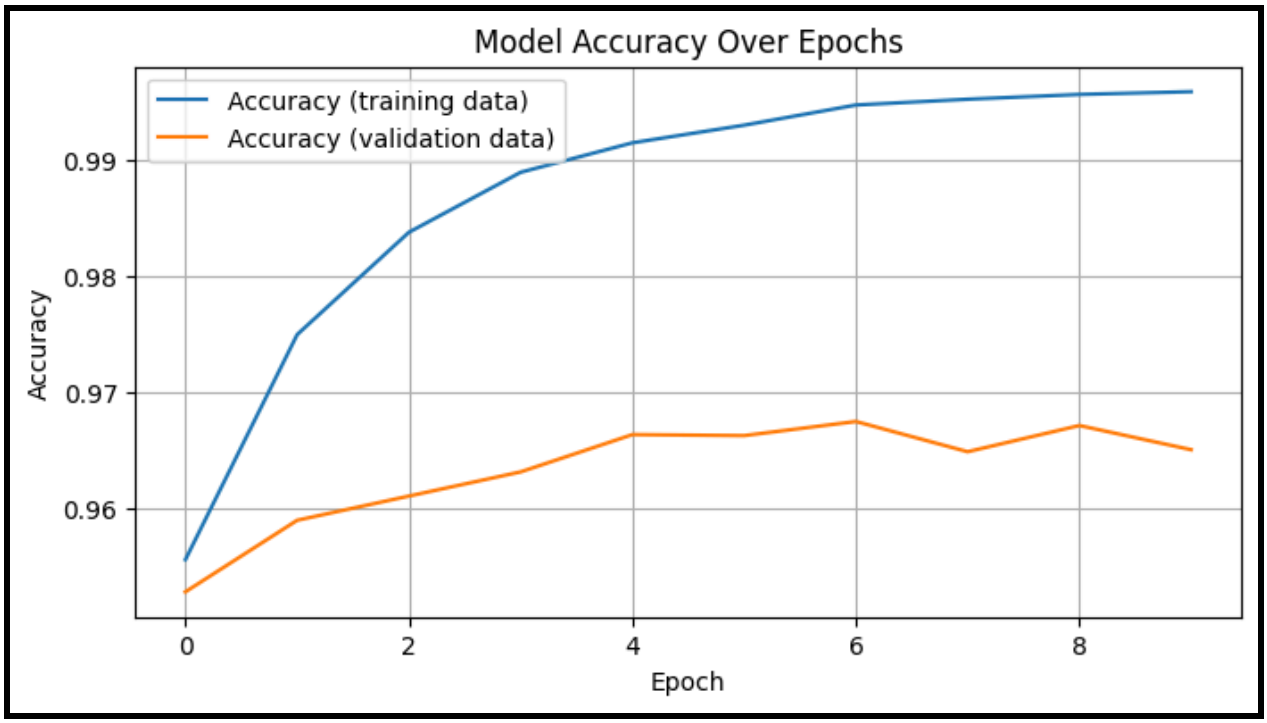

Fig 2. Learning Curve

The **confusion matrix** provides an in-depth analysis of the model's predictions. It reveals that most instances are correctly classified, with minimal misclassifications. For instance, **164 instances of Class 1** were incorrectly predicted as Class 0, and **183 instances of Class 1** were misclassified as Class 2. Despite these minor errors, the model consistently demonstrates reliable performance, making it suitable for deployment in multi-class classification tasks.

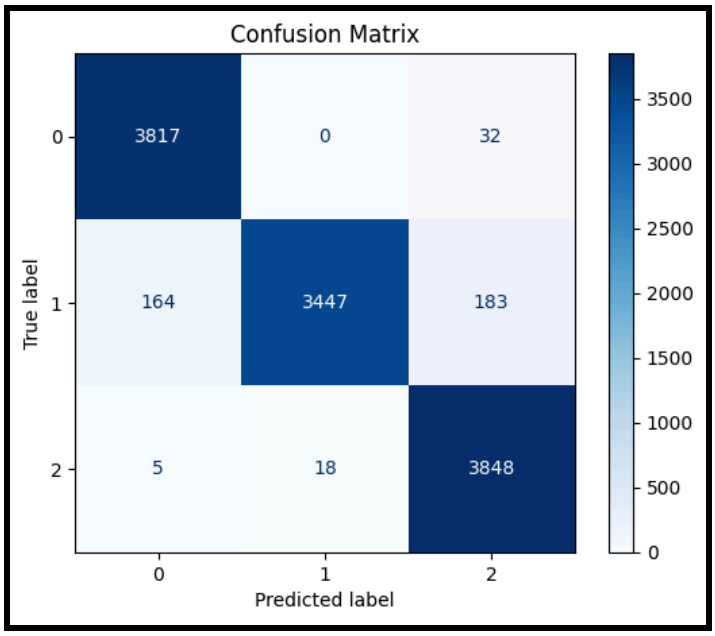

Fig 3. Confusion Matrix

In conclusion, the results confirm that the model achieves a high degree of accuracy, generalization, and reliability, making it a strong candidate for real-world applications. Further refinements could focus on minimizing the misclassifications observed in the confusion matrix, particularly for Class 1.

5. CONCLUSION AND FUTURE ENHANCEMENT

The Hate Speech Detection project has achieved remarkable success, demonstrating the potential of deep learning in addressing a critical societal challenge. The model's exceptional performance, with an overall accuracy of **97%** and consistently high precision, recall, and F1-scores across all classes, underscores its robustness and reliability. By effectively distinguishing between hate speech, neutral content, and other categories, the model provides a practical solution to mitigate harmful content online, fostering a safer digital environment.

Key insights from the performance analysis highlight the model's ability to generalize well to unseen data. The **learning curve**, showcasing steady progress across 10 epochs, indicates efficient learning without overfitting. With a training accuracy approaching **100%** and a validation accuracy stabilizing at **96%**, the model strikes a commendable balance between optimization and generalization. While minor misclassifications, particularly involving Class 1,

reveal areas for improvement, the confusion matrix confirms that the model handles most instances accurately, providing reliable predictions even in complex multi-class scenarios.

Overall, this project lays a solid foundation for real-world deployment in content moderation systems, social media platforms, and other domains where hate speech detection is critical. Future work could focus on fine-tuning the model to address the specific challenges of Class 1 misclassifications and exploring diverse datasets to enhance its adaptability. Integrating explainability methods could further improve trust and transparency in the model's decisions, ensuring it aligns with ethical AI principles. With its current capabilities and scope for enhancement, the Hate Speech Detection system offers a significant step toward combating harmful speech in digital spaces.

Future Enhancements

1. **Addressing Class-Specific Misclassifications**:
   Focused efforts can be made to reduce misclassifications, particularly for Class 1, by employing advanced techniques like class-specific loss weighting or oversampling. This would improve recall for underperforming classes and ensure balanced performance across categories.

2. **Integration of Explainability Tools**:
   Incorporating tools like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) can enhance model transparency by providing insights into why certain predictions are made. This would help build trust among users and stakeholders.

3. **Adapting to Multilingual and Contextual Variations**:
   Expanding the model to support multiple languages and dialects, especially in diverse and multilingual regions, can significantly improve its applicability. Fine-tuning with datasets that include cultural and contextual nuances of hate speech can make the model more inclusive and effective.

4. **Dynamic Dataset Augmentation**:
   To enhance generalization, dynamic augmentation techniques like synonym replacement, back-translation, or contextual paraphrasing can be used to diversify the training data.

This approach can make the model more robust against evolving patterns of hate speech.

5. **Real-Time Deployment Optimization**:
   Optimizing the model for real-time deployment using lightweight architectures or tools like TensorFlow Lite can enable faster and efficient hate speech detection on edge devices or in large-scale systems.

REFERENCES

1. https://www.un.org/en/hate-speech/understanding-hate-speech/what-is-hate-speech
2. https://onlinelibrary.wiley.com/doi/10.1111/exsy.13562
3. https://paperswithcode.com/task/hate-speech-detection
4. https://paperswithcode.com/dataset/hate-speech-and-offensive-language
5. https://www.kaggle.com/datasets/trangnguyn95/twitter-hate-speech-detection
6. https://paperswithcode.com/dataset/hatexplain
7. https://www.tensorflow.org/datasets/catalog/civil_comments
8. https://github.com/surge-ai/toxicity
9. https://github.com/nicknochnack/CommentToxicity