Data Science Intern at Data Glacier

Project: Hate Speech Detection using Transformers (Deep Learning)

Team Member: Manhui Zhu

Email: zmanhui09@outlook.com

Country: China

College: University of Southern California

Specialization: Data Science

Table of Contents

1. Project Plan

Weeks	Date	Deliverables
Week 7	June 19, 2024	Problem Statement, Data Intake Report, Project Plan
Week 8	June 26, 2024	Data Preprocessing
Week 9	July 2, 2024	EDA (Exploratory Data Analysis)
Week 10	July 9, 2024	Feature Extraction
Week 11	July 16, 2024	Model Building and Training
Week 12	July 23, 2024	Model Performance Evaluation
Week 13	July 30, 2024	Final Submission (Slides + Report + Code)

2. Problem Statement

The term hate speech is understood as any type of verbal, written or behavioral communication that attacks or uses derogatory or discriminatory language against a person or group based on what they are, in other words, based on their religion, ethnicity, nationality, race, color, ancestry, sex or another identity factor. In this problem, we will take you through a hate speech detection model with Machine Learning and Python.

Hate Speech Detection is generally a task of sentiment classification. A model that can classify hate speech from a certain piece of text can be achieved by training it on a data that is generally used to classify sentiments. For the task of hate speech detection model, we will use the Twitter tweets to identify tweets containing Hate speech.

3. Data Intake Report

Name: Twitter Hate Speech
Report date: 06/19/2024
Internship Batch: LISUM33
Version: 1.0
Data intake by: Manhui Zhu
Data Intake reviewer: Data Glacier
Data Storage location: https://github.com/Manhui-z/Data-Glacier-Internship/tree/0083a551094656a2b96e6b2b64fd353394d34756/Week%207

Tabular data details:

Name of data	hate_speech.csv
Total number of observations	31962
Total number of features	3
Base format of the file	.csv
Size of the data	2.95 MB

Proposed Approach:

- The full dataset is consisting of 3 features: `id` with data type int64, `label` with data type int 64, and `tweet` with data type object.
- There is no missing value in the dataset.

4. Data Preprocessing

There are mainly 4 approaches to transform raw text into a structured format, making it easier for models to analyze and learn from the data. They are text cleaning, removing stop words, tokenization, and lemmatization.

4.1 Text Cleaning

The usual tweets have many causal colloquial expressions, special characters, and emojis. These messy texts make it difficult for the model to learn the underlying pattern and classify the hate speech and non-hate speech. Therefore, we need to clean the text first before we fit data into the model for training. Here are detailed steps.

- **Lowercasing:** For the same words apple and Apple, the computer will recognize them as different words. To avoid this from happening, we need to all words in lowercase.
- **Removing User Mentions:** @Users is used when we mentioned someone in our tweets. It usually doesn't have any special meanings, so we remove @Users by using `re` (regular expression) package.
- **Removing URLs:** Since the model cannot directly interpret whether the content represented by the URLs is problematic, it is not helpful for the model training, so we remove it.
- **Removing Special Characters:** For better text understanding, we remove special characters in tweets by keep only letters, digits, and whitespace.
- **Removing leading and trailing whitespace:** we remove meaningless whitespace before and after each tweet.

4.2 Removing Stop Words

Stop Words are some high-frequency common words in English language expression like 'and', 'the', 'is', but they may not contribute to the meaning and context of the sentence. We use `nltk` library to help remove the stop words. This reduces number of words the model needs to handle so that models can focus on more meaningful words, which improves efficiency and reduce noise.

4.3 Tokenization

Tokenization splits text into smaller units called token, it is usually in units of words. It is the foundation for other steps in NLP task, like stemming, lemmatization, and vectorization, which take tokens as their input.

4.4 Lemmatization

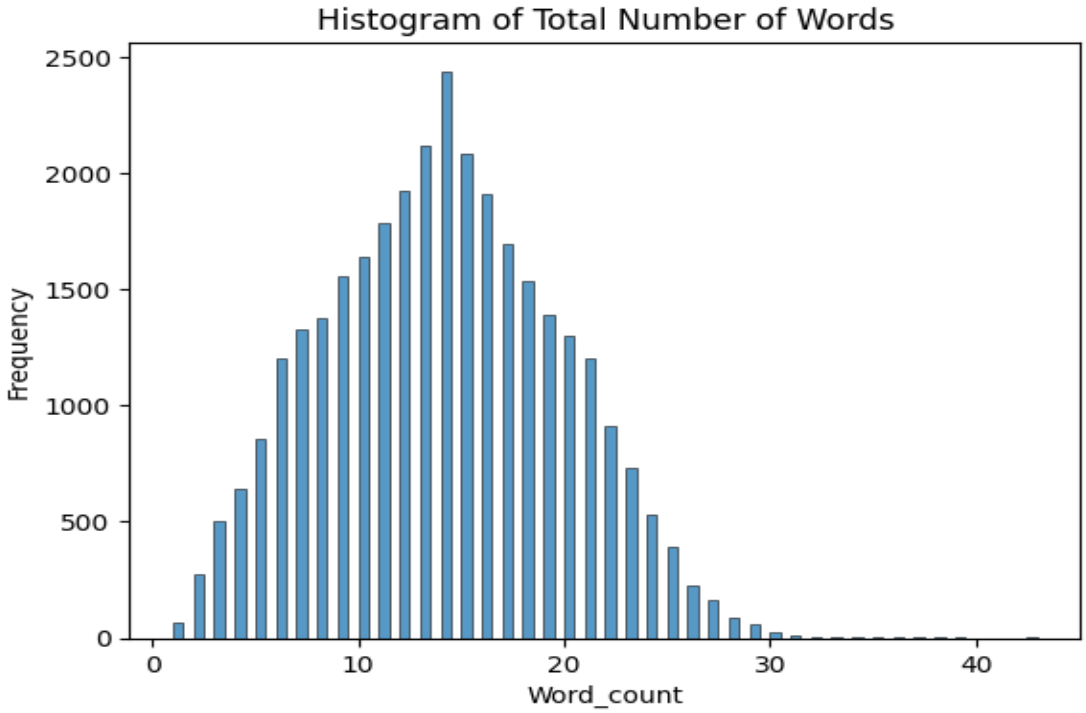
Lemmatization reduces words to their base or root form by considering the context and meaning of the word. It leads to better consistency in features, reduces redundancy, and helps in normalizing text, which is useful for classification task that requiring semantic understanding.

5. EDA (Exploratory Data Analysis)

In this part, we do the simple analysis to understand the data's structure and patterns. We check the number of total words in tweets and find the most frequents words appeared in hate and non-hate tweets. Then, we check the class distribution and use resampling technique to solve the imbalance issue.

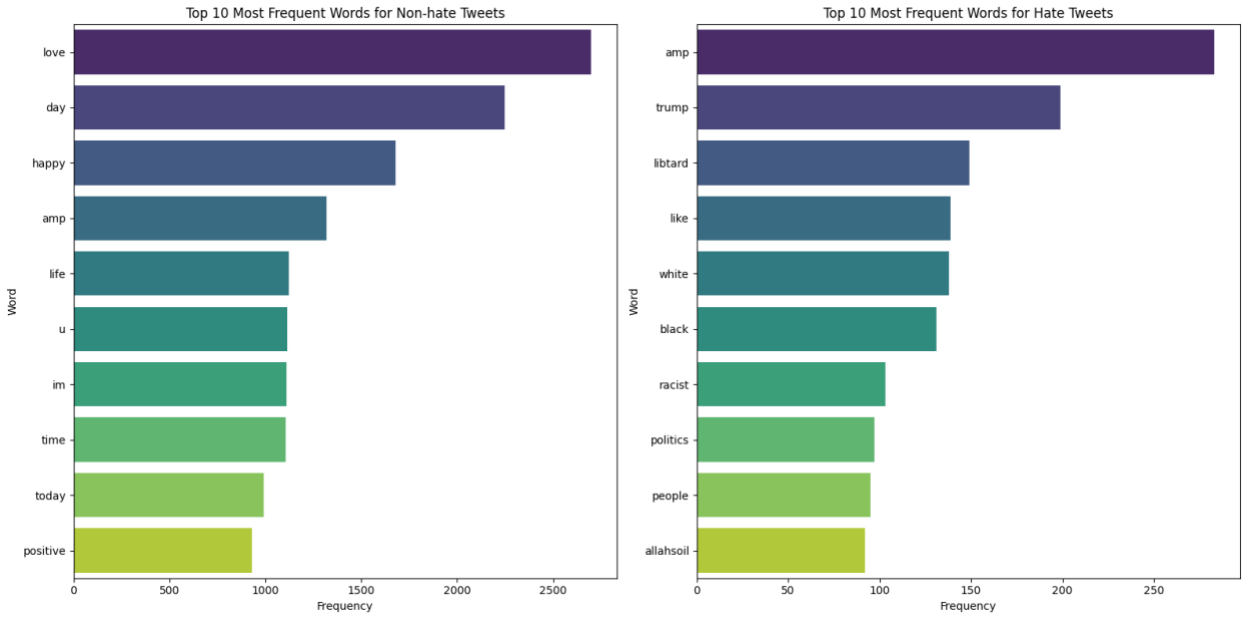
5.1 Number of Total Words and Stop Words

We check total number of words and stop words in each tweet. More than half of tweets have number of words in between 10 and 20.



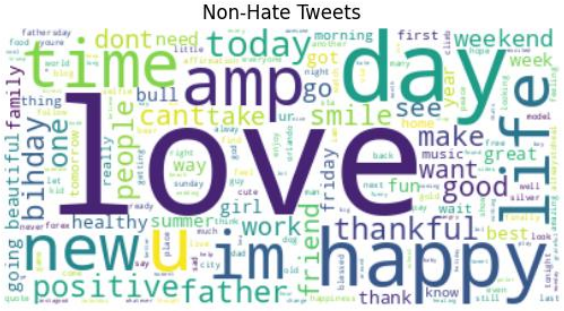
5.2 – 5.3 The Most Frequent Words in the Whole Dataset, Hate Tweets, and Non-hate Tweets

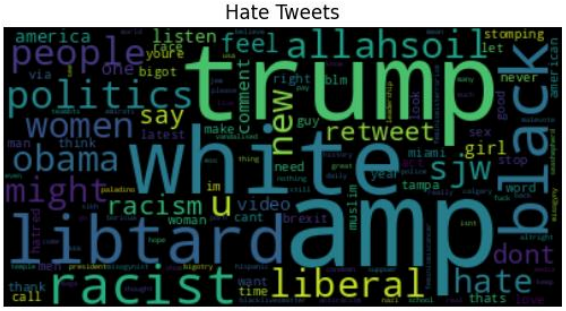
After removing the stop words, the top 10 frequent words in the whole dataset are 'love', 'day', 'happy', 'amp', 'im', 'u', 'life', 'time', 'like', 'today'. The top 10 frequent words in hate and non-hate speech are in the following graph. The word 'amp' appeared frequently in both classes. But it only means the symbol & (an ampersand), which usually appeared in HTML codes. Since our dataset may collected by directly web scraping from Twitter, it may convert symbol '&' into string 'amp'. Therefore, this word does not have significant impact for the model's understanding of the context and meaning of tweet. We can either keep it or remove it.



5.4 Word Clouds for Each Label

To better visualize the high-frequent words in each class, we plot the word cloud for each class. The larger the size of word, the more frequently it occurs. The graph below shows that ‘love’, ‘time’, ‘day’, ‘happy’, ‘life’ are high frequent words in non-hate tweets. These words are either positive or at least neutral. While in hate tweets, the high-frequent words are ‘trump’, ‘white’, ‘politics’, ‘black’, and ‘racist’. These words are associated with politics and race, which is usually controversial and prone to disputes.





5.5 – 5.6 Data Distribution & Solving Imbalance Issue

There are 29720 non-hate tweets and 2242 hate tweets. It indicates the imbalance in our dataset. The models trained on imbalanced data often become biased towards the major class, resulting in incorrectly classifying minority class instances. It will also cause underfit of the minority class, leading to poor model performance on minority instances. To avoid this problem, we resample

the minority samples (label = 1) with replacement so that it reaches the same number of samples of majority class (label = 1). After resampling, we have 29720 samples in each class.

6. Feature Extraction

In this section, we transform raw text data into structured numerical representations that machine learning models can process. By extracting meaningful features, we can reduce the data dimension, making the model more efficient and faster to train. Here are techniques we used to do the feature extraction.

6.1 Bag of Words (BoW)

This method converts text into a matrix of token counts (numerical representations), ignoring grammar and word order but keeping multiplicity. The intuition behind this method is that similar text fields will contain similar kind of words, therefore having similar bag of words. It has three steps:

- **Tokenization**: Splitting the text into individual words and tokens
- **Vocabulary**: Creating a set of unique words (tokens) from the entire text corpus
- **Vectorization**: Converting each document into a vector where each dimension corresponds to a word from the vocabulary and the value represents the word's frequency in the document.

6.2 TF-IDF (Term Frequency – Inverse Document Frequency)

This method evaluates the importance of a word in a document relative to a collection of documents (corpus), reducing the weight of common words.

- **Term Frequency (TF):** Measures how frequently a term occurs in a document. The assumption is that terms that appear more frequently within a document are more important.

$$TF(t,d) = \frac{\text{Number of term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse Document Frequency (IDF):** Measures how important a term is within the entire corpus. It helps to reduce the weight of terms that occur very frequently in many documents and increase the weight of terms that occur rarely.

$$IDF(t,D) = \log\left(\frac{\text{Total Number of documents}}{\text{Number of documents containing term } t}\right)$$

- The TF-IDF score is the product of TF and IDF:

$$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

6.3 Word Embeddings (*Word2Vec*)

This method represents words in a dense vector space, capturing semantic relationships. Words with similar meanings are located close to each other in this vector space.

Word embeddings are typically trained on large text corpora using neural networks. Here, we use the pre-trained `Word2Vec` models developed by Google (`GoogleNews-vectors-negative300.bin`), which contains vectors for a large vocabulary of around 3 million words and phrases. Each word vector is represented in a 300-dimensional space.

Twitter language is full of slang, abbreviations, and emojis. Word2Vec can learn these informal and evolving expressions, leading to accurate detection of hate tweets.

6.4 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a pre-trained transformer-based model designed for a wide range of NLP tasks. It is known for its ability to understand the context of a word in a sentence, reading text in both direction (left-to-right and right-to-left). Unlike traditional models that only look at a word's surrounding context in a single direction, BERT looks at the words that come before and after it.

- **Transformer:** it applies self-attention mechanism, allowing the model to weight the importance of different words in a sentence when encoding a particular word. It enables the model to capture long-range dependencies.
- **Positional Encoding:** Since transformers do not have any built-in notion of word order, positional encodings are added to the input embeddings to provide information about the position of each word in the sequence.
- **Encoder:** it processes the input text and generates a contextualized representation. The original transformer model consists of an encoder and a decoder, BERT only use the encoder, its architecture involves stacking multiple transformer encoders, which allows it to build rich contextual representations of text. Each encoder layer consists of:

    - Multi-head self-attention mechanism: it enables the model to focus on different parts of the sentence simultaneously, capturing various aspects of the relationships between words.
    - Feed-Forward Neural Networks (FFNN): it applies to each position separately and identically. FFNN layers introduces non-linearities and transforms the feature extracted by the self-attention mechanism, increasing the model's capacity to capture intricate relationships in the data.

- Add and normalize operations for layer normalization and residual connections: it stabilizes training and allows gradients to flow through the network more effectively.

If receiving a batch of string that are too large, the BERT tokenizer cannot process at once. So we first break the large dataset into small batch with size 32. The final embedding vector for each tweet has 768 dimensions, which aligns with the expectation of BERT base model.

7. Model Building and Training

We build both traditional machine learning models and deep learning models to do the classification task. Based on the capacity of the models to utilize features effectively, we will use different feature extraction techniques.

For traditional machine learning models:

- It is suitable to use Bag of Words (BoW) and TF-IDF techniques. They produce sparse, high-dimensional vectors, focus on word frequency but do not capture semantic meaning or context. Traditional machine learning models can handle high-dimensional, sparse data effectively.
- Traditional machine learning models work well with linear relationships between features, which BoW and TF-IDF often provide.
- These models are usually computationally less intensive, making them suitable for simpler algorithms that can handle large feature spaces.

For deep learning models:

- It is suitable to use Word Embeddings and BERT techniques. They generate dense, low-dimension vectors where each word is represented by a vector that captures semantic relationships and context.
- Deep learning models can capture and learn from the semantic and contextual information encoded in the embeddings.
- Deep learning architectures can handle the non-linear, intricate relationships and interactions between features that embeddings provide. The transformer-based models are designed to handle and benefit from the rich, contextual information that BERT provides.

Based on the different nature and complexity of features, we decide to use TF-IDF features for traditional machine learning models, and Word Embeddings features for deep learning models.

7.1 Random Forest & XGBClassifier

These two models are traditional machine learning models. Both are tree-based method, the random forest using bagging, while XGBClassifier using boosting.

Random Forest model creates multiple decision trees using different subsets of the data (bootstrapping technique, sampling with replacement) and takes a majority vote (for classification task). At each split in a tree, a random subset of features is selected to find the best split, which introduces randomness and reduces overfitting. We add parameter `n-estimators = 100` when constructing the model. It means that the random forest will consist of 100 individual decision trees.

The XGBClassifier is part of the XGBoost (Extreme Gradient Boosting) library, which is gradient-boosted decision trees. This model sequentially builds trees where each tree tries to correct the errors of the previous trees. It optimizes an objective function (often a combination of a loss function and a regularization term) using gradient descent. Different from random forest, all features are considered at each split, but the algorithm focuses on features that reduce the loss the most. We also add parameter `n-estimators = 100` here.

The test accuracy is 98.95% for random forest model and 90.16% for XGBClassifier.

7.2 LSTM (Long Short-Term Memory)

LSTM models is a neural network that can capture and understand the context and dependencies within text sequences like tweets. It is introduced to address the vanishing gradient problem of RNNs (Recurrent Neural Network) by incorporating a more complex memory cell structure.

LSTM networks use a memory cell that can maintain information over long periods of time, allowing them to learn and remember long-term dependencies in data. It uses the gating mechanisms (input gate, forget gate, output gate) that control the flow of information.

Different tweets can have varying lengths. However, LSTM model requires input sequences (sequence length: number of words/tokens in each tweet) to be the same length for batch processing. So we use `pad_sequences` method to ensures all tweets are of uniform length.

`MAX_SEQUENCE_LENGTH` is defined to set a fixed length for all sequences. After removing the stop words, the maximum sequence length of tweets is 23 (tokens/words). So we set `max_sequence_len = 23`. If a sequence is shorter than MAX_SEQUENCE_LENGTH, it will be padded with zeros (`padding='post'` means padding is added at the end of the sequence).

Since I use the word embeddings Word2Vec model from google (GoogleNews-vectors-negative300.bin) to do the feature extraction, each word vector is in shape of 300. Therefore, my LSTM model will receive inputs of shape `(num_samples, 23, 300)`.

The training stops at the 9^{th} epochs, and the highest validation accuracy is 98.25%. The test accuracy is 98.13%

7.3 Transformer-based Model (BERT)

Transformer, especially models like BERT, are the state-of-the-art architectures for many NLP tasks. They use self-attention mechanisms to understand the context of a word based on its surrounding words, making them highly effective for text classification.

First, we do the data split. Since the `validation_split` parameter in `model.fit` function is not supported for TensorFlow datasets created using the `tf.data.Dataset` API. We need to manually split your dataset into training and validation sets before creating the `tf.data.Dataset` objects.

When preparing data for a BERT model, the text undergoes several transformations to convert it into a format suitable for model input. These transformations ensure that the text is tokenized correctly, padded to the required length, and accompanied by attention masks and segment tokens where necessary. This process helps BERT understand and process the input text effectively, leading to better model performance.

- Input IDs: they are the token IDs obtained from the tokenizer, which represent the input text in a format that the BERT model can process.
- Attention masks: they are binary masks indicating which tokens are actual tokens and which are padding tokens, allowing the model to differentiate between real tokens and padding tokens during the attention mechanism. Padding tokens are added to ensure that all input sequences in a batch are of the same length.

The training stops at the 6^{th} epoch, and the highest validation accuracy is 99.04%, the test accuracy is 98.91%.