Tweet Sentiment Extraction Using Deep Learning Techniques and Open AI's Advanced Models

Hari Chandana Kotnani

Introduction:

In today's digital era, where a single tweet can significantly influence public opinion, the ability to discern the sentiment behind these tweets becomes more crucial than ever. In this project, I take on this challenge, harnessing the power of sentiment analysis to categorize tweets into positive, neutral, or negative sentiments and pinpointing the specific words or phrases that encapsulate these emotions. Alongside a blend of traditional and advanced machine learning techniques—including Naïve Bayes, Support Vector Machines (SVM), and neural networks like Long Short-Term Memory (LSTM), Bidirectional LSTM (Bi-LSTM), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN)—I also explored the capabilities of OpenAI's cutting-edge models to enhance the analysis.

This report outlines the comprehensive methodologies I employed, detailing both the array of models used and my approaches to data cleaning. While establishing a foundation with traditional sentiment analysis methods, I progressively integrated more sophisticated models like LSTM, Bi-LSTM, RoBERTa, CNN, and RNN, as well as leveraging OpenAI's advanced AI, to attain a more nuanced and accurate detection of sentiments. This holistic approach enabled me to effectively navigate and interpret the vast and varied landscape of sentiments expressed in the ever-expanding universe of Twitter.

Dataset:

In my sentiment analysis project, I utilized a dataset sourced from Kaggle, comprising a collection of tweets specifically labeled for sentiment analysis. This dataset is divided into two subsets: a training set and a test set.

Training Set:

Structure: Each entry in the training set consists of up to four columns:

- textID: A unique identifier for each tweet.
- text: The actual text of the tweet.
- selected_text: The portion of the tweet text that exemplifies the sentiment.
- sentiment: The assigned sentiment label (positive, negative, or neutral).

Test Set:

Structure: The test set is structured similarly to the training set but lacks the 'selected_text' column. It includes the sentiment label, and the textID and text columns are also present. Notably, the selected_text field in this dataset may be empty.

Preprocessing:

Text Cleaning

A crucial step in my sentiment analysis pipeline is text cleaning, which involves preparing the raw data for analysis.

- Conversion to String: Ensures that each text entry is a string, handling any None values that may exist.
- URL Removal: Strips out any URLs present in the text using regular expressions, as they are typically not relevant to sentiment analysis.
- Punctuation and Number Removal: Eliminates punctuation and numbers, focusing the analysis purely on textual content.
- Case Normalization: Converts all text to lowercase to ensure uniformity, as case differences do not usually affect sentiment.
- Whitespace Management: Trims extra spaces, tabs, and newlines, leaving a clean, consistent text format.

Each of these steps is critical in reducing noise and standardizing the data, making it more suitable for sentiment analysis.

Word Cloud Generation

To visually explore the most prominent words in my dataset, a visual representation that highlights frequently occurring words in the data.

Sentiment Distribution Visualization

Understanding the distribution of sentiments within my dataset is essential for gauging its balance and diversity. I calculated the count of each sentiment category (positive, negative, neutral) and plotted these counts as a bar chart, providing a clear visual of how the sentiments are distributed across the dataset.

Models:

Naive Bayes Classifier:

I harnessed the power of the Naive Bayes classifier, a probabilistic model renowned for its efficacy in text classification. This model calculates the likelihood of a tweet belonging to a certain sentiment class (positive, negative, or neutral) based on the frequency of words it contains. The underlying assumption of Naive Bayes is that each word contributes independently to the sentiment, which simplifies the computational complexity while maintaining effective performance.

I employed the Multinomial Naive Bayes variant, particularly apt for handling frequency-based features, such as word counts in text data. After processing the tweets into a suitable numerical format, I trained the Naive Bayes model to discern the sentiment of each tweet. The model's predictions were then evaluated against the actual sentiments of the tweets in my test dataset, demonstrating its robustness in classifying sentiments accurately.

Support Vector Machine (SVM) Classifier:

I employed the Support Vector Machine (SVM) classifier, a robust and versatile machine learning model widely used in classification tasks, including text classification like sentiment analysis. The strength of SVM lies in its ability to find the optimal hyperplane that best separates data points of different classes. In this context, it involves distinguishing tweets with different sentiments (positive, negative, or neutral) based on their textual features.

For this purpose, I used the linear kernel in my SVM model, which is particularly effective for high-dimensional data such as text. I utilized the TF-IDF vectorization method to convert tweet texts into

numerical vectors, representing the significance of words in relation to their frequency in the tweet and across the entire dataset. After transforming the text data and encoding the sentiment labels, I trained the SVM model on this data. Its performance was then evaluated on the test dataset, where the model predicted the sentiments of new tweets. The accuracy metric was used to assess how well the SVM model performed in correctly categorizing the sentiments of the tweets, demonstrating its effectiveness in handling complex, high-dimensional text data in sentiment analysis.

Long Short-Term Memory (LSTM) :

I leveraged a Long Short-Term Memory (LSTM) network, a type of recurrent neural network (RNN) particularly effective in processing sequences of data, like text. LSTMs are adept at capturing long-term dependencies and patterns in sequential data, making them a powerful tool for tasks like sentiment analysis where the context and order of words are crucial.

The initial step in using LSTM involved tokenization and vectorization of the tweet texts. Tokenization is the process of converting text into a sequence of tokens (words, in this case), while vectorization involves converting these tokens into numerical sequences, creating a uniform structure that can be processed by the neural network. To ensure consistent input size, I applied padding to these sequences, standardizing their length. The LSTM model was then built using an embedding layer to capture word relationships, followed by an LSTM layer to process the sequential data. This model structure was designed to capture the nuances in the tweets and learn the underlying patterns associated with different sentiments.

To train the LSTM network, the sentiments were converted into one-hot encoded labels, providing a clear target for the model to predict. I used an EarlyStopping callback during training to prevent overfitting, allowing the model to stop training when it no longer sees improvements. Post-training, the model's performance was evaluated on the test dataset, where it predicted the sentiment of new tweets. The accuracy of these predictions was assessed to gauge the LSTM's effectiveness.

Bidirectional Long Short-Term Memory (Bi-LSTM):

I employed a Bidirectional Long Short-Term Memory (Bi-LSTM) network, an advanced variant of the traditional LSTM model. The Bi-LSTM network is particularly effective in understanding context in text data, as it processes sequences in both forward and backward directions, capturing patterns that might be missed by unidirectional models.

The implementation of the Bi-LSTM model followed a similar process to that of the LSTM, involving tokenization and vectorization of the tweet texts. This process converted the textual data into a sequence of numerical values, suitable for neural network processing. The sequences were then padded to ensure uniform length. In constructing the Bi-LSTM model, I utilized an embedding layer for learning word relationships, followed by a bidirectional layer wrapping around the LSTM units. This structure allowed the model to capture intricate dependencies in the data from both past and future contexts.

Training the Bi-LSTM involved converting sentiment labels into one-hot encoded formats, providing a clear, categorical target for the model. To optimize training and prevent overfitting, I implemented an EarlyStopping callback, enabling the model to cease training when no further improvements were observed. Upon training completion, the model's ability to classify sentiments was assessed on the test dataset.

Convolutional Neural Network (CNN):
In my sentiment analysis framework, I integrated a Convolutional Neural Network (CNN), a model typically renowned for its performance in image processing but increasingly popular in natural language processing tasks. The key advantage of CNNs in text analysis lies in their ability to extract local and positionally invariant features, making them efficient for identifying patterns in sentences, such as key phrases indicative of sentiment.

The process began with the tokenization and vectorization of the tweet texts, converting them into a sequence of numbers. These sequences were then padded to a uniform length to ensure consistency in input size. I constructed the CNN model with an initial embedding layer, designed to capture semantic relationships between words. Following this, I added a convolutional layer, which applies filters to the input data to extract important features – in this case, patterns in the text that are relevant for determining sentiment. This was accompanied by a pooling layer to reduce dimensionality and a dense layer to aid in classification.

The sentiments were converted into one-hot encoded labels for training the model, with an EarlyStopping callback implemented to prevent overfitting. This setup allowed the CNN to efficiently learn from the training data and generalize well to unseen data. Upon training completion, the model's performance was evaluated on the test dataset.

Recurrent Neural Network (RNN):
I implemented a Recurrent Neural Network (RNN), a type of neural network that is specifically designed to handle sequential data, making it highly suitable for text processing tasks. RNNs are known for their ability to retain information from previous inputs in the sequence, allowing them to maintain a form of 'memory' about earlier parts of the text. This feature is crucial for understanding the context and flow in language.

The initial steps in employing the RNN involved tokenizing and vectorizing the tweet texts. This process involved converting the text into sequences of numerical values, reflecting the order and occurrence of words. These sequences were then padded to a standardized length to ensure a consistent input size for the neural network. The RNN model I developed included an embedding layer to capture the semantic relationships between words. It was followed by a SimpleRNN layer, the core of the model, which processed the text sequences, retaining information across the sequence to understand the context and nuances better.

To train the RNN, I converted the sentiment labels into one-hot encoded formats and used an EarlyStopping callback to halt training once the model ceased to show improvement, preventing overfitting. This approach allowed the RNN to learn effectively from the training data and adapt to new, unseen data. I evaluated the model's performance on a test dataset, with the accuracy metric reflecting how well the RNN could classify the sentiments of tweets.

RoBERTa:
I incorporated RoBERTa (Robustly Optimized BERT Approach), an advanced language processing model based on BERT (Bidirectional Encoder Representations from Transformers). RoBERTa is recognized for its effectiveness in comprehending the context and subtleties of language, making it exceptionally suited for complex tasks like sentiment analysis.

I chose to use the 'distilroberta-base' model, a streamlined version of the original RoBERTa, which maintains a high level of performance while being more computationally efficient. This was a strategic decision, considering the regular RoBERTa model requires substantial computational resources and time. The process began with transforming the sentiment labels in the dataset into a numerical format using a label encoder, preparing them for model processing. This step involved using RoBERTa's tokenizer to convert tweets into a structured format comprising input ids and attention masks, crucial for the model's understanding and focus on key parts of the text.
For training, I set specific hyperparameters like batch size and maximum sequence length, aiming to strike a balance between efficiency and model effectiveness. The training and evaluation of the model were managed using the Trainer class from the Hugging Face library, involving both training and evaluation datasets. Post-training, I evaluated the model's performance on the test dataset, where it predicted sentiments for new tweets.

The accuracy achieved by RoBERTa in this project highlighted its advanced capabilities in sentiment analysis, especially in processing complex and context-rich textual data. Opting for 'distilroberta-base' proved to be a practical choice, offering an optimal mix of computational efficiency and the sophisticated language understanding characteristic of transformer-based models.

Results:
I experimented with various models ranging from basic to cutting-edge machine learning techniques. The results are as follows:

- **Bi-LSTM (Bidirectional Long Short-Term Memory):** This model achieved an accuracy of **70.4%**. It performed relatively well across all sentiment classes, showing a balanced precision and recall, with a slight edge in detecting neutral sentiments.
- **CNN (Convolutional Neural Network):** The CNN model outperformed Bi-LSTM slightly, with an accuracy of **72.1%**. It demonstrated high precision in identifying positive and neutral sentiments, but a slightly lower recall in positive sentiments.
- **LSTM (Long Short-Term Memory):** The LSTM model had an accuracy of **69.7%**, with its strength lying in the detection of neutral sentiments. However, it showed lower precision in negative sentiments.
- **Multinomial Naive Bayes:** As a basic model, it achieved an accuracy of **65.1%**, with a strong recall in neutral sentiments but lower precision in positive sentiments.
- **RNN (Recurrent Neural Network):** The RNN model had the lowest accuracy at **52.0%**, indicating a struggle in effectively categorizing sentiments, especially in the positive class.

- **SVM (Support Vector Machine):** The SVM model showed a competitive performance with an accuracy of **71.1%**. It was particularly effective in identifying neutral sentiments with high precision and recall.
- **RoBERTa (A variant of BERT):** This cutting-edge model achieved the highest accuracy at **78.0%**. It showed high precision and recall across all sentiment classes, particularly excelling in identifying negative sentiments.

Exploring OpenAI's GPT for Sentiment Analysis

In my project, I explored the capabilities of OpenAI's GPT models for sentiment analysis. To achieve this, I utilized OpenAI's API, which provides access to powerful language models capable of understanding and processing natural language with a high degree of sophistication.

After obtaining an API key from OpenAI, I set out to apply this technology to the sentiment analysis of tweets. My approach involved selecting a subset of tweets from my test data and preparing them for analysis by the GPT model. I used the OpenAI API to create prompts that instructed the model to classify the sentiment of these tweets. This task was facilitated by the OpenAI ChatCompletion function, which I used to process the tweets through the model.

I formatted my prompt to clearly ask the GPT model to classify the sentiment of each tweet, numbering them for easy identification. After sending these tweets to the model, I received responses categorizing each tweet into sentiments such as 'positive', 'negative', or 'neutral'. These results were then meticulously extracted and processed to match them with the corresponding sentiments from my test dataset.

The final step involved comparing the sentiments predicted by the GPT model with the actual sentiments labeled in my test data. This comparison allowed me to calculate the accuracy of the GPT model's sentiment analysis, providing valuable insights into its effectiveness and reliability in this specific application. The experiment offered a glimpse into the potential of leveraging advanced AI models like OpenAI's GPT for complex natural language processing tasks such as sentiment analysis in social media contexts.

OpenAI's GPT for Sentiment Analysis Results:

In evaluating the sentiment analysis capabilities of OpenAI's GPT model, I conducted tests on two sets of tweets from my dataset. The results demonstrated the model's impressive proficiency in accurately interpreting and classifying sentiments in textual data.

- **Accuracy for 10 Tweets**: For a smaller sample of 10 tweets, the GPT model achieved a remarkable accuracy of 100.00%. This perfect score indicates that the model was able to correctly identify the sentiment of each tweet in this subset, showcasing its effectiveness in understanding and analyzing short texts.
- **Accuracy for 100 Tweets**: When the sample size was increased to 20 or more tweets, the GPT model maintained a high level of accuracy, achieving 81.25%. This result underscores the model's robustness and its ability to scale up to larger datasets while still providing reliable sentiment classifications.

It's important to note that I was unable to test the model on more than 20 tweets due to limitations in the API's response capabilities. Despite this constraint, the results from these tests offer promising insights into the potential of using OpenAI's GPT for efficient and accurate sentiment analysis in larger-scale applications.

Conclusion:

Comparing the results besides OpenAI's GPT, RoBERTa emerged as the most effective model in our sentiment analysis task, showcasing the strength of advanced transformer-based models in handling complex language tasks. Among the basic models, SVM demonstrated robust performance, surpassing traditional models like Naive Bayes and RNN. In the realm of neural networks, CNN slightly edged out LSTM and Bi-LSTM, indicating its efficiency in capturing contextual information in text data.

.