Customer Sentiment Analysis

A Graduation Project submitted in fulfilment of the requirements for the internship of Digital Egypt Pioneers Initiative DEPI

by

Abdelhalim Ashraf Abdelhalim

Ahmed Hassan

Mina Gamal Attia

Mohammed Hisham Ahmed

Riham Hussain

October 8, 2024

Contents

Chapter 1

Introduction

1.1 Introduction to Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a technique used in natural language processing (NLP) to determine the emotional tone behind a body of text. It is primarily used to identify and classify opinions or sentiments as positive, negative, or neutral. By analyzing text data, sentiment analysis helps uncover the subjective opinions of individuals, which can be critical for decision-making in businesses and organizations.

At its core, sentiment analysis is about understanding how people feel about a particular subject, product, or service based on textual information like social media posts, reviews, emails, or surveys.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

1.2 Benefits of Sentiment Analysis Across Applications

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus

vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

1.2.1 Customer Feedback Analysis

Companies use sentiment analysis to gain insights from customer reviews, feedback, and surveys. By understanding how customers feel about products or services, businesses can make data-driven decisions to improve their offerings, increase customer satisfaction, and maintain a positive brand image.

1.2.2 Social Media Monitoring

With the rise of social media platforms like Twitter, Facebook, and Instagram, organizations can track public sentiment in real-time. This allows them to monitor their online reputation, understand public opinion on certain issues, and respond quickly to crises or trends.

1.2.3 Market Research

Businesses can analyze online discussions, blogs, and product reviews to understand customer preferences, emerging trends, and competitors' strengths and weaknesses. Sentiment analysis provides valuable insights for product development, marketing strategies, and staying ahead in the market.

1.2.4 Political Campaigns

Sentiment analysis is widely used in politics to gauge public opinion on policies, candidates, and election campaigns. By analyzing public discussions on news platforms and social media, political analysts can predict election outcomes, assess the effectiveness of campaigns, and tailor their messaging accordingly.

1.2.5 Customer Service

Sentiment analysis can be applied to customer support interactions to assess the quality of service and identify issues quickly. Companies can use it to prioritize responses, detect dissatisfaction early, and improve the overall customer experience.

1.2.6 Financial Markets

In finance, sentiment analysis is used to predict market trends and make investment decisions. By analyzing news articles, social media posts, and financial reports, investors can understand market sentiment and predict how it might impact stock prices or the overall market.

1.2.7 Healthcare

n healthcare, sentiment analysis can be used to analyze patient feedback, online reviews of medical services, or public opinion on healthcare policies. It can also aid in identifying mental health issues through the analysis of texts or social media posts that contain emotional cues.

Overall, sentiment analysis offers businesses, governments, and organizations the ability to transform unstructured text data into actionable insights. By leveraging sentiment analysis, organizations can make better-informed decisions, improve customer satisfaction, and stay competitive in their respective fields.

1.3 Problem Statement

In today's digital world, social media platforms like Twitter have become a central hub for people to express their opinions and emotions on various topics, ranging from personal experiences to public events. However, with the massive amount of data generated daily, it is challenging for organizations, businesses, and researchers to manually analyze and understand the sentiment behind these tweets.

The problem this project addresses is the automatic classification of sentiments in Twitter data. Specifically, we aim to determine whether a given tweet expresses a positive or negative sentiment. This classification is essential for understanding public opinion in real time, enabling organizations to respond quickly to customer feedback, market trends, or public reactions to events.

The goal of this project is to develop a machine learning model that can accurately classify tweets as either positive or negative, based on their content. By applying natural language processing (NLP) techniques, the model will transform raw text into a structured format, which will then be analyzed to predict the underlying sentiment.

Chapter 2

Dataset

2.1 Data Description

The dataset used for this sentiment analysis project consists of 1,600,000 tweets extracted using the Twitter API. Each tweet in the dataset has been annotated with a sentiment label, indicating whether the sentiment is positive or negative.

- Size of the dataset: 1,600,000 tweets
- Source: Extracted from Twitter using the Twitter API
- Sentiment labels:
  - Positive: Tweets expressing a positive sentiment.
  - Negative: Tweets expressing a negative sentiment.

2.2 Data Preparation

In this section, we will explain how to clean and preprocess the data step by step.

2.2.1 Data Cleaning Steps

1. Lowercasing: We convert all text to lowercase to maintain consistency. For example, ”Happy” and ”happy” should be treated the same.

2. 2Remove URLs: Tweets often contain links, which do not usually carry sentiment information.

3. Remove Mentions and Hashtags: While hashtags can be useful for some tasks, they may introduce noise in sentiment analysis unless you analyze their content. Mentions (@username) can also be removed.

4. Remove Special Characters and Punctuation: Remove unnecessary punctuation, symbols, and special characters.

5. Remove Numbers: Numbers may not carry much sentiment in most cases, so they can be removed.

6. Remove Stopwords: Stopwords like ”and,” ”the,” and ”is” can be removed, as they don’t contribute much to sentiment.

7. Lemmatization: Reduce words to their base or root forms (”running” $\rightarrow$ ”run”).

8. 8. Remove Repeated Characters: Reduce repeated characters, such as ”sooooo happy” to ”so happy.”

9. Remove duplicated sentences and missing sentences.

2.2.2 Exploratory Data Analysis (EDA) Steps

1. Word Count Calculation: Determined the count of words in each sentence to assess text length.

2. Word Frequency Analysis: Calculate the frequency of each word to identify common terms.

3. Word Cloud Visualization: Created word clouds to visually represent the most frequent words.

4. Sentence Length Distribution: Plotted the distribution of sentence lengths to understand text structure.

5. Target Column Analysis: Visualized the counts of each value in the target column to evaluate the class distribution.

FIGURE 2.1: Word Frequency

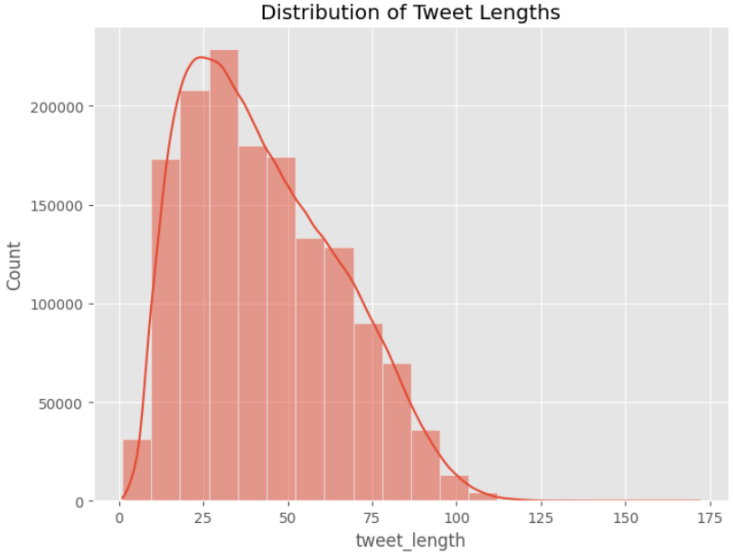

FIGURE 2.2: Distribution of Tweet length

	text	target
1058697	Wow, #ff thx to @MonaLockedIn @RHHR @lyndalipp...	1
43763	Everyones ignoring me tonight sawwy guyszs	0
729119	Wishes her stomach was feeling better	0
130025	assignment.	0
406791	i hatee ayesshaa she suckkss	0

FIGURE 2.3: Input

	text	target	tweet_length
187131	good day faith	0	14
583346	know amp	0	8
1547243	true enjoy qi night	1	19
1240292	dj born runner think ran walked gene	1	36
200870	want see th jonas brother	0	25
1509766	hey lovebugjonas brother la la landdemi lovato	1	46
1391588	cheer meet hotel vin champagne	1	30
387627	come back philippine chance watch	0	33
370326	pussycat doll guess	0	19
927818	realized people think twitter pointless must r...	1	91
449466	realizes list neyo make think couple ex good time	0	49
523020	sleepy waking work ahh	0	22
994315	good morning hope everyone quotfunquot monday lol	1	49
707999	last hoofin class	0	17
778966	yeah especially go back maybe meet harry p nig...	0	60
1195488	wordcamp sf fun john lily definitely awesome s...	1	99
1170884	enough prawn eat flavour never mind drink good	1	46
771522	wanted go see transformer tonight avail	0	39
1168074	feeling like school girl crush	1	30
377525	meas eight plus give add confused	0	33

FIGURE 2.4: Output Data After Data Cleaning and Preprocessing :

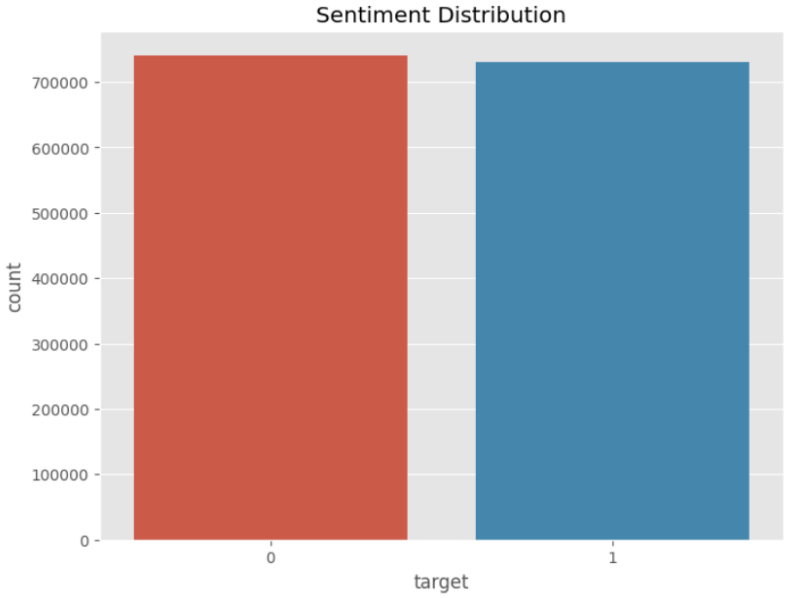

FIGURE 2.5: Sentiment Distribution

Chapter 3

Machine Learning Models

3.1 Logistic Regression

Our Jupyter notebook performs sentiment analysis on a Twitter sentiment dataset. Here's a breakdown of the steps involved:

1. Library Installation: Installs required libraries like `streamlit`, `nltk`, `contractions`, `wordcloud`, and `emoji` using `!pip`.
2. Import Libraries:
    - Imports necessary libraries for text processing, data manipulation, machine learning, and visualization.
    - Suppresses warnings using `warnings.filterwarnings("ignore")`.
    - Downloads NLTK resources for stopwords, tokenization, and wordnet.
3. Read the Dataset:
    - Reads the Twitter sentiment dataset from a GitHub URL using pandas.
    - Renames columns to sentiment and text.
4. Data Cleaning and Preprocessing:
    - Drops duplicates and missing values.
    - Converts text to lowercase.

- Removes URLs, mentions, hashtags, special characters, and numbers using regular expressions.
- Expands contractions using `contractions.fix`.
- Removes stopwords from English using NLTK stopwords.
- Performs lemmatization using `WordNetLemmatizer` for consistent word forms.
- Handles emojis by converting them to their textual representation using `emoji.demojize`.
- Removes repeated characters using regular expressions.
- Filters out empty tweets based on length.

5. Tokenization: Downloads and utilizes NLTK's `punkt` tokenizer to split text into individual words.
6. Save Cleaned Data: Saves the cleaned data frame as a CSV file named `clean_data.csv`.
7. Model Training:
   - Splits the data into training and testing sets using `train_test_split`.
   - Creates a pipeline with `TF-IDF` vectorizer and `Logistic Regression` model.
   - Trains the model on the training data.
8. Model Evaluation:
   - Predicts sentiment on the testing data using the trained model.
   - Prints a classification report showing model performance metrics like accuracy, precision, recall, and F1 score for positive and negative sentiments.
9. Confusion Matrix:
   - Creates a confusion matrix to visualize the model's prediction accuracy for each sentiment class.
   - Uses `Seaborn` heatmap for visualization.
10. Save the Model: Saves the trained model as a pickle file named `logistic_twitter_sentiment.pkl`.
11. Test the Model with New Examples: Predicts sentiment on new user-provided text samples.

3.2 Random Forest Documentation

This Jupyter notebook performs sentiment analysis on Twitter data. It follows these steps:

1. Install Libraries: Installs required libraries like `streamlit`, `nltk`, `contractions`, `wordcloud`, and `emoji`.
2. Import Libraries: Imports necessary libraries for data processing and visualization.
3. Load Data: Loads the CSV dataset from a URL and selects relevant columns.
4. Data Cleaning and Preprocessing:
    - Removes duplicates and missing values.
    - Converts text to lowercase.
    - Removes URLs, mentions, hashtags, special characters, and numbers using regular expressions.
    - Expands contractions.
    - Removes stopwords using NLTK.
    - Performs lemmatization using `WordNetLemmatizer`.
    - Handles emojis with the `emoji` library.
    - Removes repeated characters.
    - Tokenization using a regular expression tokenizer.
5. Add Tweet Length and Remove Empty Tweets: Adds a column for tweet length and removes tweets with zero length.
6. Exploratory Data Analysis (EDA):
    - Defines functions to:
        - Show word frequency.
        - Plot word cloud.
        - Plot tweet length distribution.
        - Plot sentiment distribution.

- Visualizes the EDA results using these functions.

7. Split Data: Splits data into training and testing sets using `train_test_split`.
8. Build a Random Forest Model Pipeline:
   - Uses `TfidfVectorizer` to convert text to TF-IDF vectors.
   - Implements a Random Forest Classifier with hyperparameter tuning.
   - Uses a pipeline to combine these steps.
   - Trains the model on the training data.
9. Evaluate the Model:
   - Uses `classification_report` and `confusion_matrix` to evaluate the model's performance.
10. Save the Model: Saves the trained model using `pickle`.
11. Make Predictions on New Tweets: Uses the trained model to predict sentiment for new tweet examples.

3.3 Fine-tuning with Hugging Face Model

Step-by-Step Breakdown and Pseudocode:

1. Install Libraries (Pseudocode):
   - Install required libraries like `streamlit`, `nltk`, `contractions`, `wordcloud`, and `emoji`.
2. Import Libraries (Pseudocode):
   - Import necessary libraries like `pandas`, `re`, `emoji`, `stopwords`, `WordNetLemmatizer`, `word_tokenize`, `WordCloud`, `Counter`, `matplotlib.pyplot`, `seaborn`, `contractions`, `warnings`, `sklearn.metrics`, `torch`, `transformers`, etc.
3. Download NLTK Resources (Pseudocode):

- Download resources like stopwords and punkt data if not already downloaded.

4. Read Dataset (Pseudocode):

   - Read the Twitter sentiment CSV dataset from a URL.
   - Select relevant columns (sentiment and text) and rename them.
   - Clean the data by removing duplicates, missing values, URLs, mentions, hashtags, special characters, numbers, and applying text transformations (lowercase conversion, emoticon handling, stopword removal, lemmatization).
   - Add a new column for tweet length.

5. Exploratory Data Analysis (Pseudocode):

   - Perform descriptive statistics on the data.
   - Analyze word frequency distribution.
   - Visualize word cloud to see prominent words.
   - Plot tweet length distribution.
   - Plot sentiment distribution.

6. Tokenization and Data Preparation (Pseudocode):

   - Drop irrelevant and neutral tweets as the model focuses on positive and negative sentiment.
   - Split the data into training and testing sets.
   - Define a function to read data and split it into texts and labels.
   - Tokenize the training and testing texts using a DistilBERT tokenizer.
   - Create separate datasets (training and testing) for the model.

7. Model Training (Pseudocode):

   - Set up training arguments like output directory, number of epochs, batch size, warmup steps, weight decay, logging directory and frequency, learning rate, etc.
   - Load the pre-trained DistilBERT model for sequence classification.

- Initialize the `Trainer` class with the model, training arguments, training dataset, and evaluation dataset.
- Train the model.

8. Model Evaluation (Pseudocode):
   - Evaluate the model's performance on the test dataset using metrics like accuracy, precision, recall, F1 score, etc.
   - Calculate and display the confusion matrix to visualize model predictions.
9. Model Saving and Prediction (Pseudocode):
   - Save the trained model's state dictionary as a pickle file.
   - Define a function to load the saved model and predict sentiment for new text inputs.
   - Use the prediction function to make sentiment predictions on new tweets.
   - Save the improved model by saving only the state dictionary using `torch.save`.

3.4 Neural Network Model Implementation From Scratch

I have implemented the Neural Network Model from scratch using only Numpy Library. Here's the break down of my algorithm step by step:

3.4.1 Initializing Random Weights

The function `initialRandomParams()` initializes random weights and biases for the neural network. The weights are for three layers (input, hidden, and output), and the values are generated randomly between -0.5 and 0.5.

3.4.2 Activation Functions

- **Threshold of Zero Activation:** The function `ThresholdofZero(Z)` serves as a ReLU activation function, returning 0 for negative values and the value itself for positive values.

- **Softmax Activation:** The `softmax(Z)` function normalizes the output vector into a probability distribution, ensuring the sum of the outputs equals 1. This is essential for multi-class classification.

3.4.3 Forward Propagation

The function `forward_prop()` performs forward propagation by calculating the output of each layer sequentially:

- **First hidden layer:** Uses weights $W1$ and biases $b1$.
- **Second hidden layer:** Uses weights $W2$ and biases $b2$.
- **Output layer:** Uses weights $W3$ and biases $b3$.

The result is passed through activation functions at each step.

3.4.4 Backward Propagation

The function `backward_prop()` computes gradients for weights and biases through backpropagation. This adjusts the weights and biases to reduce the error in future iterations.

3.4.5 Updating Weights

The function `Update_Weights()` updates the weights and biases after each iteration of gradient descent to minimize the error.

3.4.6 Predictions

The function `get_predictions()` returns the index of the highest value in the output vector, representing the predicted class.

3.4.7 Accuracy Calculation

The function `get_accuracy()` calculates the model's accuracy by comparing predictions with actual labels.

3.4.8 Gradient Descent Algorithm

The function `gradient_descent()` combines forward propagation, backward propagation, and weight updates to iteratively minimize the error. The process runs for a specified number of iterations.

3.4.9 Dataset Loading and Preparation

The dataset is split into training and development sets.

- **Tokenization:** Applied to convert text data into numeric form using `Tokenizer()`.
- **Padding:** Applied to ensure that all sequences have the same length. I have implemented the padded algorithm to pad using the data itself rather than padding with zeroes to avoid data unbalancing.

3.4.10 Training Process

The neural network model is trained on the padded data using gradient descent. The weights are updated after each iteration to improve accuracy. I have tried multiple values for hyper-parameters until I found satisfying results.

3.4.11 Model Saving and Loading

After training, the weights are saved as `.npy` files for future use. The model can be loaded later for testing or further training.

3.4.12 Testing Process

- **Predictions:** The function `Predictions()` performs forward propagation on test data to predict sentiment labels.
- **Evaluation:** The function `test()` compares the predictions with actual labels for evaluation.

3.4.13 Results

The model predictions are converted back into text form using the reverse tokenizer. Accuracy is calculated, and sample predictions are displayed for specific indices.

The Final Result:

- Final Training Accuracy = 96.6667 percent.
- Final Testing Accuracy = 95.357 percent.

Chapter 4

LSTM

4.1 LSTM Model Version 1

In this project, I implemented an LSTM (Long Short-Term Memory) model to classify sentiments from Twitter data.

The architecture of the LSTM model is as follows:

- Input Layer: The input layer takes in sequences of tokens, with a defined maximum length.
- Embedding Layer: This layer converts tokens into dense vectors of fixed size, which capture semantic relationships between words.
- LSTM Layer: With 64 units, this layer processes the embedded word vectors sequentially, capturing context and dependencies between words.
- Fully Connected Layer: A dense layer with 256 units followed by a ReLU activation, which helps the model learn complex patterns.
- Dropout Layer: A dropout of 50 percent is applied to prevent overfitting.
- Output Layer: The final dense layer with a sigmoid activation outputs a single value, representing the probability of the sentiment being positive.

The model was compiled with the binary cross-entropy loss function, RMSprop optimizer, and accuracy as the evaluation metric. Early stopping was also implemented to prevent overfitting by stopping training when the validation loss did not improve for three consecutive epochs.

4.2 LSTM Model Version 2

I implemented an LSTM (Long Short-Term Memory) model to classify sentiment from Twitter data and used 100,000 samples of data to train the model in this project.

1. Model Architecture:

   The LSTM model leverages a sequence of layers to process and classify the sentiment of text data. Below is the breakdown of each component:

   - **Embedding Layer**:
     - **Purpose**: Converts input words (represented as integers) into dense vectors of fixed size. These dense vectors capture the semantic meaning of the words.
     - **Input Shape**: Varies depending on the input tweet size (post-tokenization and padding).
   - **First LSTM Layer**:
     - **Units**: 100
     - **Dropout**: 20% (to avoid overfitting).
     - **Return Sequences**: True (allows the next LSTM layer to receive the full sequence output).
     - **Purpose**: Processes the sequential data, allowing the model to understand long-term dependencies between words in a tweet.
   - **Second LSTM Layer**:
     - **Units**: 50
     - **Purpose**: Further processes the sequence after the first LSTM layer, enhancing the model's ability to capture nuanced patterns in the data.
   - **Dense (Fully Connected) Layer**:

  - **Units**: 32
  - **Activation**: ReLU (introduces non-linearity into the model).
  - **Purpose**: Fully connected layer that processes the output from the LSTM layers to create high-level representations of the sequence data.
- **Dropout Layer**:
  - **Rate**: 50%
  - **Purpose**: Drops 50% of the neurons to prevent the model from overfitting and improve generalization.
- **Output Layer**:
  - **Units**: 1
  - **Activation**: Sigmoid (used for binary classification tasks).
  - **Purpose**: Outputs the final prediction—a probability value representing the sentiment as positive or negative.

2. Model Compilation:

   The model is compiled with the following settings:

- **Loss Function**:
  - **Binary Cross-Entropy**: This function is ideal for binary classification problems like sentiment analysis. It measures the distance between the predicted probability and the actual binary label (0 for negative, 1 for positive).
- **Optimizer**:
  - **Adam**: A widely used optimizer that adapts the learning rate during training to ensure smooth convergence. The learning rate is set to 0.001.
- **Evaluation Metric**:
  - **Accuracy**: The model tracks the accuracy of predictions on the validation set, ensuring that it correctly classifies as many tweets as possible into their respective sentiment categories.

4.3 BERT Text Classification Model Summary

This project implements a text classification model using BERT, designed to classify texts into two categories. The workflow includes data preparation, model definition, and training:

1. Data Preparation: The ‘TextClassificationDataset‘ class tokenizes and processes the input text data, converting it into token IDs and attention masks required for BERT.
2.
3. Model Architecture: The ‘BERTClassifier‘ utilizes a pre-trained BERT model, followed by a dropout and a fully connected layer to classify inputs.
4. Training Process: The ‘train‘ function executes model training, using CrossEntropy for loss computation, and AdamW for optimization. A learning rate scheduler adjusts the learning rate across epochs.
5. Execution Setup: Key parameters include a maximum token length of 128, a batch size of 16, and a learning rate of 2e-5. The model trains for four epochs with an 80-20 train-validation split.

This setup provides an efficient approach to training a BERT-based text classifier with configurable parameters for fine-tuning and evaluation.

4.4 Sentiment Analysis with BERT using Huggingface

1. Hugging Face Libraries:
   - Hugging Face provides two main libraries: `transformers` for models and `datasets` for datasets. These can be installed using `pip` as usual.
2. Batch Size:
   - Number of samples processed per training iteration.
3. Sentiment Labels:

- 0 (negative) and 1 (positive).

4. BERT Model:
   - `model_id="bert-base-uncased"`: Refers to a specific pre-trained BERT model that has been trained on a large corpus of text. The ”uncased” part indicates that this model is case-insensitive, meaning it treats uppercase and lowercase letters as the same.
   - `tokenizer = BertTokenizerFast.from_pretrained(model_id)`: This line creates a tokenizer object for the specified pre-trained BERT model. The tokenizer object is ready to tokenize text using the rules and vocabulary of the ”bert-large-uncased.”
5. Tokenizer:
   - Tokenizes text into `input_ids`, `token_type_ids`, and `attention_mask`.
6. Padding and CLS/SEP Tokens:
   - The `[CLS]` token marks the beginning of the input sequence, and the `[SEP]` token marks the end of the input sequence.
   - `[PAD]` tokens are added to the end of the sequence to ensure the sequence reaches a specific maximum length. These tokens ensure that all sequences in the batch have the same length.
7. Learning Rate:
   - Set to 2e-5 with no warm-up steps.
8. Overfitting:
   - Train vs. validation loss indicates potential overfitting; use early stopping.
   - Overfitting occurs when the training loss continues to decrease while the validation loss begins to increase.
   - Early stopping should be applied when the validation loss starts to increase, which in this case is just after 0.75 epochs.
9. TensorFlow Dataset:
   - Prefetching and data structuring optimize training.

- The `.prefetch()` method asynchronously fetches batches of data from the dataset while the model is training, reducing data loading times.
- `tf.data.AUTOTUNE` is a constant that indicates TensorFlow should automatically choose the number of batches to prefetch based on available resources and the execution context.

10. Techniques to Improve Performance and Mitigate Overfitting:

    - Adding dropout.
    - Using regularization.
    - Collecting more data.
    - Simplifying the model.