

TWITTER SENTIMENTAL ANALYSIS

By

-Gagandeep

-Pradeep Penumarthy

Table of Contents

INTRODUCTION.....	3
STATEMENT OF PROBLEM.....	4
APPROACH.....	5
DATA COLLECTION.....	5
CLOUD COMPUTING.....	5
AWS EC2 instance:.....	6
Amazon S3.....	6
Pyspark:.....	7
DATA PREPROCESSING.....	7
MODELING.....	8
Bag of Words (BoW):'.....	8
Recurrent Neural Network (RNN):.....	8
Long Short-Term Memory (LSTM):.....	9
BERT (Bidirectional encoder representation from transformers).....	9
RESULTS.....	17
DISCUSSION OF RELATED WORK AND COMPARISON.....	20
LIMITATIONS AND POSSIBLE EXTENSIONS.....	21
CONCLUSION.....	22
REFERENCES.....	23
APPENDICES.....	24
TABLE OF CONTENT.....	24
APPENDIX A: USER'S MANUAL.....	25
APPENDIX B: IMPLEMENTATION MANUAL.....	27
APPENDIX C: SOURCE CODE.....	30

INTRODUCTION

Twitter provides a platform for individuals to express their opinions on various topics to reach a bigger audience. Furthermore, it is a valuable source for taking people's views from different perspectives. Using this platform, we can extract and categorize the data into relevant types. For Instance, if a company launches a new product, we can extract the data from Twitter. Then we use preprocessing technique and categories the data into positive or negative feedback by applying NLP and Machine learning techniques. With these insights, we can make an analysis on which date.

The goal of the project is to provide trends of the sentiments by analyzing the tweets. This was achieved by harnessing the data using NLP and Machine learning techniques. We aim to create a dashboard that has the functionality of evaluating whether the tweeted data is negative or positive. Besides this, it also has a visualization of the graph that illustrates the trends of the positive probability of tweets by date. We also aim to find factors that may affect the user's sentiments. Although we have used a generic dataset to find the trends in the sentiments, if we use a brand-specific or topic-specific dataset then this trend will be more insightful for that particular case

In this report, we divide the project implementation into the different sections. First, addressed the challenge of finding emotions from text data by utilizing cloud computing, data preprocessing, and advanced modeling techniques. Second, the data collection from public platforms . Then Cloud computing, specifically AWS EC2 instances and Amazon S3, which provides the computational power and secure storage needed for efficient processing. Next, Spark aids for data processing within this cloud environment and then Data preprocessing is a pivotal step to ensure accurate analysis. Followed by, exploring various sentiment analysis models, including Bag of Words, RNN, LSTM, and BERT. After that, results that highlight insights and model accuracies and limitations where suggest possible extensions. In conclusion, our paper goes through the implementation of sentiment analysis on Twitter, providing an understanding of methodologies and results.

STATEMENT OF PROBLEM

Social media provide a platform for freedom of speech to the public for good will but some people may take this as an advantage and misuse it by spreading hateful content. To address this issue, we can use sentimental analysis which classifies whether the information posted is toxic or not. However, the information obtained from these platforms is huge and varies. This cannot be handled by a normal system as it is restricted by many components like memory, computational power, etc. In addition to this, the content can't become toxic based on the words used but on the context of the issue. For instance tweets is like "I live to work" are not the same as I work to live" it is important to secure the context of the text for sentimental analysis. Also we aim to find the underlying factors that may affect the user sentiments.

APPROACH

DATA COLLECTION

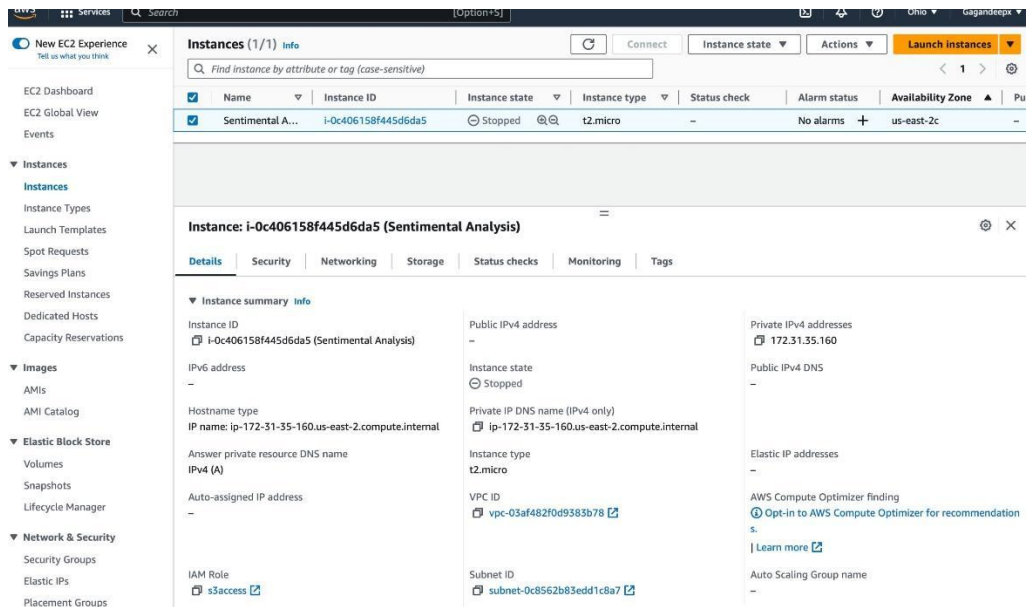
The sentiment140 dataset is taken from Kaggle. The Twitter API search is used to gather the data based on the keyword search. The data set is created using an automated approach with 1,600,000 tweets. All the tweets in the data set are annotated with sentimental labels as 0 for negative, 2 for neutral, and 4 for positive. So this data set can be used for sentimental analysis. This data has six fields:

1. target: sentimental labels as 0 for negative, and 4 for positive
2. ids: it is the primary id for each tweet.
3. date: it time and data when the tweet is posted.
4. flag: The query used to obtain the tweet. If there's no query, the value is NO_QUERY.
5. user: username of the posted tweet.
6. text: it is a tweet posted by the user.

CLOUD COMPUTING

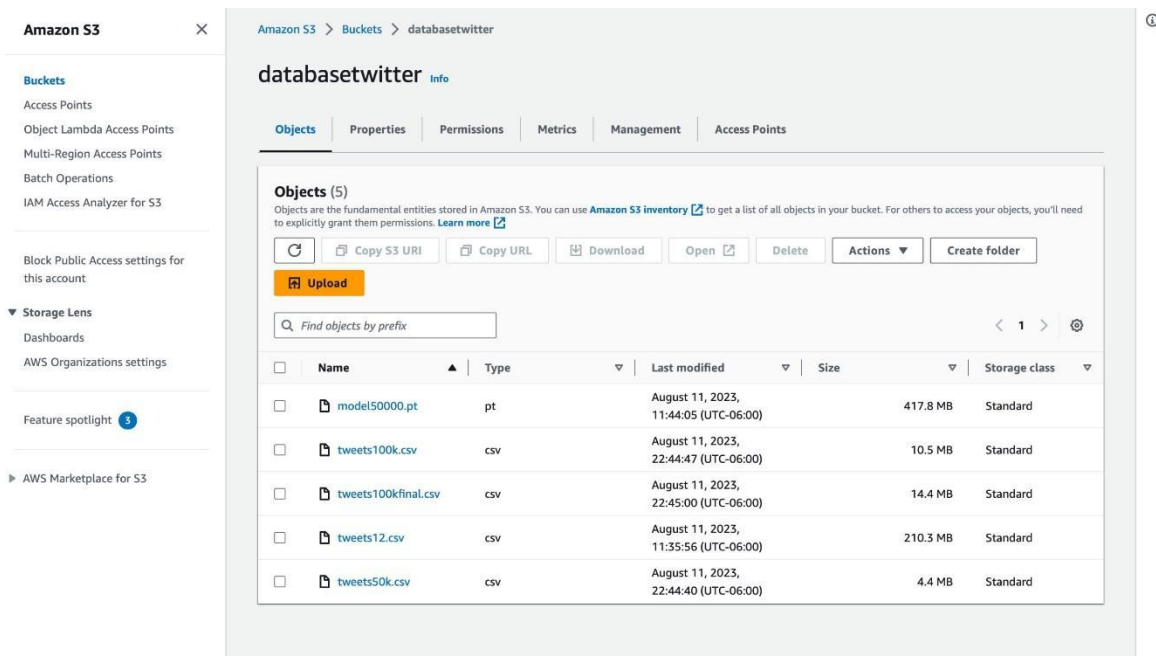
AWS EC2 instance:

The AWS EC2 instance type 't2.micro' belongs to the general-purpose instances by Amazon Web Services (AWS) for cloud computing. Named the T2 Micro, this instance type is designed to provide a balanced mix of compute capacity and cost-efficiency. With 1 virtual CPU and 1.0 GiB of memory. Powered by an Intel Xeon processor operating at a clock speed of 3.3 GHz, this instance type is capable of handling modest computational workloads. The architecture of the CPU is based on i386. In terms of network performance, the 't2.micro' instance type supports IPv6 and can accommodate both IPv4 and IPv6 network configurations. It is important to note that the T2 Micro instance is a cost-effective option.[1]



Amazon S3

The AWS Free Tier provides access to Amazon S3, an advanced object storage infrastructure designed to various purpose of data storage. It has mojour components of security, durability, and scalability, Amazon S3 has a secure connection for storing and managing data of all types. it allows 20,000 GET requests, enabling them to retrieve data effortlessly from their S3 storage. Additionally, it also allows 2,000 PUT requests, provides access to upload new data for storage.[2]



Pyspark:

Pyspark in loading and processing of data: It is an open-source library for big data processing framework and a Python library for Apache Spark. It provides sparks distributing power while using Python code. PySpark offers several benefits when it comes to performing sentiment analysis on Twitter data. Twitter generates a massive amount of data, and PySpark's distributed computing capabilities allow the process of data efficient. Spark's parallelism can help to analyze tweets in parallel, significantly speeding up the analysis process. It provides data transformation functionalities that are essential for cleaning and preprocessing Twitter data. Spark's rich set of functions removes irrelevant information, handle hashtags, mentions, and emoticons, and perform text normalization. Furthermore, it provides a structured and tabular way to handle Twitter data, making it easier to manipulate, filter, and analyze. This is achieved By applying SQL-like queries and Data Frame operations to extract valuable insights from the data.in addition to it integrates with other Spark libraries and tools [9].

DATA PREPROCESSING

In a cluster, the data file is copied to that worker node. This includes determining data types for each column in the CSV based on the content of the first few rows. The inferred schema will influence how the data is organized in the data frame. The data frame combines the data from the CSV file and its inferred schema. it uses lazy evaluation which saves computational power.

```
df = spark.read.csv('tweets12.csv')
```

Sampling: The sampling method is used to randomly select a subset row from the data frame without replacement which means each row will occur only once in the sampled data. Furthermore, the number of rows to be sampled is configurable and these rows are selected without losing the randomness. Seed attributes aid to generates the same set of rows of samples each time when the code is executed. With this technique, the huge data is replicated as a small data with the same statistical properties.

```
n = 50000
df = df.sample(False, fraction=n / df.count(), seed=42)
```

Date formation: Convert the data string to a date time object and extract the day of week, month, day, and year and return the list which contains the day of week and date.

```
# Defining UDF to process the date
def process_date(date_str):
    date_obj = datetime.strptime(date_str, '%a %b %d %H:%M:%S PDT %Y')
    day_of_week = date_obj.strftime('%A')
    month = date_obj.strftime('%B')
    day = date_obj.strftime('%d')
    year = date_obj.strftime('%Y')
    return [day_of_week, f"{month} {day} {year}"]
```

Removing media and tags : A custom function is designed to remove the tags and media like url called tweet_cleaner. We applied this function to all the tweets present in the data set.

```
def tweet_cleaner(x):
    text = re.sub("[@&][A-Za-z0-9_]+", "", x) # Remove mentions
    text = re.sub(r"http\S+", "", text) # Remove media links
    return text
```

Converting text to lowercase: After cleaning the text, we are converting the tweets into lower cases as we are using the Bert base model uncased and to maintain the consistency of words.

```
# Converting all text to lowercase
df = df.withColumn('text', lower(df['text']))
```

Make the tweet into a single line by replacing the line breaks with a empty value.

```
# Removing newline characters
df = df.withColumn('text', regexp_replace(df['text'], '\n', ''))
```

Handling Empty values: Replace the null value when there is no tweet

```
# Replacing empty strings with null values
df = df.withColumn('text', when(df['text'] != '', df['text']))
```

Removing Null values: Drop all the rows with null values in the text column.

```
# Dropping rows with null values in 'text' column
df = df.na.drop(subset=['text'])
```

MODELING

In natural language processing and machine learning, The following algorithms played a critical role in the past for text analysis and sequence modeling. We analyzed them to utilize in our project.

Bag of Words (BoW):

Each word's frequency in the text is counted and used to construct a numerical vector representation. Where the order of the word is disregarded and text is represented as a bag. Where the frequency of words is used for text classification[7].

Disadvantages:

BoW doesn't consider the context or grammar of the words, which can limit its ability to capture meanings in the text. Which leads to low accuracy in the text toxicity classification.

Recurrent Neural Network (RNN):

RNN is a type of neural network which are built to handle sequential data. Contradicting the traditional feedforward network RNN, RNN has a connection from the previous network output

as feedback such that the current text will be processed with the previous output. This aids to preserve the context of the text. This is more useful when the task involves in a sequence of data, where each input depends on the context of previous inputs[7].

Disadvantages:

RNNs suffer from the vanishing gradient problem, leading to difficulties in capturing long-range dependencies that lead to loss of text context.

Long Short-Term Memory (LSTM):

To compensate vanishing gradient problem and capture long-term dependencies in sequences LSTM is designed. Memory cells and different gates are introduced to control the flow of information allowing them to selectively remember or forgot information [7].

Disadvantages:

LSTMs have a more complex architecture , this can make them harder to understand, train, and optimize. They might struggle with dependencies that span very long sequences.

BERT (Bidirectional encoder representation from transformers)

The remarkable feature of BERT is its ability to capture bidirectional context, which allows it to understand the meaning of a word based on the words that come before and after it in a sentence. It's a pre-trained model on large datasets and it is based on the transformer architecture. Where encoders are stacked that captures capture more details in the data So we can use Bert to learn Sentimental analysis [7].

```
# Loading the pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

Embedding Layers: These layers convert input tokens into continuous vector representations (embeddings).

A tokenizer is used to process and convert raw text into a format suitable for input into NLP models like BERT. It breaks down the text into smaller units called tokens, which are the basic elements that the model understands. It is important to convert raw words and sentences into English to vectors which aid to add value and importance to words in the sentence so Twitter data is prepared in a way that the Bert model can process. The tokenizer will convert each tweet into a sequence of token IDs, adding special tokens for padding, separations, and potential classification which will be used as Bert's model during training and evaluation.

This method initializes the tokenizer using a pre-trained model. The 'bert-base-uncased' pre-trained model is being used to initialize the tokenizer. This ensures that the tokenizer is configured to understand and tokenize text according to the BERT model.

Input text: I love using the Twitter to learn new things.

Tokenized Input: {'input_ids': tensor([[101, 1045, 2293, 2478, 1996, 4274, 2000, 4553, 2047, 2472, 2477, 1012, 102, 0, 0, 0, 0, 0, 0, 0]]),

'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]])}

The input_ids tensor represents the token IDs of the tokens in the input text.

101 - [CLS] Beginning of the sequence

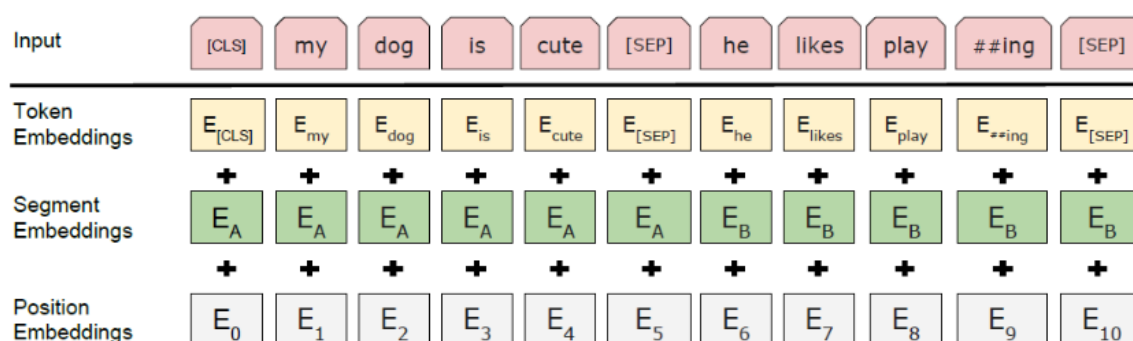
Sentence Classification (e.g., Sentiment Analysis): sentiment analysis, the [CLS] token is used to represent the entire sequence's classification. The final hidden state of the [CLS] token after processing is often used as a pooled representation of the entire sequence. This representation is then fed into a classification layer to make predictions about the sequence.

102 – [SEP] Ending of the sequence.

[UNK] Any token not appearing in its vocabulary is replaced by [UNK] for "unknown"

7 zeros – Padding so each sequence has the same length

The attention_mask tensor indicates which tokens are actual content and which are padding tokens.



BERT input representation

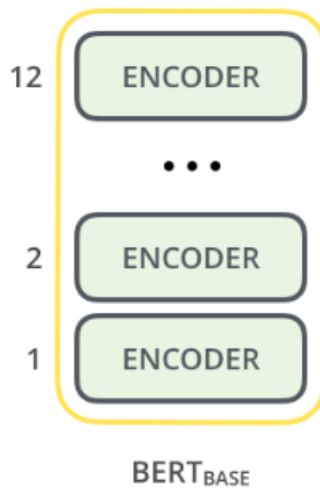
[7]

Token Embeddings: pre-trained embeddings for different words in this model we are using 'bert-base-uncased'

Segment Embeddings: the sentence number that is encoded into a vector.

Position Embeddings: Represents the position of the sentence. If there are two sentences then position 0 represents the first sentence and 1 represents the second sentence. It uses sin and cosine functions.

Bert Base Uncased



[7]

Uncased: It accepts all text as lowercase, removing the difference between uppercase and lowercase letters. This is used for generalization, as it helps the model to recognize words regardless of capitalization.

12-layer: The BERT model architecture consists of a series of transformer layers. Each layer processes the input data in parallel and captures contextual relationships. The "12-layer" specification means that this BERT variant consists of 12 transformer layers.

768-hidden: The "768-hidden" specification indicates the dimensionality of the hidden layers within each transformer block. Higher values generally allow the model to capture more patterns and relationships in the data but also increase computational requirements.

110M parameters: More parameters generally provide the model with a greater capacity to learn complex patterns from data. In this case, the "110M parameters" means that the model has approximately 110 million learnable parameters.

It's "uncased" because, during pre-training, all text was converted to lowercase, removing the need for the model to distinguish between uppercase and lowercase letters. The "cased" variant retains the original capitalization.

Encoders:

The Encoders mainly consist of two layers, they are feedforward neural networks and self-attention.

Self-attention: it is used to weigh the importance of different words in a sequence related to each other, capturing contextual relationships and dependencies between them. It is an important step for finding the context between the words. The encoder allows each word to gather information from other words in the sequence, taking into account their contextual relevance. This mechanism is highly effective for capturing long-range dependencies, understanding the meaning

of words in context, and improving the model's ability to perform tasks like translation, sentiment analysis, and more [3].

The context of the words is achieved by processing the input sequence, self-attention operates as follows:

1. **Input Embeddings:** The input sequence, consisting of token embeddings (word vectors) and positional encodings, is provided to the transformer.
2. **Key, Query, and Value Projections:** For each word in the sequence, the self-attention mechanism creates three kinds of representations: key vectors, query vectors, and value vectors. These vectors are projections of the input embeddings. All these vectors are the same dimensionality of 64.
3. **Dot-Product Attention:** The self-attention mechanism computes a weighted sum of the value vectors using the similarity between the query vector of a word and the key vectors of all other words. The similarity is computed as the dot product between the query and key vectors, which is then scaled using a normalization factor.
4. **Attention Weights:** The normalized dot products are transformed into attention weights through a SoftMax operation, making them sum to 1. These weights indicate how much attention each word should pay to other words in the sequence.
5. **Weighted Sum:** The weighted sum of the value vectors, using the attention weights as coefficients, generates the self-attention output for each word. This output encodes information about the word's relationships with other words in the sequence, capturing its contextual significance.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad [3]$$

6. **Multi-Head Attention:** BERT, multiple self-attention "heads" are employed. Each head learns different aspects of relationships and provides different representations. The outputs of the multiple heads are concatenated and linearly projected to generate the final self-attention output.

Feedforward

It allows the model to capture more complex interactions between words. The feedforward neural network's application is position-wise and independent for each word, ensuring that the model retains the context awareness captured by self-attention. Phases in feed-forward

1. **Output from Self-Attention:** After passing through the multi-head self-attention mechanism, each word in the input sequence has been processed to capture contextual relationships with other words. The outputs of self-attention are often referred to as "contextual embeddings" because they represent the words in a context-aware manner.

2. Feedforward Neural Network: The output of the self-attention mechanism is passed through a feedforward neural network, which consists of two linear layers with an activation function in between.
3. Position-Wise Processing: The feedforward neural network is applied position-wise to each word's contextual embeddings independently. This means that the transformation is applied separately to each position in the sequence, preserving the relationships captured by the self-attention mechanism.
4. Activation Function: A non-linear activation function, typically the Rectified Linear Unit (ReLU), is applied element-wise after the first linear layer. This introduces non-linearity to the transformation and allows the model to learn complex mappings between inputs and outputs.
5. Dimensionality: The feedforward neural network's intermediate layer typically has a larger number of hidden units (higher dimensionality) than the input and output layers. This allows the network to capture more intricate patterns in the data.
6. Output: The final output of the feedforward neural network is used as the input to the next transformer layer or as the output of the current transformer block.

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',  
num_labels=2)
```

It combines the core BERT model, which includes self-attention and feedforward layers, with additional layers tailored for classification.

The additional layers include a classification head that takes the pooled output from the [CLS] token (which summarizes the entire sequence) and passes it through linear layers to produce classification logits.

Pre-training:

Pre-training involves predicting masked words in sentences (Masked Language Model) and predicting whether two sentences follow each other (Next Sentence Prediction). The model learns a wide range of language features, making its embeddings useful for various downstream NLP tasks without extensive fine-tuning.

15% of tokens were selected for prediction, and the training objective was to predict the selected token given its context [8].

Next sentence prediction: Given two spans of text, the model predicts if these two spans appeared sequentially in the training corpus, outputting either [IsNext] or [NotNext]. The first span starts with a special token [CLS] (for "classify"). The two spans are separated by a special token [SEP] (for "separate"). After processing the two spans, the 1-st output vector (the vector coding for [CLS]) is passed to a separate neural network for the binary classification into [IsNext] and [NotNext][8].

```
# Splitting the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Checking the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

Load the data from the cloud and split the data set into two parts there are training and testing where the training is 80 % and testing is 20 % of the loaded data.

```
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True)
test_encodings = tokenizer(X_test.tolist(), truncation=True, padding=True)
```

The tweet data loaded from the training dataset is the are converted to individual tokens. To avoid fluctuations in the length of the sequences we are adding padding to shorter tweets whit this there is no need to maintain the variable to store the length. In case the tweet is too long, it is truncated to a specific length.

```
# Defining batch size and creating data loaders
batch_size = 16
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)
```

Defining data loaders to handle tokenized and encoded data during training and testing. `batch_size` specifies the number of data samples that will be processed in each iteration of training or evaluation. Smaller batch sizes may offer more stability during training. For the training data loader shuffles the data at the beginning of each epoch. This help the model to learn unique task in the iteration.

```
# Defining optimizer and loss function
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
loss_fn = torch.nn.CrossEntropyLoss()
```

An optimizer is responsible for updating the model's parameters based on the computed gradients during backpropagation. We are using the `AdamW` optimizer, which is an optimization algorithm based on the Adam optimizer with weight decay. The learning rate for the optimizer, controls the size of the steps taken in the parameter space during optimization.

The loss function is used to calculate the difference between the predicted outputs of the model and the actual labels. For binary classification tasks like sentiment analysis, loss function internally combines the softmax activation and the negative log-likelihood loss.

```

model.train()
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        logits = outputs.logits
        loss.backward()
        optimizer.step()

```

The model is trained on the entire data set three times to train at which it will minimize loss and best accuracy. The value is achieved through cross-validation techniques. For each iteration, the model is trained on all batches from batch loaders of tokenized and training data to capture complex patterns in the language. When Huge data is processed the device is processing is transferred to GPU. The gradients are forced to change their value to zero as they accumulate gradients from previous iterations. Then compute the prediction and loss from the model in the model forward pass and back propagate to compute the gradients. At last, update the model's parameters for gradient-based optimization.

```

# Evaluation on the test set
model.eval()
predictions = []
with torch.no_grad():
    for batch in test_loader:
        input_ids, attention_mask, _ = batch
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        _, predicted_labels = torch.max(logits, dim=1)
        predictions.extend(predicted_labels.tolist())

```

For evaluating the model, we switch the model to evaluation mode So it will not change in gradients. The input data from the test loader is passed to the model for evaluating the performance of the model. The predictions from the model. The predictions have a classification of two probabilities where we choose the maximum probity as the output as positive or negative by giving dimensions as one. Finally, we are appending to a list in which it will have all the prediction.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
```

The accuracy score is calculated as the ratio of correctly predicted labels to the total number of samples in the test set. It's a metric used to evaluate classification models. Higher accuracy indicates better model performance.

```
def tokenize_and_infer(text):
    input_encoding = tokenizer.encode_plus(
        text,
        truncation=True,
        padding=True,
        max_length=128,
        return_tensors='pt'
    )

    input_ids = input_encoding['input_ids'].to(device)
    attention_mask = input_encoding['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probabilities = torch.softmax(logits, dim=1)
        _, predicted_label = torch.max(probabilities, dim=1)

    return predicted_label.item(), probabilities[0][0].item(),
probabilities[0][1].item()
```

we put our trained sentiment analysis tool to work, examining a large dataset of 100,000 rows. This Twitter data is passed through the trained model, figuring out if it was positive, or negative.

RESULTS

Through an intense data processing we standardize the data for modeling. Next, we shuffled and create 16 batches from the modeling. Then in modeling we achieved impressive accuracy of 83 percent. Sample results illustrating sentiment classification across various contexts demonstrate the model's proficiency. We outperformed the existing system performance by integrating S3 for storage and EC2 for computing. Furthermore, efficient handled the 1.6 Million records by parallel processing with aid of Apache spark.

```
In [5]: df
```

```
Out[5]:
```

	target	date	user	text	DayofWeek	Predicted Label	N_prob	P_prob
0	0	June 20 2009	VanessaLilyxo	day of arrands with mama., fun to spend some q...	Saturday	0	0.976704	0.023296
1	0	May 02 2009	fi69	@velvetella what time you done? poor you. x	Saturday	0	0.984709	0.015291
2	0	May 30 2009	OINKOINKLOVE	I've been very nice. I just don't know what'...	Saturday	0	0.997973	0.002027
3	1	May 01 2009	Ballyhoo	Thanks 2 all new followers! I will try and do ...	Friday	1	0.002335	0.997665
4	1	June 15 2009	kvnmcI	@peterford Happy Birthday and enjoy your day	Monday	1	0.003555	0.996445
...
99995	0	June 02 2009	joewestbrook	@themainecanada I'm not in Cali	Tuesday	0	0.993586	0.006414
99996	1	June 01 2009	nmyers89	What a lovely way to start to the day.	Monday	1	0.002594	0.997406
99997	0	June 02 2009	KComer	@HitzProductions thats not fair tho u cnt put...	Tuesday	0	0.996631	0.003369
99998	0	June 16 2009	Mz_Mann	@DeDeSizzling I have no username there. Your ...	Tuesday	0	0.849989	0.150011
99999	1	June 15 2009	haagmans	ow yeay: review, angels and demons.. it was ok...	Monday	1	0.001742	0.998258

100000 rows x 8 columns

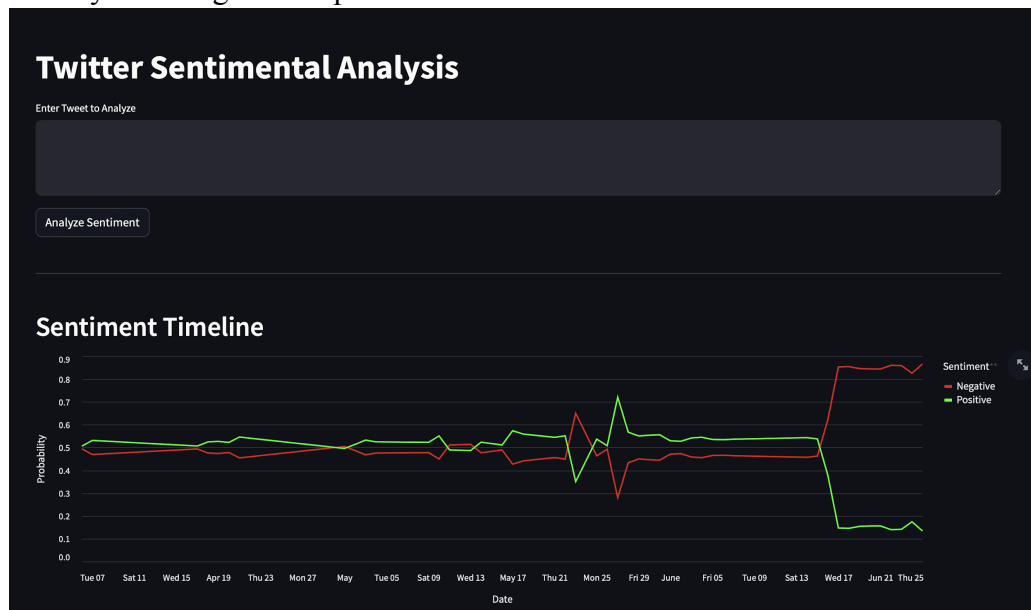
```
In [3]: from sklearn.metrics import accuracy_score
```

```
In [4]: print(accuracy_score(df['target'], df['Predicted Label']))
```

0.83876

The model output is a probability of the positivity which is classified into positive and negative tweets based on the threshold value.

The unique feature of the project is an interactive dashboard which accepts the tweets from the user and classify it as negative or positive.



Analysis on Weekdays

	DayofWeek	Negative	Positive	Total_count
1	Monday	0.4795	0.5205	19,388
5	Tuesday	0.5623	0.4377	11,452
6	Wednesday	0.6658	0.3342	5,984
4	Thursday	0.6959	0.3041	6,661
0	Friday	0.5328	0.4672	14,153
2	Saturday	0.5151	0.4849	20,813
3	Sunday	0.4706	0.5294	21,549

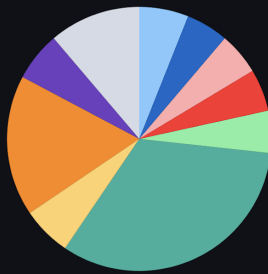
Legend: Negative (Blue), Positive (Orange)

Day	Negative (%)	Positive (%)
Monday	0.48	0.52
Tuesday	0.56	0.44
Wednesday	0.67	0.33
Thursday	0.70	0.30
Friday	0.53	0.47
Saturday	0.52	0.48
Sunday	0.47	0.53

[illegible]

Top 10 Users

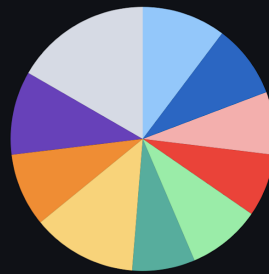
Top Ten Users with Most Negative Tweets



user

- MsJerzi
- fallwatcher
- fifthand56th
- kateyy_
- ksekher
- lost_dog
- mustntgrumble
- tweetpet
- twishes
- wowlew

Top Ten Users with Most Positive Tweets



user

- KevinEdwardsJr
- longestpoem
- scarletmandy
- tweetradder1
- tweetradder14
- tweetradder2
- tweetradder4
- tweetradder5
- what_bugs_u
- what_bugs_u

DISCUSSION OF RELATED WORK AND COMPARISON

The literature presented in the works of Imane El Alaoui and Youssef Gahi, Nusrat Jahan Prottasha et al., Go et al., and Vaswani et al. is closely relates with our project's domain of sentiment analysis, albeit with varied approaches and focuses. El Alaoui and Gahi's work underscores the significance of addressing big data challenges in sentiment analysis, specifically in the context of Twitter during a major event like the US election[4].

Prottasha et al.'s exploration of transfer learning through BERT BERT's benefits over context-independent approaches closely mirrors our pursuit of improved sentiment analysis accuracy through advanced models[5].

Vaswani et al.'s introduction of transformer architecture for sentiment analysis speaks to the evolution of techniques, emphasizing the relevance of adapting advanced structures to tackle complex language tasks. This aligns with our project's integration of BERT models, showcasing the dynamic landscape of modern sentiment analysis methodologies[6].

In comparison to these works, our project stands as an exploration of sentiment analysis, by creating a powerful dashboard to visualize and understand toxicity trends within Twitter data. While the above works contribute unique insights and methodologies, our project aims to provide a broader understanding of sentiment patterns and dynamics, aided by the integration of the cloud, big data technique with sparks, and visualization techniques.

LIMITATIONS AND POSSIBLE EXTENSIONS

While our study has provided us with valuable insights into sentiment patterns, there are some areas that have further exploration. The reasons behind the observed day-of-week sentiment trends could involve analyzing external influences such as news events or shifts in market trends. Additionally, examining the connection between tweet length, sentiment, and engagement might reveal whether shorter messages tend to convey stronger emotions. It's worth noting that richer data, including location information, could uncover more findings—enabling us to find sentiment trends within specific areas and regions. However, we must acknowledge certain limitations and opportunities for future extensions in our study. Our analysis primarily focuses on observable patterns and correlations, and while we can find potential causal relationships, cause-and-effect which require more analysis.

CONCLUSION

In conclusion, this project has achieved upgrading of sentiment analysis for Twitter data. By successfully implementing BERT and machine learning techniques, we built an application of sentiment analysis for understanding public sentiment which aids better decision-making processes. We used an advanced technique like BERT, which secures the context of the language while processing the text. By using the transfer learning technique, we also reduced the training time which utilizes existing knowledge of language understanding. To address the storage issue, we integrated Amazon S3 storage and utilized Apache Spark on an EC2 increasing scalability and distributed processing. In addition to this, we created an insightful dashboard with tweet toxicity trends on daily bias. Furthermore, we add a text box in which a text the model can be classified based on the toxicity of the text. By utilizing powerful tool for sentiment analysis and trend visualization, we've added depth to the understanding of opinions.

REFERENCES

- [1] <https://instances.vantage.sh/aws/ec2/t2.micro>
- [2] https://aws.amazon.com/s3/?did=ft_card&trk=ft_card
- [3] https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- [4] <https://www.sciencedirect.com/science/article/pii/S1877050919317077>
- [5] <https://www.mdpi.com/1424-8220/22/11/4157>
- [6] <https://www-cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf>
- [7] <https://iq.opengenus.org/embeddings-in-bert/#:~:text=Embeddings%20are%20nothing%20but%20vectors,make%20the%20final%20input%20token.>
- [8] [https://en.wikipedia.org/wiki/BERT_\(language_model\)](https://en.wikipedia.org/wiki/BERT_(language_model))
- [9] <https://spark.apache.org/docs/latest/api/python/index.html#:~:text=PySpark%20is%20the%20Python%20API,for%20interactively%20analyzing%20your%20data.>

APPENDICES

Table of Contents

Appendix A: User's Manual.....	
Appendix B: Implementation Manual.....	
Appendix C: Source Code.....	

Appendix A: User's Manual

Table of Contents

1. Introduction
2. System Requirements
3. Installation
4. Using the Sentiment Analysis Dashboard Locally
5. Interpreting Results

1. Introduction

The Twitter Sentiment Analysis project is designed to analyze the sentiment of tweets using a pre-trained BERT model. This User Manual provides step-by-step instructions on how to set up the sentiment analysis dashboard locally, interpret the results, and get in touch with the support team.

2. System Requirements

- An internet-enabled device (computer, laptop etc).
- Web browser (Google Chrome, Mozilla Firefox, Safari, etc.).
- Stable internet connection.

3. Installation

- Libraries that are required to run the code like streamlit, transformers etc.
- Trained model which can be downloaded from <https://drive.google.com/file/d/1LxZEmsN3f1nxd-87vFo69VpIlBdnKHyP/view?usp=sharing>
- Post processed Dataset which can be downloaded from https://drive.google.com/file/d/1WzeLEAvTMXtCgwqsnVJo_1WbTw89-HP3/view?usp=sharing

4. Using the Sentiment Analysis Dashboard Locally

- Copy the code for dashboard.py from the report and create a .py file on your local machine.

- Navigate to the directory where the dashboard files are stored.
- Open the Streamlit dashboard file (e.g., `dashboard.py`) using a code editor.
- Run the Streamlit app using the command: `streamlit run dashboard.py`.
- The sentiment analysis dashboard will open in your default web browser.
- Enter the tweet you want to analyze in the input text area.
- Click the "Analyze Sentiment" button.
- The sentiment label (positive/negative) along with an associated color and emoji will be displayed.
- If the sentiment is positive, a falling snow animation will also appear.
- Explore the various visualizations on the dashboard to understand the sentiment analysis results in more detail.

5. Interpreting Results

- **Sentiment Label:** The sentiment label indicates whether the analyzed tweet is classified as "Positive" or "Negative."
- **Color and Emoji:** The color (green for positive, red for negative) and emoji (😊 for positive, 😞 for negative) provide a visual representation of the sentiment.
- **Timeline:** The timeline chart shows how the sentiment probabilities (negative and positive) vary over time.
- **Analysis on Weekdays:** This section displays how sentiment probabilities vary across different weekdays.
- **WordCloud on Weekdays:** Word clouds show the most frequent words in both negative and positive tweets for a selected weekday.
- **Top 10 Users:** These charts show the users with the most negative and positive tweets.

Appendix B: Implementation Manual

Table of Contents

1. Introduction
2. AWS Cloud Setup
3. Code Deployment
4. Data Preprocessing
5. Model Training and post processing
6. Spark and PySpark Configuration
7. Dashboard Deployment

1. Introduction

The Twitter Sentiment Analysis project involves analyzing tweet sentiment using a pre-trained BERT model and deploying the analysis on the AWS Cloud. This Implementation Manual provides guidance on setting up the project infrastructure, deploying the code, and configuring the necessary components.

2. AWS Cloud Setup

- Set up an AWS account and create an EC2 instance to run the Spark and PySpark code.
- Also create an s3 bucket to upload the initial dataset
- Configure security groups, IAM roles, and permissions to ensure access to required resources.
- Note down the instance's public IP or URL for accessing the dashboard.

3. Code Deployment

Upload the code files (Preprocessing, Modeling, Post-Processing, Dashboard) to the EC2 instance using SSH, SCP, or other methods.

4. Data Preprocessing

- Preprocessing.py is run for preprocessing

5. Model Training and post processing

- Modeling.py is run for training the model
- Trained model is saved and used for post processing. Post-Processing.py is used for the prediction.

6. Spark and PySpark Configuration

- Spark and PySpark is installed on the EC2 instance.
- The preprocessing, and post-processing steps are running using PySpark on the EC2 instance.

7. Dashboard Deployment

- First installed Streamlit on the EC2 instance.
- Used the dashboard.py file for the dashboard.
- Deployed the Streamlit app on the EC2 instance using the instance's URL.

Details on the code is explained in approach section.

EC2

The screenshot displays the AWS Management Console interface for an EC2 instance. The left sidebar shows navigation options like 'EC2 Dashboard', 'Instances', 'Images', 'Elastic Block Store', and 'Network & Security'. The main content area shows a table of instances with one instance, 'Sentimental Analysis', in a 'Stopped' state. Below the table, the 'Details' tab for instance 'i-0c406158f445d6da5' is expanded, showing various attributes such as Instance ID, IP addresses, Hostname type, and IAM Role.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Sentimental A...	i-0c406158f445d6da5	Stopped	t2.micro	–	No alarms	us-east-2c

Instance: i-0c406158f445d6da5 (Sentimental Analysis)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Instance summary

Instance ID i-0c406158f445d6da5 (Sentimental Analysis)	Public IPv4 address –	Private IPv4 addresses 172.31.35.160
IPv6 address –	Instance state Stopped	Public IPv4 DNS –
Hostname type IP name: ip-172-31-35-160.us-east-2.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-35-160.us-east-2.compute.internal	Elastic IP addresses –
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendation s. Learn more
Auto-assigned IP address –	VPC ID vpc-03af482f0d9383b78	Auto Scaling Group name –
IAM Role s3access	Subnet ID subnet-0c8562b83edd1c8a7	

Amazon S3

Buckets

Access Points

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens

Dashboards

AWS Organizations settings

Feature spotlight

AWS Marketplace for S3

Amazon S3 > Buckets > databasetwitter

databasetwitter

Info

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects (5)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI

Copy URL

Download

Open






Delete

Actions

Create folder

Upload

Find objects by prefix

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 model50000.pt	pt	August 11, 2023, 11:44:05 (UTC-06:00)	417.8 MB	Standard
<input type="checkbox"/>	 tweets100k.csv	csv	August 11, 2023, 22:44:47 (UTC-06:00)	10.5 MB	Standard
<input type="checkbox"/>	 tweets100kfinal.csv	csv	August 11, 2023, 22:45:00 (UTC-06:00)	14.4 MB	Standard
<input type="checkbox"/>	 tweets12.csv	csv	August 11, 2023, 11:35:56 (UTC-06:00)	210.3 MB	Standard
<input type="checkbox"/>	 tweets50k.csv	csv	August 11, 2023, 22:44:40 (UTC-06:00)	4.4 MB	Standard

29 | Page

Appendix C: Source Code

PREPROCESSING

```
from datetime import datetime
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf, when, lower, regexp_replace
from pyspark.sql.types import StringType, ArrayType
import re
import findspark
findspark.init()
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True) # Property used to
format output tables better

# Loading the CSV file
df = spark.read.csv('tweets12.csv')
df = df.select('_c0', '_c2', '_c4', '_c5') # Select required columns
df = df.withColumnRenamed('_c0', 'target').withColumnRenamed('_c2',
'date').withColumnRenamed('_c4', 'user').withColumnRenamed('_c5', 'text')

# Only using 50000 samples to train the model as free tier ec2 cannot handle
training on big data/ it takes more time
n = 50000
df = df.sample(False, fraction=n / df.count(), seed=42)

# Defining UDF to process the date
# Separating the day of the week and the date into different columns
def process_date(date_str):
    date_obj = datetime.strptime(date_str, '%a %b %d %H:%M:%S PDT %Y')
    day_of_week = date_obj.strftime('%A')
    month = date_obj.strftime('%B')
    day = date_obj.strftime('%d')
    year = date_obj.strftime('%Y')
    return [day_of_week, f"{month} {day} {year}"]

process_date_udf = udf(process_date, ArrayType(StringType()))
df.show()

# Applying the UDF to create new columns
df = df.withColumn('DayofWeek', process_date_udf(df['date'])[0])
df = df.withColumn('date', process_date_udf(df['date'])[1])
df = df.withColumn('target', when(df['target'] == 4, 1).otherwise(df['target']))

def tweet_cleaner(x):
    text = re.sub("[@&][A-Za-z0-9_]+", "", x) # Remove mentions
```

```

    text = re.sub(r"http\S+", "", text)          # Remove media links
    return text

# Registering the UDF
tweet_cleaner_udf = udf(tweet_cleaner, StringType())

# Applying the UDF to clean the 'text' column
df = df.withColumn('text', tweet_cleaner_udf(df['text']))

# Converting all text to lowercase
df = df.withColumn('text', lower(df['text']))

# Removing newline characters
df = df.withColumn('text', regexp_replace(df['text'], '\n', ''))

# Replacing empty strings with null values
df = df.withColumn('text', when(df['text'] != '', df['text']))

# Dropping rows with null values in 'text' column
df = df.na.drop(subset=['text'])

df.show()
# Save the processed DataFrame as CSV
df.select('target', 'date', 'DayofWeek', 'user',
'text').write.csv('tweets50k.csv', header=True, mode='overwrite')
df.show()
# Stop the Spark session
spark.stop()

```

MODELING

```

import torch
from transformers import BertTokenizer, BertForSequenceClassification
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Loading the pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=2)

import pandas as pd
from sklearn.model_selection import train_test_split

```

```

df = pd.read_csv('tweets50k.csv')
X = df['text'] # Features
y = df['target'] # Target variable

# Splitting the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Checking the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# Set device (GPU if available, else CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Tokenize and encode the text data
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True)
test_encodings = tokenizer(X_test.tolist(), truncation=True, padding=True)

# Converting encodings to PyTorch tensors
train_dataset =
torch.utils.data.TensorDataset(torch.tensor(train_encodings['input_ids']),
                                torch.tensor(y_train.tolist()))
test_dataset =
torch.utils.data.TensorDataset(torch.tensor(test_encodings['input_ids']),
                                torch.tensor(y_test.tolist()))

# Defining batch size and creating data loaders
batch_size = 16
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

# Defining optimizer and loss function
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
loss_fn = torch.nn.CrossEntropyLoss()

# Training loop

```



```

num_epochs = 3
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        logits = outputs.logits
        loss.backward()
        optimizer.step()

    # Evaluation on the test set
    model.eval()
    predictions = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids, attention_mask, _ = batch
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            _, predicted_labels = torch.max(logits, dim=1)
            predictions.extend(predicted_labels.tolist())

    # Calculate accuracy
    accuracy = accuracy_score(y_test, predictions)

    print(f"Epoch {epoch+1} - Accuracy: {accuracy:.4f}")
torch.save(model, 'model150kfinal.pt')

```

POST PROCESSING

```

from transformers import BertTokenizer, BertForSequenceClassification
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, ArrayType, FloatType
import torch

# Creating a Spark session
spark = SparkSession.builder.master("local[*]").getOrCreate()

```

```

spark.conf.set("spark.sql.repl.eagerEval.enabled", True)
# Loading the trained model
model = torch.load('model50000.pt', map_location=torch.device('cpu'))
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
df = spark.read.csv('tweets50k.csv', header=True, inferSchema=True)

# Loading the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Defining UDFs for tokenization and inference
def tokenize_and_infer(text):
    input_encoding = tokenizer.encode_plus(
        text,
        truncation=True,
        padding=True,
        max_length=128,
        return_tensors='pt' # Return PyTorch tensors
    )

    input_ids = input_encoding['input_ids'].to(device)
    attention_mask = input_encoding['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probabilities = torch.softmax(logits, dim=1)
        _, predicted_label = torch.max(probabilities, dim=1)

    return predicted_label.item(), probabilities[0][0].item(),
probabilities[0][1].item()

tokenize_and_infer_udf = udf(tokenize_and_infer,
returnType=ArrayType(FloatType()))

df.show()
# Applying the UDF to create new columns
df = df.withColumn('N_prob', tokenize_and_infer_udf(df['text'])[1])
df = df.withColumn('P_prob', tokenize_and_infer_udf(df['text'])[2])

from pyspark.sql.functions import when
df = df.withColumn('Predicted_Label', when(df['P_prob'] >= 0.5, 1).otherwise(0))

# Show the updated DataFrame
df.show()
# Saving the data
df.write.csv('tweetsfinal.csv', header=True, mode='overwrite')
# Stop the Spark session

```

```
spark.stop()
```

DASHBOARD

```
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
import numpy as np
import re
from tensorflow.keras.models import Model
import streamlit as st
from PIL import Image
import numpy as np
from numpy import vstack
import zipfile
import io
import pandas as pd
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import altair as alt
import nltk
import matplotlib.pyplot as plt

nltk.download('stopwords')

st.set_page_config(
    page_title="Twitter Sentimental Analysis",
    page_icon="✅",
    layout="wide",
)
df = pd.read_csv('tweets100kfinalk.csv')
model = torch.load('model150000.pt',map_location=torch.device('cpu'))
model.eval()
#function to run sentimental analysis on the given tweet
def run_sentiment_analysis(text):
    # Assuming you have already trained the model and have the trained model
    object

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
```

```

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

    # Tokenize and encode the input text
input_encoding = tokenizer.encode_plus(
    text,
    truncation=True,
    padding=True,
    max_length=128,
    return_tensors='pt'
)
input_ids = input_encoding['input_ids'].to(device)
attention_mask = input_encoding['attention_mask'].to(device)
outputs = model(input_ids, attention_mask=attention_mask)
logits = outputs.logits
probabilities = torch.softmax(logits, dim=1)
_, predicted_label = torch.max(probabilities, dim=1)

# Determine sentiment label and color
if predicted_label == 0:
    sentiment_label = "Negative"
    sentiment_color = "red"
    sentiment_emoji = "😞"
else:
    sentiment_label = "Positive"
    sentiment_color = "#33FF57" # Bright green color
    sentiment_emoji = "😊"

return sentiment_label, sentiment_color, sentiment_emoji

# Display the title and text input area
st.title("Twitter Sentimental Analysis")
# text box for typing the tweet
txt = st.text_area('Enter Tweet to Analyze', '')
# button for calling the function to classify the tweet
if st.button('Analyze Sentiment'):
    sentiment_label, sentiment_color, sentiment_emoji =
run_sentiment_analysis(txt)
    st.write('', f'<font color="{sentiment_color}" size="+10">{sentiment_label}
{sentiment_emoji}</font>', unsafe_allow_html=True)

    if sentiment_label=='Positive':
        st.snow()

st.divider()

# Showing the timeline
st.header("Sentiment Timeline")

```

```

custom_color_scheme = alt.Scale(
    range=['#e41a1c', '#00FF00', '#4daf4a', '#984ea3', '#ff7f00', '#ffff33',
'#a65628']
)
timeline = grouped_data = df.groupby('date').agg({
    'N_prob': ['mean', 'count'], # Calculate the mean and count of 'N_prob' for
each day
    'P_prob': 'mean' # Calculate the mean of 'P_prob' for each day
}).reset_index()
timeline.columns = ['Date', 'Negative', 'Total_count', 'Positive']
timeline = pd.melt(timeline, id_vars=['Date', 'Total_count'],
value_vars=['Negative', 'Positive'],
                var_name='Sentiment', value_name='Probability')
def create_line_chart(data):
    chart = alt.Chart(data).mark_line().encode(
        x='Date:T',
        y='Probability:Q',
        color=alt.Color('Sentiment:N', scale=custom_color_scheme), # Specify the
color scheme here
        tooltip=[alt.Tooltip('Date:T', title='Day of Week'),
alt.Tooltip('Probability:Q', title='Percentage'),
alt.Tooltip('Total_count:Q', title='Total_count')]
    ).properties(
        title=''
    )
    return chart
chart = create_line_chart(timeline)

st.altair_chart(chart, theme="streamlit", use_container_width=True)

st.divider()
# showing the graph on weekdays

st.header("Analysis on Weekdays")

grouped_data = df.groupby('DayofWeek').agg({
    'N_prob': 'mean', # Calculate the mean and count of 'N_prob' for each day
    'P_prob': ['mean', 'count'] # Calculate the mean and count of 'P_prob' for
each day
}).reset_index()

# Flatten the multi-level column index
grouped_data.columns = ['DayofWeek', 'Negative', 'Positive', 'Total_count']

order_of_weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']

```



```

        return pd.Series([text])
df[['plain_text']] = df.text.apply(tweet_cleaner)
#Convert all text to lowercase
df.plain_text = df.plain_text.str.lower()
#Remove newline character
df.plain_text = df.plain_text.str.replace('\n', '')
#Replacing any empty strings with null
df = df.replace(r'^\s*$', np.nan, regex=True)
if df.isnull().sum().plain_text == 0:
    print('no empty strings')
else:
    df.dropna(inplace=True)

dayfilter = st.selectbox("Select the Day of the Week", ['Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
from nltk.corpus import stopwords
col1, col2 = st.columns(2)

stop_words = set(stopwords.words('english'))

df1 = df[df['DayofWeek']==dayfilter]
df2 = df1[df1['Predicted Label']==0]

query_words={'recession', '#', 'http' }
stop_words.update(query_words)
for word in query_words:
    df2.plain_text = df2.plain_text.str.replace(word, '')
#Creating word cloud
from wordcloud import WordCloud, ImageColorGenerator
wc=WordCloud(stopwords=stop_words, collocations=False, max_font_size=55,
max_words=50, background_color="black")
wc.generate(' '.join(df2.plain_text))
with col1:
    st.header("Negative Tweets")
    st.image(wc.to_array(),width=620)

df3 = df1[df1['Predicted Label']==1]

query_words={'recession', '#', 'http' }
stop_words.update(query_words)
for word in query_words:
    df3.plain_text = df3.plain_text.str.replace(word, '')
#Creating word cloud
from wordcloud import WordCloud, ImageColorGenerator
wc=WordCloud(stopwords=stop_words, collocations=False, max_font_size=55,
max_words=50, background_color="black")
wc.generate(' '.join(df3.plain_text))

```

```

with col2:
# Display the word cloud using Streamlit
    st.header("Positive Tweets")

    st.image(wc.to_array(), width=620)

st.divider()

#showing top 10 users that post most negative and most positive tweets.

st.header("Top 10 Users")

col1,col2=st.columns(2)

grouped_data = df.groupby('user').agg({
    'N_prob': ['mean', 'count'],
    'P_prob': ['mean', 'count']
}).reset_index()

# Flatten the multi-level column names
grouped_data.columns = ['user', 'N_prob_mean', 'N_count', 'P_prob_mean',
'P_count']

# Filter users with a count greater than 5
grouped_data = grouped_data[(grouped_data['N_count'] > 5) &
(grouped_data['P_count'] > 5)]

# Sort the DataFrame based on N_prob_mean and N_prob_count in descending order
top_negative_users_df = grouped_data.sort_values(['N_prob_mean', 'N_count'],
ascending=[False, False]).head(10)

# Sort the DataFrame based on P_prob_mean and P_prob_count in descending order
top_positive_users_df = grouped_data.sort_values(['P_prob_mean', 'P_count'],
ascending=[False, False]).head(10)

with col1:

    chart = alt.Chart(top_negative_users_df).mark_arc().encode(
        theta='N_count:Q',
        color='user:N',
        tooltip=['user:N', 'N_count:Q']
    ).properties(
        title='Top Ten Users with Most Negative Tweets'
    )
    # Display the pie chart

```



```
st.altair_chart(chart, use_container_width=True)

with col2:

    chart = alt.Chart(top_positive_users_df).mark_arc().encode(
        theta='P_count:Q',
        color='user:N',
        tooltip=['user:N', 'P_count:Q']
    ).properties(
        title='Top Ten Users with Most Positive Tweets'
    )
    # Display the pie chart
    st.altair_chart(chart, use_container_width=True)
```