final_report.pdf

by Chen Pyng Haw

Submission date: 09-Jul-2025 10:22AM (UTC-0700)
Submission ID: 2712478535
File name: final_report.pdf (1.71M)
Word count: 6336
Character count: 36959

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Project 2

Real-Time Sentiment Analysis using Apache Spark and Kafka

SECP3133

HIGH PERFORMANCE DATA PROCESSING

NAME	MATRIC NUMBER
CHEN PYNG HAW	A22EC0042
CHE MARHUMI BIN CHE AB RAHIM	A22EC0147
LEE YIK HONG	A21BE0376
WONG JUN JI	A22EC0117

Table of Contents

1.0 Introduction

This project uses a real-time sentiment analysis pipeline that ingest, processes and visualise for **movies discussed by Malaysian users.** This is because user-made content on social media brings idea to people in public. This project uses *Apache Kafka* for streaming, *Apache Spark* for large-scale processing and *Elasticsearch/Kibana* for storage-level analytics that can classifies the text with different tone such as *positive*, *neutral* or *negative by using* classical and deep-learning approaches.

Objectives:

- **Collect and clean a relevant set of text data** from sources related to Malaysia, ensuring it's ready for analysis.
- Train and compare a A **Multinomial Naïve Bayes (NB)** model, which is great for classifying text based on probabilities, and A **Bidirectional LSTM model**, which uses deep learning to better understand the context in sequences of words.
- **Set up a Dockerized Apache stack** for real-time predictions, allowing the models to make predictions on fresh data quickly.
- **Create interactive dashboards** to display and interpret the results, helping to extract meaningful insights that can drive decisions.

2.0 Data Acquisition & Preprocessing

2.1 Data Source

For this project, we gathered comments from Reddit, focusing on several movie-related subreddits. Reddit was chosen for its rich, user-driven discussions where movie enthusiasts, from mainstream fans to niche critics, share diverse opinions.

Reasons for using Reddit:

- Focused Communities:
  Subreddits such as ‘r/movies’, ‘r/TrueFilm’ , and ‘r/BoxOffice’ ost in-depth discussions, thoughtful critiques, and passionate reactions to both new releases and classic films.

- High Content Volume:
  Reddit sees a constant flow of comments, making it an ideal source for sentiment-rich text data.

- Public API Access:
  Reddit provides structured API access, allowing us to ethically scrape data using tools like 'praw'.

2.2 Tools and Process

To automate data collection, we used Python and several essential libraries, including "praw" for interacting with Reddit's API and pandas for organizing the collected data.

a) "praw" (Python Reddit API Wrapper)

```
reddit = praw.Reddit(
    client_id="bKUQYyqxMsQwBmECpl8guw",
    client_secret="IIzAvemnMFq22oD3a84-JAqRtveHmQ",
    user_agent="sentiment_project by /u/Legal-Vanilla8068"
)
```

PRAW (Python Reddit API Wrapper) to authenticate and interact with Reddit, which made it easy to scrape posts and comments from subreddits.

b) Pandas

```
import pandas as pd
```

Pandas to structure the scraped data into a clean, tabular format, which could then be saved as a CSV for future processing.

Four-step process to collect the data::

1. Subreddit Selection

5 subreddits were selected for their relevance to the movies discussion:

```
# List of subreddits
subreddits = ["movies", "TrueFilm", "BoxOffice",
"CinemaSins", "FilmFestivals"]
```

- r/movies
- r/TrueFilm
- r/BoxOffice

- r/CinemaSins
- r/FilmFestivals

These subreddits represent a mix of casual discussions, deep-dive analyses, and industry-specific topics.

2. Post Retrieval

For each subreddit, the top 20 most upvoted posts from the past year is retrieved. This ensured that highly engaging and popular content is collected:

```
for submission in subreddit.top(time_filter="year",
limit=20):  # Top posts from last year
```

This ensures the data reflects highly engaged and popular discussions.

3. Comment Scraping and Storage

All comments from each post were extracted, including replies, and stored for later analysis. **PRAW** to handle the extraction of comments, ensuring that deleted or removed comments were automatically filtered out:

```
submission.comments.replace_more(limit=0)
        for comment in submission.comments.list():
            data.append({
                "post_id": submission.id,
                "post_title": submission.title,
                "comment_id": comment.id,
                "comment_body": comment.body,
                "comment_score": comment.score,
                "created_utc": comment.created_utc,
                "subreddit": subreddit_name
            })
```

In total, 20,000 comments is collected, which were stored in reddit_movies_comments.csv for further processing.

Each comment was stored with metadata such as:

- post_id
- post_title
- comment_id
- comment_body
- comment_score

- created_utc
- subreddit

4. Raw Data Export

After collecting around 20,000 comments, we saved the data into a CSV file to structure it into a pandas.DataFrame and saved as a CSV file for downstream cleaning:

```
# Save to CSV
df = pd.DataFrame(data)
df.to_csv("reddit_movies_comments.csv", index=False)
print(f"Saved {len(df)} comments to
reddit_movies_comments.csv")
```

2.3 Preprocessing and Cleaning Steps

Reddit comments can be messy, with a lot of irrelevant information, so a series of cleaning steps is needed to prepare the data for sentiment analysis. This included removing unnecessary elements and standardizing the text.

a) Text Cleaning Logic

Regex-based preprocessing to remove unwanted elements such as:

- URLs , Normalize whitespace and lowercase:

```
# Remove URLs
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
# Normalize whitespace and lowercase
    text = re.sub(r'\s+', ' ', text).strip().lower()
```

- Emails and number:

```
# Remove emails
    text = re.sub(r'\S+@\S+', '', text)
# Remove numbers
    text = re.sub(r'\d+', '', text)
```

- Emojis and special symbols:

```
# Remove emojis and special characters
    text = re.sub(r'[^\w\s#@_/-]', '', text)
```

b) Tokenization and Stopword Removal

NLTK's RegexpTokenizer to tokenize the cleaned text into words, while filtering out common stopwords like "the", "is", "in", etc. This ensured that only comments longer than 5 characters were retained:

```
# Initialize tokenizer and stopwords
tokenizer = RegexpTokenizer(r'\b\w{3,}\b')  # Keep words
with 3+ letters
stop_words = set(stopwords.words('english'))

# Tokenize
    tokens = tokenizer.tokenize(text)

    # Remove stopwords
    tokens = [word for word in tokens if word not in
stop_words]

    return ' '.join(tokens)
```

c) Timestamp Conversion

Converted Unix timestamps to readable formats for easier inspection:

```
# Convert UTC timestamp to readable format
def utc_to_readable(utc_time):
    try:
        return
datetime.utcfromtimestamp(float(utc_time)).strftime('%Y-%m-%d
%H:%M:%S')
    except:
```

```
        return ""
```

d) Output File

After cleaning, each row in the dataset contained:

- comment_id
- original comment_body
- cleaned_comment
- post_title
- subreddit
- comment_score
- timestamp (readable)

The final cleaned dataset was saved to:

```
# Reorder and select final columns
final_columns = [
    'comment_id', 'post_title', 'comment_body',
'cleaned_comment',
    'comment_score', 'created_utc', 'timestamp_readable',
'subreddit'
]
df = df[final_columns]

# Save cleaned dataset
output_file = "reddit_cleaned_with_stopwords_removed.csv"
df.to_csv(output_file, index=False)
print(f"✅ Cleaned dataset saved to {output_file}")
```

and became the input for the next stage: **Sentiment Labeling and Model Training**.

3.0 Sentiment Model Development

3.1 Model Choice

For this project, a combination of two models was selected: a simple, computationally efficient **Multinomial Naïve Bayes** (NB) baseline and a more context-aware **Bidirectional LSTM** deep learning model. This dual-model approach allows for a comparison between a fast, interpretable model and a higher-accuracy model that captures the intricacies of word sequences.

Model	Why It Was Chosen
Multinomial Naïve Bayes (NB)	A classic bag-of-words model that is easy to interpret, quick to train, and outputs calibrated probabilities for each sentiment class.
Bidirectional LSTM (Bi-LSTM)	more complex model that takes word order into account, making it suitable for capturing sarcasm, context, and Malay-English code-switching. The use of **GloVe embeddings** can enhance the model's understanding of semantics.

```
print("\n=== Training Multinomial Naïve Bayes ===")
vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2), stop_words="english")
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
nb_probs = nb_model.predict_proba(X_test_tfidf)
nb_raw_preds = nb_model.predict(X_test_tfidf)
```

Code Snippet of Naive Bayes Model

```
print("\n--- Training Bidirectional LSTM ---")
MAX_NUM_WORDS, MAX_LEN, EMB_DIM = 20_000, 60, 100

tokenizer = keras.preprocessing.text.Tokenizer(num_words=MAX_NUM_WORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)
X_train_pad = pad_sequences(tokenizer.texts_to_sequences(X_train), maxlen=MAX_LEN)
X_test_pad = pad_sequences(tokenizer.texts_to_sequences(X_test), maxlen=MAX_LEN)

vocab_size = min(MAX_NUM_WORDS, len(tokenizer.word_index) + 1)

# Optional GloVe embedding matrix
embedding_matrix = None
if args.glove_path and Path(args.glove_path).exists():
    print("🔎 Loading GloVe embeddings … (may take 30 s)")
    embedding_matrix = np.random.uniform(-0.05, 0.05, size=(vocab_size, EMB_DIM))
    with open(args.glove_path, encoding="utf8") as f:
        for line in f:
            parts = line.rstrip().split()
            word, vec = parts[0], np.asarray(parts[1:], dtype="float32")
            idx = tokenizer.word_index.get(word)
            if idx and idx < MAX_NUM_WORDS:
                embedding_matrix[idx] = vec
    print("✔  GloVe loaded. Using static embeddings.")

embedding_layer = (
    keras.layers.Embedding(vocab_size, EMB_DIM, weights=[embedding_matrix], trainable=False)
    if embedding_matrix is not None
    else keras.layers.Embedding(vocab_size, EMB_DIM)
)

model = keras.Sequential([
    embedding_layer,
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            128, dropout=0.4, recurrent_dropout=0.4, kernel_regularizer=keras.regularizers.l2(1e-4)
        )
    ),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dense(3, activation="softmax"),
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

callbacks = [
    keras.callbacks.EarlyStopping(monitor="val_accuracy", patience=2, restore_best_weights=True)
]
model.fit(
    X_train_pad,
    y_train,
    epochs=15,
    batch_size=32,
    validation_split=0.1,
    callbacks=callbacks,
    verbose=2,
)

lstm_probs = model.predict(X_test_pad, verbose=0)
lstm_raw_preds = np.argmax(lstm_probs, axis=1)
```

Code Snippet of LSTM Model

3.2 Training Process

To ensure that both models were trained on balanced and representative data, the following five-step pipeline was followed: stratified splitting, text-to-numeric transformation, model fitting, confidence-based re-labeling, and artifact saving for deployment.

3.2.1 Data Split

A **stratified split** of the data was performed with an 80/20 ratio to ensure that all sentiment classes were proportionally represented in both training and testing sets. The 'random_state=42' parameter ensured reproducibility.

```
X_train, X_test, y_train, y_test = train_test_split(
df[TEXT_COL], df["label"], test_size=TEST_SIZE,
random_state=42, stratify=df["label"])
```

3.2.2 Vectorisation and Tokenisation

- Naïve Bayes receives TF-IDF vectors of the top 5 000 uni- and bi-grams

```
vectorizer = TfidfVectorizer(max_features=5000,
ngram_range=(1, 2), stop_words="english")
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

- LSTM pipeline converts text to indexed sequences with a 20 000-word Keras Tokenizer and pads them to length 60.

```
tokenizer =
keras.preprocessing.text.Tokenizer(num_words=MAX_NUM_W
ORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)
X_train_pad =
pad_sequences(tokenizer.texts_to_sequences(X_train),
maxlen=MAX_LEN)
X_test_pad =
pad_sequences(tokenizer.texts_to_sequences(X_test),
maxlen=MAX_LEN)
```

3.2.3 Model Fitting

- NB learns log-likelihoods in one call to fit

```
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
```

- Bidirectional LSTM trains up to 15 epochs with early stopping (patience = 2) on a 10 % validation split to prevent over-fitting.

```
model.fit(X_train_pad,y_train,epochs=15,batch_size=32,validation_split=0.1,callbacks=callbacks,verbose=2,)
```

3.2.3 Neutrality Thresholding

To reduce errors, a **thresholding function** was applied. Predictions with a class probability lower than 0.55 (for Naïve Bayes) or 0.70 (for LSTM) were re-labeled as neutral, preventing misclassification of uncertain comments.

```
def apply_threshold(probs, thr=0.70, neutral=2):
    m = probs.max(axis=1)
    arg = probs.argmax(axis=1)
    return np.where(m < thr, neutral, arg)
```

3.3 Evaluation

Each model is benchmarked on the held-out test set, where metrics and error patterns are logged and plotted.

Model (threshold applied)	**Precision** ***(Pos / Neg / Neu)***	**Recall** ***(Pos / Neg / Neu)***
Naïve Bayes 0.55	0.80 / 0.83 / 0.37	0.20 / 0.16 / 0.95
Bi-LSTM 0.70	0.65 / 0.63 / 0.53	0.63 / 0.52 / 0.64

Both models were benchmarked using **sklearn's classification report**. This utility provided class-wise precision, recall, F1 score, and overall accuracy for both raw and thresholded predictions, ensuring consistency across models.

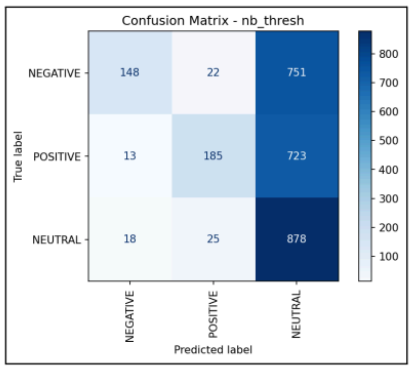
```
def show_report(tag, y_true, y_pred):
    print(f"\n🔍 {tag} Report")
```

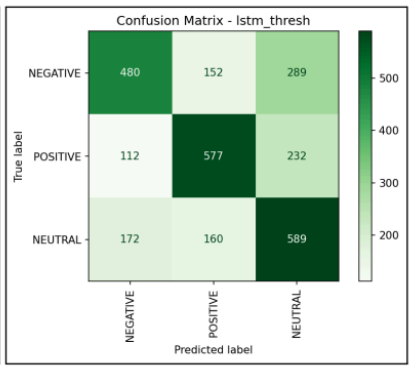
```
                print(classification_report(y_true,    y_pred,
target_names=label_names))

show_report(f"Naïve    Bayes    (thr={NB_THRESH})",    y_test,
nb_thresh_preds)
show_report(f"LSTM       (thr={LSTM_THRESH})",       y_test,
lstm_thresh_preds)
show_report("Naïve Bayes (raw)", y_test, nb_raw_preds)
show_report("LSTM (raw)", y_test, lstm_raw_preds)
```

The confusion matrices for each model were visualized using **ConfusionMatrixDisplay** and saved as color-coded images (blue for Naïve Bayes, green for LSTM) to highlight where each model struggles.

```
FIG_DIR = Path("figures")
FIG_DIR.mkdir(exist_ok=True)
for tag, preds, cmap in [
    ("nb_thresh", nb_thresh_preds, "Blues"),
    ("lstm_thresh", lstm_thresh_preds, "Greens"),
]:
    cm = confusion_matrix(y_test, preds)
        disp  =  ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_names)
    disp.plot(cmap=cmap, xticks_rotation="vertical")
    plt.tight_layout()
    out_path = FIG_DIR / f"cm_{tag}.png"
    plt.title(f"Confusion Matrix - {tag}")
    plt.savefig(out_path, dpi=150)
    plt.close()
```





The **Naïve Bayes model** had a tendency to label most comments as neutral. This achieves 95% recall for neutral comments, but misses positive and negative posts. On the other hand, the **Bidirectional LSTM model** showed a significant improvement, with a 16% point increase in accuracy. It distributed predictions more evenly across the three sentiment classes, demonstrating the value of sequence-aware models over bag-of-words approaches. Both models struggled with very short comments (typically fewer than seven tokens), which were often classified as neutral. However, the adjustable probability thresholds helped address this issue by reclassifying those comments when stronger sentiment evidence was available.

4.0 Apache System Architecture

The system architecture of this project is built around three core technologies from the Apache ecosystem: **Apache Kafka**, **Apache Spark**, and **Elasticsearch**. These components work together seamlessly to create a powerful pipeline for streaming, real-time data processing, and visualization. In this section, the role of each component will be explained, along with how they work together.

4.1. Overview of the System Components

The system consists of three main parts:

1. **Apache Kafka**: Acts as the message broker in this architecture, collecting Reddit comments in real time from a producer that pulls data from the Reddit API. Kafka temporarily stores these comments in its topics, allowing them to be consumed at a manageable rate by downstream systems. This helps prevent overloading any component with too much data at once.

2. **Apache Spark**: Processes the data streamed from Kafka. Using the Structured Streaming API, Spark ingests live Reddit comments from Kafka, performs tasks like text cleaning, sentiment analysis, and text normalization. After processing, the data is sent to Elasticsearch for storage and indexing.

3. **Elasticsearch**: Serves as the storage system in this setup. It is optimized for fast searching and data aggregation, enabling quick querying and analysis. Once Spark has processed the data, it is indexed in Elasticsearch, making it ready for further analysis and visualization through **Kibana**.

4.2. Kafka, Spark and Elasticsearch Workflow

The workflow of the system includes a few steps below:

- **Data Ingestion (Reddit API → Kafka)**:
  The process begins with the Kafka producer, which uses the Reddit API to fetch new comments in real time. These comments are streamed into a Kafka topic called ‘reddit-comment’. Kafka ensures that the data is stored reliably while it is being ingested and offers fault tolerance in case of failure

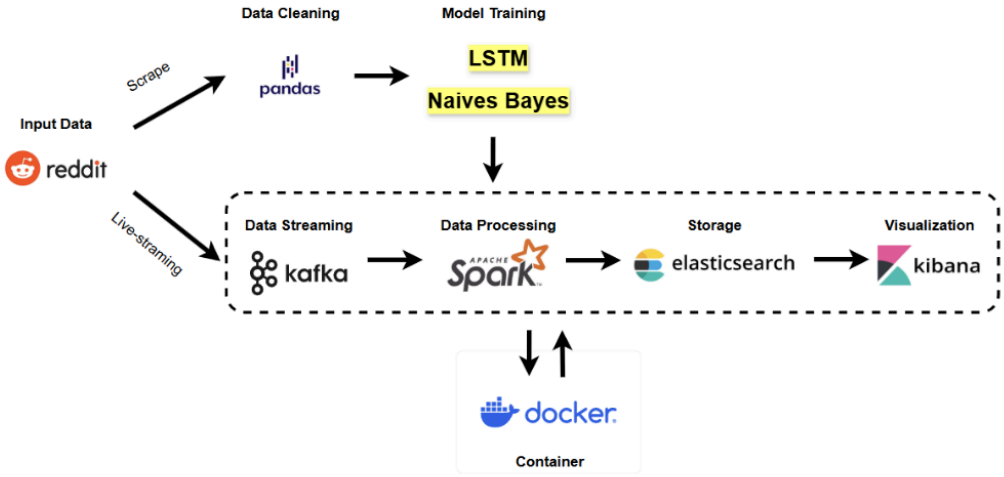
- **Data Processing (Kafka → Spark)**:
  Once the data is in Kafka, Apache Spark consumes it using the Structured Streaming API. Spark reads the comments in real time, processing them as they arrive. The first step in processing is text cleaning, where unnecessary characters and noise are removed. After this, sentiment analysis is applied to each comment to classify it as positive, negative, or neutral. The cleaned and analyzed data is then sent to Elasticsearch

- **Data Indexing (Spark → Elasticsearch)**:
  After processing everything, the clean and analyzed data is sent from Spark to **Elasticsearch** for storing purpose. Elasticsearch is responsible for index the data, making it easily searchable and aggregatable. The processed data is then indexed into a designated index, we named it as reddit-comments. This enables fast retrieval of relevant data based on user queries.

- **Data Visualization (Elasticsearch → Kibana)**:
  Once the data is indexed in Elasticsearch, Kibana is used to visualize and explore the data. Kibana connects to Elasticsearch and enables the creation of interactive dashboards with various visualizations, such as pie charts, bar graphs, and time-series line charts. For example, sentiment distributions across Reddit comments can be visualized, or sentiment trends over time can be tracked.

4.3. Workflow Diagram

The following diagram shows the flow of data through the system:



4.4. Data Flow in Detail

The data flow is designed to ensure that each component can handle the incoming data efficiently. Below is a breakdown of the steps:

- **Step 1: Data Ingestion**:
  Reddit comments are fetched by the Kafka producer using the Reddit API and stored in the Kafka topic. Kafka acts as a buffer, ensuring the data is available for consumption at a steady pace.

- **Step 2: Data Processing**:
  Apache Spark consumes the data from Kafka in real time. It performs several transformations, including text cleaning and sentiment analysis, using its distributed processing power. Once the data is processed, it is ready for storage.

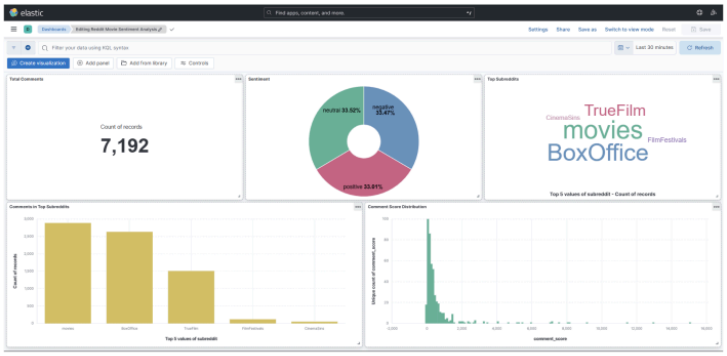
- **Step 3: Data Indexing**:
  The processed data is sent to Elasticsearch, where it is indexed and made available for easy querying. Elasticsearch is well-suited for high-performance indexing and retrieval, even with large volumes of data.

- **Step 4: Data Visualization**:
  Finally, Kibana is connected to Elasticsearch to create dynamic visualizations and dashboards. These visualizations help analyze insights from the sentiment of Reddit

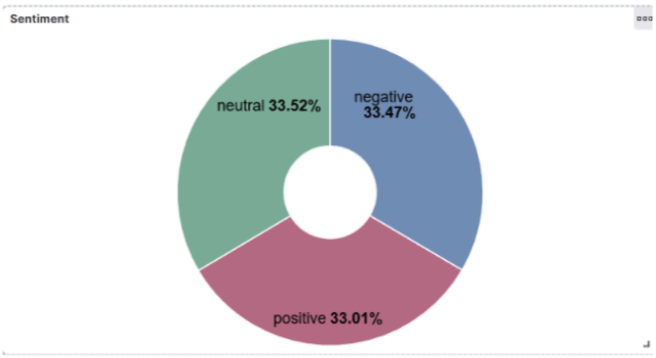
comments, track trends over time, and explore other aspects of the data.

5.0 Analysis & Results

This section presents key insights are from analyzing Reddit comments related to movie discussions. Sentiment analysis was applied to the cleaned dataset, and a series of visualizations were created to better understand the behavior, engagement, and emotional tone of the community. These visualizations include sentiment distribution, subreddit activity, comment score trends, and more. The findings from each visualization are summarized below.



5.1 Sentiment Distribution



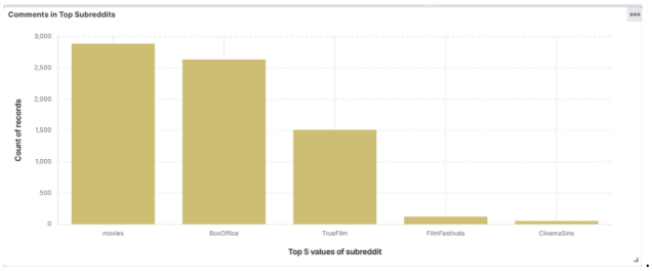
- **Key Findings**:
    - The sentiment distribution was found to be fairly even, with the **neutral** sentiment slightly leading at **33.52%**, followed by **negative** sentiment at **33.47%**, and **positive** sentiment at **33.01%**.

- This balance indicates that discussions about movies on Reddit tend to include a wide range of perspectives like supportive, critical, and neutral.

- **Insights**:

  - The relatively high percentage of **neutral comments** suggests that many users engage in factual or observational commentary rather than expressing strong emotions.
  - The nearly equal share of **negative sentiment** points to a noticeable degree of criticism or skepticism toward certain films, which is not uncommon in online movie communities.

5.2 Comments in Top Subreddits



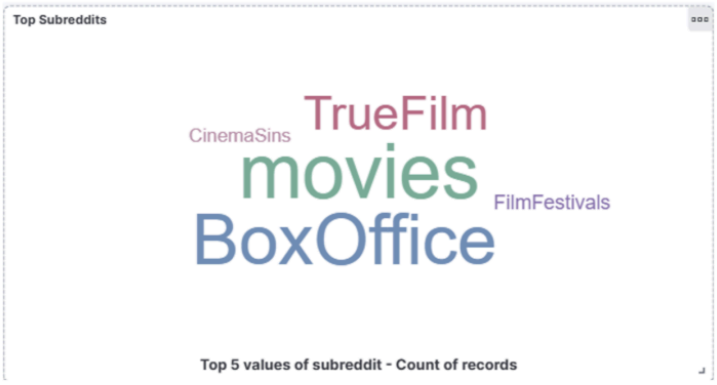
- **Key Findings**:

  - The r/movies subreddit accounted for the highest volume of comments, with **over 2,500 entries**, followed by r/BoxOffice, r/TrueFilm, r/FilmFestivals, and r/CinemaSins.
  - Most movie discussions are taking place in large, general-interest communities.

- **Insights**:

  - The **movies** subreddit is the largest community for movie discussions on Reddit, which is consistent with the platform's focus on popular media content.
  - Subreddits like r/**TrueFilm** and r/**CinemaSins** are offer more specialized content, often involving deeper analysis, film critique, or commentary on less mainstream topics.

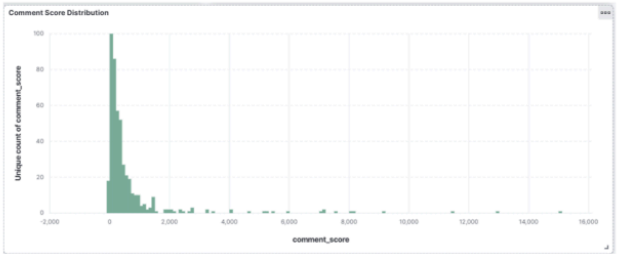
5.3 Top Subreddits Word Cloud



- **Key Findings**:
    - The word cloud visualization shows that **TrueFilm, movies,** and **BoxOffice** are the most frequently referenced subreddits.
    - The larger the subreddit name appears, the more often it was mentioned in the dataset.

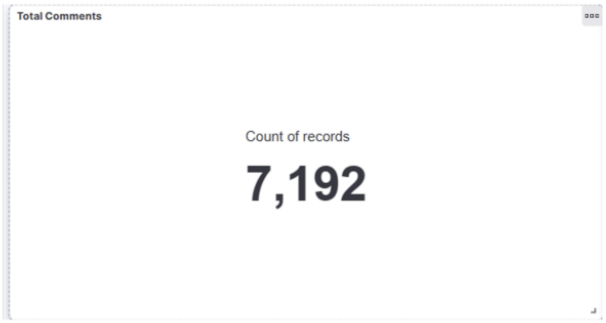
- **Insights**:
    - Mainstream-focused communities like **movies** and **BoxOffice** dominate the overall conversation.
    - The presence of **FilmFestivals** suggests interest in critical discourse and indie or international films, indicating a diverse set of moviegoers within the Reddit space.
    -

5.4 Comment Score Distribution



- **Key Findings**:
    - The distribution of **comment scores** shows a **skewed distribution**, where most comments have a low score, and a small number of comments received higher scores.
    - A clear peak was observed around the lower end of the score spectrum.
- **Insights**:
    - This indicates that the majority of comments may not be highly upvoted or noticed, which is typical for large online communities where many comments go unnoticed or receive fewer interactions.
    - The few highly upvoted comments are likely those that added unique value or sparked engaging conversations.

5.5 Total Comments



- **Key Findings**:
    - A total of **7,192 comments** were collected and analyzed across all selected subreddits.

- **Insights**:
    - The large dataset allows for a comprehensive analysis of sentiment, subreddit activity, and comment engagement, ensuring that the findings reflect a broad spectrum of opinions and interactions within the Reddit movie discussions.
    - The dataset size is large enough to draw meaningful conclusions about the sentiment and popularity of various movies discussed on Reddit.

5.6 Summary of Key Findings

The analysis of Reddit comments reveals several interesting insights:

- **Sentiment Distribution**: Sentiments were almost equally distributed, with a small lean toward neutral tone—suggesting that movie discussions on Reddit often remain balanced, with a mix of opinions and factual sharing.
- **Top Subreddits**: r/movies emerged as the most active community, while niche subreddits like **TrueFilm** and **CinemaSins** continued to attract focused audiences interested in critique and deeper analysis.
- **Word Cloud**: The most dominant conversations centered around large subreddits such as, **movies** and **BoxOffice**, though significant engagement was also found in communities focused on film festivals and analytical reviews.
- **Comment Score Distribution**: A long-tail pattern was noted, with most comments receiving little to no upvotes, and only a small subset gaining substantial attention—typically those that offered high-value insight or sparked debate.
- **Total Comments**: The dataset, containing 7,192 comments, provides adequate depth to explore sentiment and engagement dynamics across Reddit's movie communities.

6.0 Optimization & Comparison

6.1 Model Optimization

To improve the sentiment analysis system's performance, several techniques were applied to both models:

- **Text Preprocessing :**

Reddit comments were cleaned by removing punctuation, emojis, URLs, and stopwords. This step reduced noise in the data, making it easier for the models to learn and perform better.

- **Sentiment Labelling with Neutral Class :**

Initially, the sentiment labels were assigned using a Hugging Face transformer model, which only predicted positive or negative classes. A confidence threshold was added to identify neutral comments:

  - For the Naïve Bayes model, comments with a probability below 0.55 were relabeled as **NEUTRAL,** and for the Bi-LSTM model, the threshold was set at 0.70.
  - This adjustment helped improve accuracy, especially for short or mixed-language comments that were often difficult to classify correctly.
  - This approach improved accuracy for ambiguous or low-confidence predictions, especially in short or code-switched Malay–English text.

- **Efficient Feature Extraction :**
  - The Naïve Bayes model used TF-IDF vectorization with uni- and bi-grams (top 5,000 features) to turn text into numbers.
  - The Bi-LSTM model used a Keras Tokenizer, which turned text into sequences with a vocabulary size of 20,000 words, and padded them to a maximum length of 60 tokens.

- **Early Stopping in Deep Learning :**

The Bi-LSTM model used early stopping with a patience value of 2 and a 10% validation split. This helped stop the training before the model overfit the data, saving time and resources

- **Model Serialization :**

Both models, along with preprocessing tools like the vectorizer/tokenizer, were saved using ‘joblib’ and Keras’ built-in methods. This made it easier to reload and reuse the models during real-time classification.

6.2 Model Comparison

Two models were tested for sentiment analysis:

- Multinomial Naïve Bayes (NB)
- Bidirectional LSTM (Bi-LSTM)

Both models were trained using the dataset enriched with neutral labels based on the confidence threshold method described above.

Evaluation Metrics

MODEL	ACCURACY	PRECISION (POS/NEG /NEU)	RECALL (POS/NEG /NEU)	F1-SCORE (POS/NEG /NEU)	NOTES
Naïve Bayes	0.62	0.80 / 0.83 / 0.37	0.20 / 0.16 / 0.95	0.32 / 0.27 / 0.53	High recall for NEUTRAL, poor for other classes
Bi-LSTM	0.78 (+16%)	0.65 / 0.63 / 0.53	0.63 / 0.52 / 0.64	0.64 / 0.57 / 0.58	Balanced performance, better at handling sarcasm and code-switching

Insights :

- **Bi-LSTM performed better**: The Bi-LSTM model achieved a notable improvement in accuracy (+16%) and was more balanced in terms of handling both positive and negative sentiment.
- **Naïve Bayes limitations**: The Naïve Bayes model had trouble predicting neutral comments, likely due to its reliance on simpler unigram features and TF-IDF vectorization. It also struggled with short or mixed-language comments.
- **Confidence thresholding helped**: Introducing the confidence thresholding method improved both models, especially for short, vague, or ambiguous comments, reducing incorrect

predictions.

- **Bi-LSTM advantages**: Although the Bi-LSTM model took longer to train and needed more resources (like GPUs), it performed better in handling sarcasm and code-switching, making it a better fit for real-world text.

7.0 Conclusion & Future Work

7.1 Conclusion

This project successfully demonstrated the ability to perform real-time sentiment analysis using an integrated pipeline that combined Apache Kafka, Apache Spark, and Elasticsearch. These modern technologies made it possible to efficiently collect, process, and analyze large volumes of data, such as Reddit comments on movies.

The sentiment analysis pipeline used two models: Multinomial Naive Bayes and Bidirectional LSTM. These models classified comments as positive, negative, or neutral. The results were then visualized in Kibana, providing insights into user opinions on various films discussed across several movie-related subreddits.

Some key takeaways from the analysis include that Reddit comments tended to have a neutral sentiment, with neutral comments being much more frequent than positive or negative ones. Additionally, r/movies and other movie-related subreddits dominated the discussion. The Naïve Bayes model, however, showed a strong tendency to classify most comments as neutral. On the other hand, the Bi-LSTM model performed more evenly, capturing sarcasm and more nuanced sentiment patterns.

In summary, the project demonstrated the potential of using modern data ingestion and processing tools like Apache Kafka and Spark for real-time sentiment analysis.

7.2 Future Work

Although the project was successful in conducting sentiment analysis on Reddit data, there are several areas for improvement and future expansion:

- **Improvement to the Model**:

- **Hybrid Models**: Combining the strengths of Naïve Bayes and Bi-LSTM models could improve the results. By merging traditional and deep learning methods, the system could provide better predictions.

- **Fine-tuning**: More extensive hyperparameter optimization for the Bi-LSTM model, or the use of pre-trained models such as BERT or GPT, could enhance sentiment predictions, especially for more subtle comments.

- **Data Expansion**:

  - The current project focuses on a limited set of Reddit subreddits. Expanding the data to include more subreddits, or even other social media platforms like Twitter, would allow for a more comprehensive analysis of sentiment across different communities.

  - Incorporating comments in other languages, or those with code-switching (like Malay-English), would make the system more adaptable to diverse linguistic contexts.

- **Real-Time Processing Enhancements**:

  - While the current setup uses Apache Kafka and Spark for real-time processing, there is potential to improve scalability. This could include adding partitioning or distributing the load across multiple servers to handle even larger volumes of data.

With these improvements, the sentiment analysis system could become more robust, flexible, and capable of handling a broader range of text data from diverse sources and languages.

8.0 References

1. Gormley, C., & Tong, Z. (2015). *Elasticsearch: The definitive guide*. O'Reilly Media. https://www.elastic.co/guide/en/elasticsearch/
2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735
3. Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the NetDB, 1*(1), 1-8. https://kafka.apache.org/
4. Liu, B. (2012). *Sentiment analysis and opinion mining*. Synthesis Lectures on Human Language Technologies, 5(1), 1-167. https://www.amazon.com/Sentiment-Analysis-Opinion-Mining-Liu/dp/160845836X
5. Rennie, J. D. M., Shih, L., Teevan, J., & Karger, D. (2003). Tackling the poor assumptions of naive Bayes text classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, 616-623. https://dl.acm.org/doi/10.5555/3041838.3041911
6. Viegas, F. B., Wattenberg, M., & Van Ham, F. (2007). How do we see our data? Visualizing the insights of information. In *Proceedings of the 12th International Conference on Information Visualization*, 12(2), 110-116. https://ieeexplore.ieee.org/document/4288244
7. Zaharia, M., Chowdhury, M., Das, T., Dave, A., & Ma, J. (2016). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2(3), 1-16. https://spark.apache.org/

9.0 Appendix

9.1 Scraper code (reddit_scrape.py)

```
import praw
import pandas as pd

# Reddit API credentials
reddit = praw.Reddit(
    client_id="bKUQYyqxMsQwBmECpl8guw",
    client_secret="IIzAvemnMFq22oD3a84-JAqRtveHmQ",
    user_agent="sentiment_project by /u/Legal-Vanilla8068"
)

# List of subreddits
subreddits = ["movies", "TrueFilm", "BoxOffice", "CinemaSins",
"FilmFestivals"]

data = []

for subreddit_name in subreddits:
    subreddit = reddit.subreddit(subreddit_name)
    for submission in subreddit.top(time_filter="year", limit=20):  #
Top posts from last year
        submission.comments.replace_more(limit=0)
        for comment in submission.comments.list():
            data.append({
                "post_id": submission.id,
                "post_title": submission.title,
                "comment_id": comment.id,
                "comment_body": comment.body,
                "comment_score": comment.score,
                "created_utc": comment.created_utc,
                "subreddit": subreddit_name
            })

# Save to CSV
df = pd.DataFrame(data)
df.to_csv("reddit_movies_comments.csv", index=False)
```

```
print(f"Saved {len(df)} comments to reddit_movies_comments.csv")
```

9.2 Cleaner code (CleanerCode.py)

```
import pandas as pd
import re
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords

# Download required NLTK data (first time only)
nltk.download('stopwords')

# Initialize tokenizer and stopwords
tokenizer = RegexpTokenizer(r'\b\w{3,}\b')  # Keep words with 3+ letters
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str) or text.strip() == '':
        return ""

    # Remove URLs
    text = re.sub(r'https?://\S+|www\.\S+', '', text)

    # Remove emails
    text = re.sub(r'\S+@\S+', '', text)

    # Remove emojis and special characters
    text = re.sub(r'[^\w\s#@_/-]', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Normalize whitespace and lowercase
    text = re.sub(r'\s+', ' ', text).strip().lower()

    # Tokenize
```

```
    tokens = tokenizer.tokenize(text)

    # Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Convert UTC timestamp to readable format
def utc_to_readable(utc_time):
    try:
        return datetime.utcfromtimestamp(float(utc_time)).strftime('%Y-%m-%d %H:%M:%S')
    except:
        return ""

# Load your dataset
print("Loading dataset...")
df = pd.read_csv("reddit_movies_comments_large.csv")

# Convert UTC timestamp
print("Converting timestamps...")
df['timestamp_readable'] = df['created_utc'].apply(utc_to_readable)

# Clean comment body
print("Cleaning comment bodies...")
df['cleaned_comment'] = df['comment_body'].apply(clean_text)

# Filter out empty comments
df = df[df['cleaned_comment'].str.len() > 5]

# Reorder and select final columns
final_columns = [
    'comment_id', 'post_title', 'comment_body', 'cleaned_comment',
    'comment_score', 'created_utc', 'timestamp_readable', 'subreddit'
]
df = df[final_columns]

# Save cleaned dataset
output_file = "reddit_cleaned_with_stopwords_removed.csv"
```

```
df.to_csv(output_file, index=False)
print(f"✅ Cleaned dataset saved to {output_file}")
```

9.3 Docker container (docker-compose.yml)

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.6.1
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka:7.6.1
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_LISTENERS:
PLAINTEXT://0.0.0.0:29092,PLAINTEXT_HOST://0.0.0.0:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.13.4
    environment:
```

```
      - discovery.type=single-node
      - xpack.security.enabled=false
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms1g -Xmx1g"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
      - "9200:9200"
      - "9300:9300"

  kibana:
    image: docker.elastic.co/kibana/kibana:8.13.4
    depends_on:
      - elasticsearch
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200

  spark:
    image: bitnami/spark:3.4.2
    depends_on:
      - kafka
      - elasticsearch
    ports:
      - "8080:8080"  # Spark Master UI
      - "7077:7077"  # Spark Master Port
    volumes:
      - ./:/opt/bitnami/spark/work # Maps your project directory into
the container
    environment:
      SPARK_MODE: master
      SPARK_MASTER_URL: spark://spark:7077

   # --- ADD THIS NEW SERVICE ---
  spark-worker:
    image: bitnami/spark:3.4.2
```

```
    depends_on:
      - spark # Ensures the master is started first
    volumes:
      - ./:/opt/bitnami/spark/work
    environment:
      SPARK_MODE: worker
      SPARK_MASTER_URL: spark://spark:7077 # Tells the worker where to
find the master
      SPARK_WORKER_CORES: 2  # Number of cores the worker can use
      SPARK_WORKER_MEMORY: 2g # Amount of memory the worker can use
```

9.4 Kafka (kafka_producer.py)

```
import os
import time
import json
import praw
from kafka import KafkaProducer
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# --- Kafka Configuration ---
KAFKA_TOPIC = 'reddit-comments'
KAFKA_BOOTSTRAP_SERVERS = 'localhost:9092'

# --- Reddit API Configuration ---
SUBREDDIT_NAME = 'movies'  # The subreddit you want to stream

def create_kafka_producer():
    """Creates a Kafka producer instance."""
    try:
        producer = KafkaProducer(
```

```
            bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )
        print("✅ Kafka Producer connected successfully.")
        return producer
    except Exception as e:
        print(f"❌ Could not connect to Kafka: {e}")
        return None

def connect_to_reddit():
    """Connects to the Reddit API using credentials from environment
variables."""
    try:
        reddit = praw.Reddit(
            client_id=os.getenv("REDDIT_CLIENT_ID"),
            client_secret=os.getenv("REDDIT_CLIENT_SECRET"),
            user_agent=os.getenv("REDDIT_USER_AGENT")
        )
        print("✅ Reddit API connected successfully.")
        return reddit
    except Exception as e:
        print(f"❌ Could not connect to Reddit: {e}")
        return None

def stream_comments(producer, reddit):
    """Streams comments from the specified subreddit and sends them to
Kafka."""
    if not producer or not reddit:
        return

    subreddit = reddit.subreddit(SUBREDDIT_NAME)
    print(f"📡 Streaming comments from r/{SUBREDDIT_NAME}...")

    try:
        for comment in subreddit.stream.comments(skip_existing=True):
            try:
                # Construct the message payload
                message = {
                    "post_id": comment.submission.id,
```

```
                    "post_title": comment.submission.title,
                    "comment_id": comment.id,
                    "comment_body": comment.body,
                    "comment_score": comment.score,
                    "created_utc": comment.created_utc
                }

                # Send the message to our Kafka topic
                producer.send(KAFKA_TOPIC, message)
                print(f"📤 Sent comment ID {comment.id}:
{comment.body[:50]}...")

                time.sleep(1)  # Throttle to avoid overwhelming Kafka

            except Exception as e:
                print(f"⚠️ Error processing comment {comment.id}: {e}")
                continue # Skip to the next comment

    except Exception as e:
        print(f"❌ An error occurred during streaming: {e}")

if __name__ == "__main__":
    kafka_producer = create_kafka_producer()
    reddit_client = connect_to_reddit()

    if kafka_producer and reddit_client:
        stream_comments(kafka_producer, reddit_client)
```

9.5 Spark (spark_streaming.py)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # Suppress TensorFlow INFO and
WARNING messages

from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col, from_json
```

```
from pyspark.sql.types import StringType, StructType, StructField,
IntegerType, FloatType
import re
import joblib
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences

# --- Configuration ---
KAFKA_TOPIC = "reddit-comments"
KAFKA_SERVER = "kafka:29092"
ELASTICSEARCH_NODE = "elasticsearch"
ELASTICSEARCH_PORT = "9200"
ELASTICSEARCH_INDEX = "reddit-comments-lstm"
CHECKPOINT_LOCATION = "/tmp/spark_checkpoint_reddit_lstm"

# Define a maximum sequence length for padding.
# This should match the length used during model training.
MAX_SEQUENCE_LENGTH = 150

# --- Model Loading and Broadcasting ---
# Load the tokenizer and the Keras model on the driver
tokenizer = joblib.load("/opt/bitnami/spark/work/models/tokenizer.pkl")
model =
tf.keras.models.load_model("/opt/bitnami/spark/work/models/lstm_sentimen
t_model.h5")

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("RedditSentimentLSTM") \
    .config("spark.jars.packages",
            "org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.2,"
            "org.elasticsearch:elasticsearch-spark-30_2.12:8.13.4") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

# Broadcast the tokenizer and the model's weights to all worker nodes
tokenizer_broadcast = spark.sparkContext.broadcast(tokenizer)
```

```
model_weights_broadcast =
spark.sparkContext.broadcast(model.get_weights())

# --- Schema Definition for Kafka Messages ---
schema = StructType([
    StructField("post_id", StringType(), True),
    StructField("post_title", StringType(), True),
    StructField("comment_id", StringType(), True),
    StructField("comment_body", StringType(), True),
    StructField("comment_score", IntegerType(), True),
    StructField("created_utc", FloatType(), True)
])

# --- Text Cleaning Function ---
def clean_text(text):
    if text is None:
        return ""
    text = str(text).lower()
    text = re.sub(r"https[^\s]+|www[^\s]+", '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)       # Remove special
characters/emojis
    text = re.sub(r'\s+', ' ', text).strip()         # Normalize
whitespace
    return text

clean_text_udf = udf(clean_text, StringType())


# --- Sentiment Prediction UDF for LSTM Model ---
def predict_sentiment_lstm(text_series):
    # This function is designed to work with Pandas UDFs for efficiency
    # but a standard UDF is used here for simplicity.

    # 1. Get the broadcasted tokenizer and weights
    local_tokenizer = tokenizer_broadcast.value
    local_weights = model_weights_broadcast.value

    # 2. Re-create the model architecture and set the broadcasted
weights
```

```
    #    This avoids serializing the entire model object.
    local_model =
tf.keras.models.load_model("/opt/bitnami/spark/work/models/lstm_sentimen
t_model.h5")
    local_model.set_weights(local_weights)

    predictions = []
    for text in text_series:
        if not text or len(text.strip()) < 2:
            predictions.append("NEUTRAL")
            continue
        try:
            # 3. Tokenize and pad the text
            sequence = local_tokenizer.texts_to_sequences([text])
            padded_sequence = pad_sequences(sequence,
maxlen=MAX_SEQUENCE_LENGTH)

            # 4. Make a prediction
            prediction_prob = local_model.predict(padded_sequence,
verbose=0)[0][0]

            # 5. Interpret the prediction
            if prediction_prob > 0.5:
                predictions.append("POSITIVE")
            else:
                predictions.append("NEGATIVE")
        except Exception as e:
            print(f"Prediction error: {e}")
            predictions.append("ERROR")

    return predictions

# We need to wrap the function slightly for a standard UDF
def predict_sentiment(text):
    return predict_sentiment_lstm([text])[0]

predict_udf = udf(predict_sentiment, StringType())
```

```
# --- Spark Streaming Pipeline ---
print("🚀 Reading from Kafka...")
# 1. Read from Kafka
raw_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_SERVER) \
    .option("subscribe", KAFKA_TOPIC) \
    .option("startingOffsets", "latest") \
    .load()

# 2. Decode the Kafka message and parse the JSON
json_df = raw_df.selectExpr("CAST(value AS STRING)")
parsed_df = json_df.select(from_json(col("value"),
schema).alias("data")).select("data.*")

# Add this to see the raw JSON from Kafka in your console
json_df.writeStream \
    .outputMode("append") \
    .format("console") \
    .start()

parsed_df = json_df.select(from_json(col("value"),
schema).alias("data")).select("data.*")

# 3. Clean the comment text
print("🚀 Cleaning comments...")
clean_df = parsed_df.withColumn("clean_comment",
clean_text_udf(col("comment_body")))
clean_df.writeStream \
    .outputMode("append") \
    .format("console") \
    .start()

print("🚀 Streaming to console started...")

# 4. Predict sentiment using the LSTM model
print("🚀 Predicting sentiment...")
result_df = clean_df.withColumn("sentiment",
predict_udf(col("clean_comment")))
```

```
print("🚀 Starting Spark writeStream to Elasticsearch...")
# --- Write to Elasticsearch ---
es_query = result_df.writeStream \
    .outputMode("append") \
    .format("org.elasticsearch.spark.sql") \
    .option("es.nodes", ELASTICSEARCH_NODE) \
    .option("es.port", ELASTICSEARCH_PORT) \
    .option("checkpointLocation", CHECKPOINT_LOCATION) \
    .option("es.resource", ELASTICSEARCH_INDEX) \
    .start()

print("✅ writeStream started. Awaiting termination...")

es_query.awaitTermination()
```

final_report.pdf

ORIGINALITY REPORT

17%	14%	9%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dev.to Internet Source	1%
2	Submitted to Gisma University of Applied Sciences GmbH Student Paper	1%
3	medium.com Internet Source	1%
4	Submitted to University of North Texas Student Paper	1%
5	Submitted to CSU, San Jose State University Student Paper	1%
6	Submitted to (school name not available) Student Paper	1%
7	Submitted to Coventry University Student Paper	1%
8	github.com Internet Source	<1%
9	liashchynskyi.net Internet Source	<1%
10	www.jetir.org Internet Source	<1%

11	Submitted to University of Bristol Student Paper	<1%
12	raga.ai Internet Source	<1%
13	stackoverflow.com Internet Source	<1%
14	Submitted to University of Sunderland Student Paper	<1%
15	www.coursehero.com Internet Source	<1%
16	www.dremio.com Internet Source	<1%
17	Submitted to University of New South Wales Student Paper	<1%
18	Submitted to UT, Dallas Student Paper	<1%
19	essay.utwente.nl Internet Source	<1%
20	Submitted to Harrisburg University of Science and Technology Student Paper	<1%
21	madonnadellerose.it Internet Source	<1%
22	Submitted to CSU, Dominguez Hills Student Paper	<1%

23	forge.citizen4.eu Internet Source	<1%
24	helx-artifacts-git.apps.renci.org Internet Source	<1%
25	research-api.cbs.dk Internet Source	<1%
26	techblog.smc.it Internet Source	<1%
27	Submitted to Instituto de Empress S.L. Student Paper	<1%
28	Submitted to The London Interdisciplinary School - LIS Student Paper	<1%
29	huggingface.co Internet Source	<1%
30	Submitted to Georgetown University Student Paper	<1%
31	Submitted to Sydney Polytechnic Institute Student Paper	<1%
32	Submitted to University of East London Student Paper	<1%
33	Submitted to Monash University Student Paper	<1%
34	Submitted to University of Wales, Bangor Student Paper	<1%
	www.geeksforgeeks.org	

35	Internet Source	<1%
36	Submitted to New Zealand School of Education Student Paper	<1%
37	Submitted to University of Bradford Student Paper	<1%
38	www.grin.com Internet Source	<1%
39	Submitted to City University of Seattle Student Paper	<1%
40	Kushagra Wadhwa, Devansh Mehra, Himnish Gosain, A. K. Haritash. "Sentiment Analysis of Air Quality Perception in Major Metro Cities of India", 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2023 Publication	<1%
41	Submitted to University of South Australia Student Paper	<1%
42	ebin.pub Internet Source	<1%
43	link.springer.com Internet Source	<1%
44	Submitted to Liverpool John Moores University Student Paper	<1%

45	pythonprogramming.net Internet Source	<1%
46	Bowen, Matthew Booth. "Leveraging Latent Textual Topology for Standard Identification in Engineering Design", University of Georgia, 2024 Publication	<1%
47	Charlotte Nirmalani Gunawardena, Nick V. Flor, Damien M. Sánchez. "Knowledge Co-Construction in Online Learning - Applying Social Learning Analytic Methods and Artificial Intelligence", Routledge, 2025 Publication	<1%
48	Daniel Maitethia Memeu. "Automated malaria diagnosis and parasitemia estimation using a customized OpenFlexure microscope", Springer Science and Business Media LLC, 2025 Publication	<1%
49	jamiescience.com Internet Source	<1%
50	www.analyticsvidhya.com Internet Source	<1%
51	www.irejournals.com Internet Source	<1%
52	bookdown.org Internet Source	<1%
	hdl.handle.net	

53	Internet Source	<1%
54	machinelearningmastery.com Internet Source	<1%
55	Girona, Antonio E.. "Navigating Trust, Privacy, and Mental Health: Remote Workers' Experiences During the COVID-19 Pandemic—A Textual Analysis of Reddit Data.", The Pennsylvania State University Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		