SECP3133-02

HIGH PERFORMANCE DATA PROCESSING

Real-Time Sentiment Analysis using Apache Spark and Kafka

Prepared By Group 7:

VINESH A/L VIJAYA KUMAR A22EC0290

JOSEPH LAU YEO KAI A22EC0055

TIEW CHUAN SHEN A22EC0113

NUR FARAH ADIBAH BINTI IDRIS A22EC0245

Lecturer Name:

DR. ARYATI BINTI BAKRI

Section:

02

Date of Submission:

11/7/2025

Table of Contents

1. Introduction

This section explains the background,objective and scope of the project. It provides the overview of the challenges in the real world on handling large scale, unstructured social media data and justification is made on the need of sentiment analysis automation. The objective and the scope are mentioned to plan for the whole project workflow.

1.1 Background of the project

In this digital era, large amounts of content generated by users from all over the world are shared on social media platforms, e-commerce sites and news portals. Opinions, emotions and sentiments from the public are reflected through these platforms, on products, services, events and social issues. Internet and media usage of Malaysians are highest among the Southeast Asia region, which emphasizes the needs on public sentiment analysis, to obtain the insights on business, policy and organizations in order to understand and respond to public concerns in an effective method.

One of the main issues is that traditional sentiment analysis relies on batch processing, which is unable to obtain fast changing public opinions in a short time, especially during elections, viral controversies or product launches. Therefore, the need for real time sentiment analysis systems is increased, for stakeholders such as government and company to make updated and informed decisions.

Big data and real-time streaming technologies are addressed in the project, and tools such as Apache Kafka, Apache Spark and ElasticSearch are implemented. These tools are integrated with natural language processing(NLP) and machine learning techniques, in order to build a scalable, efficient inclusive sentiment analysis pipeline which fits Malaysian-relevant data sources.

1.2 Objectives

The primary objectives of this project are:

1. To collect a dataset of user posts and comments from Malaysia social media, Reddit.
2. To apply standard natural language processing(NLP) towards dataset collected from Reddit, to produce cleaned data for the machine learning phase.
3. To design, build, train and evaluate at least two sentiment classification models, for making comparisons on their effectiveness, and to choose the best model for the next phase.
4. To build a real time streaming pipeline in Docker using Apache Kafka, Apache Spark, for applying trained sentiment models.
5. To create sentiment analysis visualization for key insights and trends.
6. To analyze the performance of the entire sentiment analysis system.

1.3 Project Scope

The scope of this project includes the following components:

1. **Data Collection:** Text data is collected primarily from the r/malaysiafood, r/malaysians and /malaysiauni subreddit, which focuses on content on food reviews, malaysia current issues and news and university content. The dataset includes post and comments in mainly English.
2. **Containerization:** Docker container is implemented in this project for sentiment analysis. For real time processing pipeline and visualization, components in these processes are containerized using Docker for isolation and easy implementation by other teammates. Docker Compose is used for multi-container orchestration.
3. **Data Preprocessing:** Standard NLP techniques are applied, which lowercasing, URL and emoji removal, punctuation and digit removal, tokenization, stop word removal and lemmatization are implemented. Both batch and streaming pipelines are implemented to ensure consistency.
4. **Machine Learning Models-** Two models are trained and deployed: Naive Bayes and Logistic Regression. Feature extraction is performed using tokenization, stop word removal, Hashing TF-IDF. Models trained are then compared and best model is chosen for pipeline.
5. **Visualization:** Sentiment results are visualized through interactive ElasticSearch dashboards, which real time and historical insights are provided for the stakeholders. Dashboards are used to display sentiment trends, keyword frequencies, and public opinion changes over time.
6. **Real-Time Pipeline Implementation:**
    - Use **Apache Kafka** for real-time data ingestion.
    - Process the data stream using **Apache Spark Streaming**.
    - Apply the trained sentiment models to classify sentiments in real time.
7. **Performance Evaluation:**
    - Compare the accuracy, precision, recall, and processing latency of the models.
8. **Documentation and Reporting:**
    - Prepare comprehensive project documentation, dashboards, and a final presentation summarizing key insights and technical achievements.

2. Data Acquisition & Preprocessing

This section describes how raw data was obtained and transformed into a usable and clean format for model training.

2.1 Sources

For this sentiment analysis project, two datasets were utilised, which are the Sentiment140.csv dataset retrieved from Kaggle and the Reddit comments dataset. The table below describes both datasets.

Table 2.1 Data Sources

Dataset	**Description**	**Acquisition**
Twitter Sentiment140 dataset	This dataset contains 1.6 million tweets extracted using a Twitter API. The name sentiment140 came from the fact that negative is labelled as 0, neutral as 1 and positive as 4.	https://www.kaggle.com/datasets/kazanova/sentiment140 We download the dataset from the above link and place it in the subfolder data\raw_data of this project folder.
Reddit comments dataset	This dataset contains unlabeled Reddit comments.	We retrieve several JSONL files from several Malaysian subreddits using a Kafka producer(reddit_raw_producer.py) and consumer(reddit_raw_consumer.py), and place the retrieved data in the data/raw_data folder.

2.2 Tools

For acquisition and data preprocessing, we use Python with various libraries and other technologies:

- **pandas**: Data loading, manipulation and combining datasets.
- **json**: Parsing JSONL formatted Reddit data.
- **re**: For regular expression-based pattern matching and text cleaning.
- **os**: Path management and file system operations.
- **NLTK**: Various text preprocessing, such as **nltk.corpus.stopwords** for stopword remover, **nltk.stem.WordNetLemmatizer** for reducing words to their base form and **nltk.tokenize.word_tokenize** for breaking down text into individual words.
- **praw:** Python Reddit API wrapper
- **Apache Kafka:** Extract Reddit data from the Reddit API

2.3 Cleaning Steps

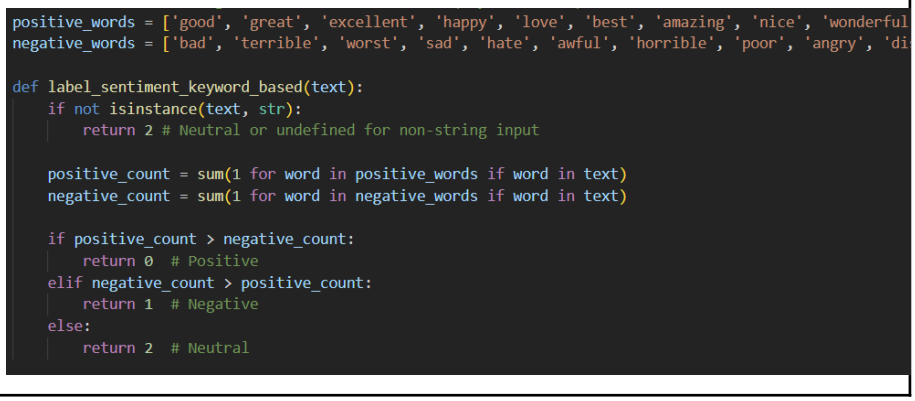
The raw dataset in data\raw_data goes through a series of preprocessing steps to ensure high-quality, consistent and suitable data for sentiment analysis in **preprocessing.ipynb** code. The table below describes all the steps implemented in the preprocessing.ipynb.

Table 2.3 Data Preprocessing

Steps	**Description / Code Snippets**
Lowercase conversion	All text was converted to lowercase to standardise words and reduce vocabulary size. `text = text.lower()`

URL and mention removal	Remove URLs using regular expressions as it do not carry sentiment value and remove user mentions.
	
	`text = re.sub(r'http\S+|www\S+|https\S+', '', text`
	`text = re.sub(r'@\w+', '', text)`

Hashtag symbol removal	Remove # symbol, but the hashtag word was retained as the word carries sentiment. `text = re.sub(r'#(\w+)', r'\1', text)`
Special character and Number removal	Non-alphabetic characters, such as punctuation and numbers, are removed. `text = re.sub(r'[^a-z\s]', '', text)`
Whitespace consolidation	Multiple spaces were replaced with a single space to ensure clean tokenisation. `text = re.sub(r'\s+', ' ', text).strip()`
Tokenization	The cleaned text was tokenised into individual words using NLTK's word_tokenizer. `words = word_tokenize(text)`
Stop word removal	Filtered out common English stopwords using NLTK's predefined stop word list. `stop_words = set(stopwords.words('english'))`
Lemmatization	Converting words to their base or dictionary form using the WordNetLemmatizer library. `cleaned_words = [lemmatizer.lemmatize(word) for`

	`word in words if word not in stop_words]`
Sentiment Labelling	The extracted Reddit data are unlabeled, and we label them into positive, negative and neutral using keyword-based method. 

```python
positive_words = ['good', 'great', 'excellent', 'happy', 'love', 'best', 'amazing', 'nice', 'wonderful
negative_words = ['bad', 'terrible', 'worst', 'sad', 'hate', 'awful', 'horrible', 'poor', 'angry', 'di

def label_sentiment_keyword_based(text):
    if not isinstance(text, str):
        return 2 # Neutral or undefined for non-string input

    positive_count = sum(1 for word in positive_words if word in text)
    negative_count = sum(1 for word in negative_words if word in text)

    if positive_count > negative_count:
        return 0  # Positive
    elif negative_count > positive_count:
        return 1  # Negative
    else:
        return 2  # Neutral
```

3. Sentiment Model Development

This section describes sentiment model development from model choice, training process and up to model evaluation.

3.1 Model Choice

For this project, we train two models using raw data acquired from Kaggle and Reddit. This model aims to be used in a Spark Kafka streaming workflow to predict streaming Reddit comments as neutral, positive or negative.

3.2 Training Process

model_training.ipynb: Model training and evaluation (Logistic Regression, Naive Bayes, etc.)

2_sentiment_classification.ipynb: Sentiment classification experiments

Training Process Description

The notebook outlines the development and training of two sentiment classification models: a Multinomial Naive Bayes model and a Logistic Regression model. Below is a breakdown of the actual training process implemented:

Data Preprocessing

For both models:

Text data was cleaned through lowercasing, punctuation, special characters, and stop words were removed. Tokenization was performed to split the text into words.

Naive Bayes used a TF-IDF Vectorizer to convert the cleaned text into numerical features. Split data into training and testing sets using train_test_split().

Model Training

Multinomial Naive Bayes created an instance of MultinomialNB and trained on the TF-IDF vectorized training data and labels using fit().

```
# Import necessary Spark and MLlib libraries

from pyspark.sql import SparkSession

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

```
from pyspark.sql.types import StructType, StructField, IntegerType,
StringType

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml.linalg import Vectors, Vector

from pyspark.ml.feature import Tokenizer, HashingTF, IDF,
StopWordsRemover

from pyspark.ml import Pipeline
```

Figure 3.2.1: Import necessary Spark and MLib libraries

This code cell imports the necessary libraries from PySpark for building machine learning pipelines, specifically for classification tasks. It includes SparkSession for initializing Spark, functions for data manipulation (col, udf), data types (StructType, StructField, IntegerType, StringType), and components for building the machine learning pipeline like LogisticRegression, MulticlassClassificationEvaluator, feature transformers (Tokenizer, HashingTF, IDF, StopWordsRemover), and the Pipeline class itself.

```
# Initialize SparkSession

# The appName is a label for your application in the Spark UI

# The .getOrCreate() method ensures that if a SparkSession already
exists, it will be used,

# otherwise, a new one will be created.

spark = SparkSession.builder \

    .appName("PySparkModelTraining") \

    .getOrCreate()



# Set log level to ERROR to reduce verbosity for cleaner output

spark.sparkContext.setLogLevel("ERROR")



print("SparkSession initialized successfully.")
```

Figure 3.2.2: Initialize Spark Session

This code initializes a SparkSession. A SparkSession is the entry point to programming Spark with the Dataset and DataFrame API.

SparkSession.builder: Creates a builder for SparkSession.
.appName("PySparkModelTraining"): Sets an application name, which can be helpful for monitoring in the Spark UI.
.getOrCreate(): This is a crucial part. It checks if a SparkSession instance already exists. If it does, it reuses it; otherwise, it creates a new one. This prevents issues with multiple Spark sessions running simultaneously in the same environment.
spark.sparkContext.setLogLevel("ERROR"): This line sets the logging level for the SparkContext to "ERROR". This is done to reduce the amount of output printed to the console during execution, showing only error messages and suppressing less important information like INFO or WARN messages.
print("SparkSession initialized successfully."): Confirms that the SparkSession has been set up.

```
# --- 1. Load Cleaned Data ---

# Define path for cleaned data (CSV format)
cleaned_data_path = "../data/cleaned_data.csv"

try:
    # --- FIX: Define explicit schema for reading CSV ---
    # This ensures Spark reads the columns with the correct data types,
    # preventing misinterpretations by inferSchema=True.
    cleaned_data_schema = StructType([
        StructField("content", StringType(), True),
        StructField("clean_comment", StringType(), True),
        StructField("sentiment", IntegerType(), True) # Explicitly
define sentiment as IntegerType
    ])

    # Load the CSV file using the defined schema
    df_cleaned = spark.read.csv(cleaned_data_path, header=True,
schema=cleaned_data_schema)
    print(f"\nCleaned data loaded successfully from
{cleaned_data_path}")

    # Rename 'sentiment' column to 'label' for MLlib compatibility
    df_cleaned = df_cleaned.withColumnRenamed("sentiment", "label")

    # --- ADDED: Re-filter out empty clean_comment rows in Spark ---
    # This ensures consistency with the preprocessing script's
filtering.
```

```
    initial_spark_rows = df_cleaned.count() # Count before filtering
    df_cleaned = df_cleaned.filter(col("clean_comment").isNotNull() &
(col("clean_comment") != ""))
    rows_filtered_in_spark = initial_spark_rows - df_cleaned.count()
    if rows_filtered_in_spark > 0:
        print(f"Filtered {rows_filtered_in_spark} rows with empty or
null 'clean_comment' after loading in Spark.")

    # --- ADDED FIX: Filter out rows with null labels ---
    initial_rows_before_label_filter = df_cleaned.count()
    df_cleaned = df_cleaned.filter(col("label").isNotNull())
    rows_filtered_for_null_labels = initial_rows_before_label_filter -
df_cleaned.count()
    if rows_filtered_for_null_labels > 0:
        print(f"Filtered {rows_filtered_for_null_labels} rows with NULL
'label' values.")

    print("\n--- Cleaned Data Sample (first 5 rows) ---")
    df_cleaned.show(5, truncate=False)
    df_cleaned.printSchema()
    print(f"Total rows in cleaned DataFrame after Spark-side filtering:
{df_cleaned.count()}")
except Exception as e:
    print(f"Error loading cleaned data: {e}")
    print("Please ensure the preprocessing step ran successfully and
'cleaned_data.csv' was saved at the specified path.")
    # If loading fails, create a dummy DataFrame to avoid breaking the
notebook flow
    dummy_schema = StructType([
        StructField("content", StringType(), True),
        StructField("clean_comment", StringType(), True),
        StructField("label", IntegerType(), True)
    ])
    df_cleaned = spark.createDataFrame([
        ("This is a positive comment.", "positive comment", 0),
        ("This is a negative comment.", "negative comment", 1),
        ("This is a neutral comment.", "neutral comment", 2)
    ], schema=dummy_schema)
    print("Loaded dummy cleaned data for demonstration due to file
loading issue.")
```

Figure 3.2.3: Load Dataset and Data Cleaning

This code cell is responsible for loading the cleaned data into a Spark DataFrame and performing some initial data cleaning steps.

1. cleaned_data_path = "../data/cleaned_data.csv": Defines the file path where the cleaned data is expected to be found.
2. try...except block: This is used to handle potential errors during the file loading process. If the file is not found or there's another issue, it catches the exception and prints an error message. It also creates a small dummy DataFrame to allow the rest of the notebook to run without crashing, which is helpful for demonstration or debugging.
3. cleaned_data_schema = StructType([...]): Explicitly defines the schema (column names and data types) for the CSV data. This is a good practice in Spark to ensure data is loaded correctly and avoids potential issues with Spark inferring the schema, especially for data types like integers.
4. df_cleaned = spark.read.csv(...): Reads the CSV file into a Spark DataFrame.
    - header=True: Indicates that the first row of the CSV is the header.
    - schema=cleaned_data_schema: Applies the explicitly defined schema.
5. df_cleaned = df_cleaned.withColumnRenamed("sentiment", "label"): Renames the "sentiment" column to "label". This is done because MLlib (Spark's machine learning library) typically expects the target variable column to be named "label".
6. df_cleaned = df_cleaned.filter(...): Filters the DataFrame to remove rows where the clean_comment column is null or empty. This ensures that only valid text data is used for feature engineering and model training. It also prints how many rows were filtered out.
7. df_cleaned = df_cleaned.filter(col("label").isNotNull()): Filters out rows where the label column is null. This is essential as machine learning models require valid labels for training. It also reports the number of rows removed.
8. df_cleaned.show(5, truncate=False): Displays the first 5 rows of the cleaned DataFrame after loading and initial filtering, showing all content without truncation.
9. df_cleaned.printSchema(): Prints the schema of the DataFrame, confirming the column names and data types.
10. df_cleaned.count(): Prints the total number of rows remaining in the DataFrame after all filtering steps.

In summary, this cell loads the cleaned text data, defines its structure, renames the label column for compatibility, removes any rows with missing text or labels, and provides a preview of the resulting DataFrame.

```
# --- 2. Text Feature Engineering (TF-IDF) ---

# Ensure 'clean_comment' column is not null and is string type
# This line is redundant if the above filter is effective, but
harmless.
```

```
df_cleaned = df_cleaned.withColumn("clean_comment",
col("clean_comment").cast(StringType()))

# 2.1 Tokenization: Split text into words
tokenizer = Tokenizer(inputCol="clean_comment", outputCol="words")

# 2.2 Stop Words Removal (Optional, as clean_text already handles this,
but can be used for extra robustness)
# For this dataset, English stopwords are primarily removed by the
preprocessing script.
# If you uncomment, make sure 'words' column exists from tokenizer.
# remover = StopWordsRemover(inputCol=tokenizer.getOutputCol(),
outputCol="filtered_words")

# 2.3 HashingTF: Convert words into fixed-size feature vectors (term
frequencies)
# numFeatures is the size of the vocabulary (or feature vector
dimension).
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
outputCol="raw_features", numFeatures=10000)

# 2.4 IDF: Calculate Inverse Document Frequency
idf = IDF(inputCol=hashingTF.getOutputCol(), outputCol="features")

# 2.5 Create a Pipeline to chain these steps
pipeline = Pipeline(stages=[tokenizer, hashingTF, idf])

# Fit the pipeline to the data and transform it to get the 'features'
column
print("\n--- Building Feature Engineering Pipeline and Transforming
Data ---")
pipeline_model = pipeline.fit(df_cleaned)
df_features = pipeline_model.transform(df_cleaned)

print("Feature engineering completed. Sample of features and labels:")
df_features.select("clean_comment", "features", "label").show(5,
truncate=False)
df_features.printSchema()

# --- ADDED: Check for empty/null feature vectors after TF-IDF ---
print("\n--- Checking Feature Vector Validity ---")
# Count rows where 'features' column is null or has zero size (empty
vector)
```

```
# For SparseVector, size is the number of non-zero elements.
# A vector with no non-zero elements (i.e., all zeros) can be
problematic for some models.
# Check for null features first
null_features_count =
df_features.filter(col("features").isNull()).count()
if null_features_count > 0:
    print(f"WARNING: Found {null_features_count} rows with NULL feature
vectors. These will be filtered out.")
    df_features = df_features.filter(col("features").isNotNull())

# Re-count after any potential filtering for null features
print(f"Total rows in DataFrame after feature engineering and null
feature filtering: {df_features.count()}")
```

Figure 3.2.4: Text Feature Engineering

This code cell performs the crucial step of text feature engineering using TF-IDF (Term Frequency-Inverse Document Frequency), which converts the text data into numerical feature vectors that can be used by machine learning models.

1. df_cleaned = df_cleaned.withColumn(...): This line ensures the clean_comment column is treated as a StringType. While the previous cell's schema definition should handle this, this line adds robustness.
2. tokenizer = Tokenizer(...): This initializes a Tokenizer. It splits the text in the clean_comment column into individual words (tokens) and puts them into a new column called words.
3. hashingTF = HashingTF(...): This initializes a HashingTF transformer. It takes the words column and converts each word into a numerical feature vector using the hashing trick.
    - numFeatures=10000: Sets the size of the feature vector to 10,000. This is the vocabulary size.
    - outputCol="raw_features": Names the output column containing the term frequency vectors.
4. idf = IDF(...): This initializes an IDF (Inverse Document Frequency) transformer. It takes the raw_features (term frequency vectors) and scales them down based on how frequently a term appears across all documents. This helps to give more weight to rarer, more informative terms.
    - outputCol="features": Names the final output column containing the TF-IDF feature vectors.
5. pipeline = Pipeline(stages=[...]): This creates a Pipeline. A pipeline chains multiple transformers (like Tokenizer, HashingTF, IDF) and estimators together. This allows you to treat the sequence of feature engineering steps as a single process.

6. pipeline_model = pipeline.fit(df_cleaned): This fits the pipeline to the data. During fitting, the IDF transformer calculates the inverse document frequencies based on the entire dataset.
7. df_features = pipeline_model.transform(df_cleaned): This transforms the original DataFrame (df_cleaned) using the fitted pipeline. This applies the tokenization, HashingTF, and IDF steps sequentially, adding the words, raw_features, and features columns to the DataFrame.
8. df_features.select(...).show(...): Displays a sample of the resulting DataFrame, showing the original clean_comment, the generated features vector, and the label.
9. df_features.printSchema(): Prints the schema of the DataFrame, showing the newly added columns.
10. Checking for empty/null feature vectors: This section includes a check to see if any rows resulted in null feature vectors after the transformation. While the previous filtering of empty comments should prevent truly empty vectors, this adds an extra layer of validation.

In essence, this cell transforms raw text comments into dense numerical vectors using the TF-IDF technique, making the data ready for machine learning models.

```
# --- 3. Data Splitting and Model Training ---

# 3.1 Data Splitting
# Split the data with features and labels into training and testing
sets.
train_df, test_df = df_features.randomSplit([0.7, 0.3], seed=42)

print(f"\nTraining data rows: {train_df.count()}")
print(f"Test data rows: {test_df.count()}")

# 3.2 Model Definition: Multi-class Logistic Regression
lr = LogisticRegression(featuresCol="features", labelCol="label",
family="multinomial")

# 3.3 Model Training
print("\n--- Starting Multi-class Logistic Regression Model Training
---")
lr_model = lr.fit(train_df)
print("Model training completed.")
```

Figure 3.2.5: Data Splitting and Model Training

This code cell performs two main actions: splitting the data into training and testing sets and then training a Logistic Regression model on the training data.

1. train_df, test_df = df_features.randomSplit([0.7, 0.3], seed=42): This line splits the df_features DataFrame (which contains the TF-IDF features and labels) into two DataFrames: train_df and test_df.
    - [0.7, 0.3]: Specifies the proportions for the split. 70% of the data will go into the training set, and 30% will go into the testing set.
    - seed=42: Sets a random seed. This ensures that the split is reproducible, meaning you will get the same train and test sets each time you run this code with the same data.
2. print(f"..."): These lines print the number of rows in the training and testing DataFrames to confirm the split was performed.
3. lr = LogisticRegression(...): This initializes a LogisticRegression model.
    - featuresCol="features": Tells the model to use the column named "features" as the input features.
    - labelCol="label": Tells the model to use the column named "label" as the target variable.
    - family="multinomial": Specifies that this is a multi-class classification problem (more than two classes). Logistic Regression can be binomial (for two classes) or multinomial (for more than two).
4. lr_model = lr.fit(train_df): This is the model training step. The fit() method trains the initialized LogisticRegression model using the train_df DataFrame. The result is a LogisticRegressionModel object (lr_model) which contains the learned coefficients and intercept.
5. print("Model training completed."): Confirms that the training process has finished.

In summary, this cell prepares the data for modeling by splitting it and then trains the Logistic Regression model using the training portion of the data.

```
# --- 4. Make Predictions and Model Evaluation ---

# 4.1 Make Predictions
predictions = lr_model.transform(test_df)

print("\n--- Predictions Sample (first 10 rows) ---")
predictions.select("label", "prediction", "probability",
"rawPrediction").show(10, truncate=False)

# 4.2 Model Evaluation (Multi-class)
evaluator_accuracy =
MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="f1")
```

```
evaluator_weighted_precision =
MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedPrecision")
evaluator_weighted_recall =
MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedRecall")

accuracy = evaluator_accuracy.evaluate(predictions)
f1_score = evaluator_f1.evaluate(predictions)
weighted_precision = evaluator_weighted_precision.evaluate(predictions)
weighted_recall = evaluator_weighted_recall.evaluate(predictions)

print(f"\nAccuracy on test set: {accuracy}")
print(f"F1 Score on test set: {f1_score}")
print(f"Weighted Precision on test set: {weighted_precision}")
print(f"Weighted Recall on test set: {weighted_recall}")

# 4.3 Save Trained Model
model_save_path = "../data/trained_model/lr"
print(f"\nSaving trained Logistic Regression model to
{model_save_path}...")
try:
    lr_model.save(model_save_path)
    print("Model saved successfully.")
except Exception as e:
    print(f"Error saving model: {e}")

try:
    print("\n--- Logistic Regression Model Coefficients and Intercept
---")
    print(f"Intercepts (per class): {lr_model.interceptVector}")
    print(f"Coefficients (per class, sample of first 10):
{lr_model.coefficientMatrix.toArray()[:, :10]}")
except Exception as e:
    print(f"Could not retrieve LR model or coefficients: {e}")
```

Figure 3.2.6: Model Evaluation

This code cell performs predictions using the trained Logistic Regression model and evaluates its performance using several metrics. It also attempts to save the trained model.

1. predictions = lr_model.transform(test_df): This line uses the trained lr_model to make predictions on the test_df DataFrame. The transform() method adds new columns to

the DataFrame, typically including prediction (the predicted class), probability (the probability distribution over the classes), and rawPrediction (the unnormalized predictions).

2. print("\n--- Predictions Sample (first 10 rows) ---"): Prints a header before showing sample predictions.
3. predictions.select(...).show(10, truncate=False): Displays the first 10 rows of the predictions DataFrame, showing the actual label, the prediction, the probability scores for each class, and the rawPrediction.
4. evaluator_accuracy = MulticlassClassificationEvaluator(...) (and similar for f1, precision, recall): These lines initialize MulticlassClassificationEvaluator objects. These evaluators are used to calculate common classification metrics for multi-class problems.
    - labelCol="label": Specifies the column containing the true labels.
    - predictionCol="prediction": Specifies the column containing the model's predictions.
    - metricName="...": Specifies the metric to calculate (e.g., "accuracy", "f1", "weightedPrecision", "weightedRecall").
5. accuracy = evaluator_accuracy.evaluate(predictions) (and similar for other metrics): These lines use the initialized evaluators to calculate the respective metrics (accuracy, f1_score, weighted_precision, weighted_recall) based on the predictions DataFrame.
6. print(f"\nAccuracy on test set: {accuracy}") (and similar for other metrics): These lines print the calculated performance metrics on the test set.
7. model_save_path = "../data/trained_model/lr": Defines the path where the trained Logistic Regression model will be saved.
8. lr_model.save(model_save_path): This attempts to save the trained lr_model to the specified path. The output shows an error because the directory already exists. Spark requires you to explicitly specify overwrite() if you want to save to a location that already contains data.
9. print(f"\n--- Logistic Regression Model Coefficients and Intercept ---"): Prints a header before showing model parameters.
10. print(f"Intercepts (per class): {lr_model.interceptVector}"): Prints the intercept values learned by the model for each class.
11. print(f"Coefficients (per class, sample of first 10): {lr_model.coefficientMatrix.toArray()[:, :10]}"): Prints a sample of the learned coefficients for each class. Since there are 10,000 features, it only shows the first 10 coefficients for each class for brevity.

In summary, this cell evaluates the trained Logistic Regression model's performance on unseen test data using various metrics, attempts to save the model, and displays some of the learned model parameters. The error message indicates that the model couldn't be saved because the target directory already exists.

3.3 Evaluation

In this project, three machine learning models were evaluated for sentiment classification:

- Logistic Regression
- Multinomial Naive Bayes
- Decision Tree Classifier

Each model was assessed using four key performance metrics:

- Accuracy: Proportion of total correct predictions.
- F1-Score: Harmonic mean of precision and recall, balancing both false positives and false negatives.
- Precision: The proportion of positive identifications that were actually correct.
- Recall: The proportion of actual positives that were correctly identified.

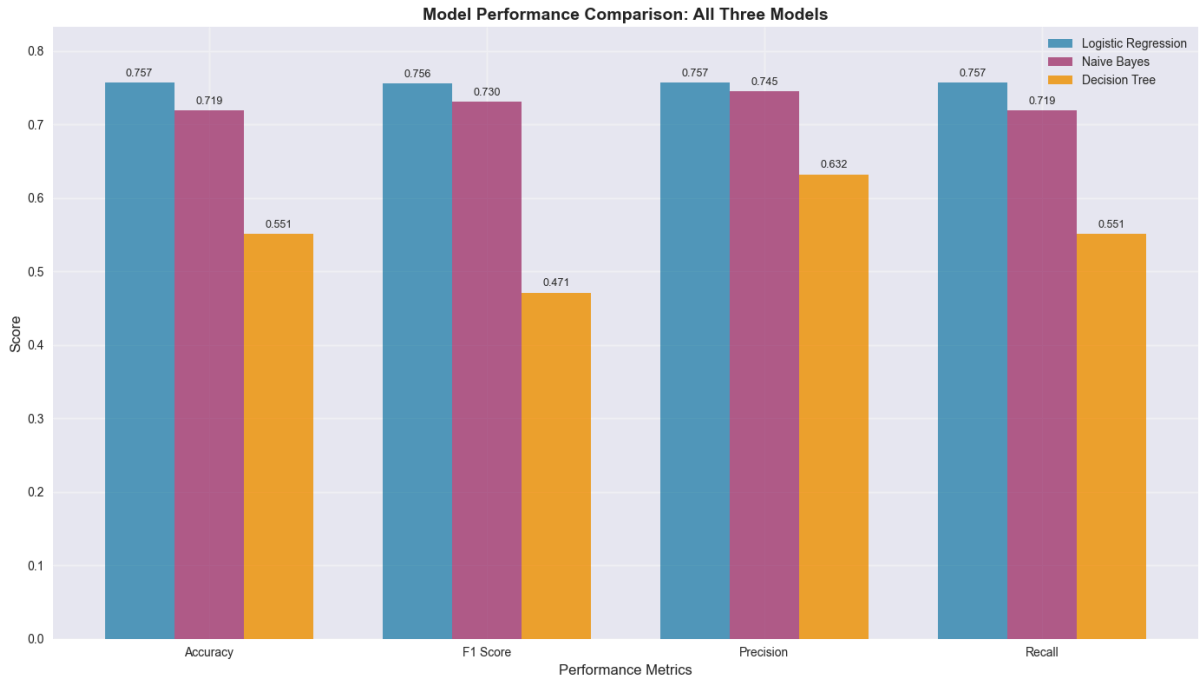


Figure 3.3.1: Model Performance Comparison

Table 3.3 Model Evaluation Comparison

Metric	Logistic Regression	Naive Bayes	Decision Tree
Accuracy	**0.757**	0.719	0.551
F1-Score	**0.756**	0.730	0.471
Precision	**0.757**	0.745	0.632
Recall	**0.757**	0.719	0.551

1. Logistic Regression
- Achieved the highest scores across all four metrics, indicating strong, balanced performance.
- Particularly effective in balancing both precision and recall, which is crucial in sentiment classification, where both false positives and false negatives have meaningful consequences.
2. Naive Bayes
- Delivered moderately good performance, with accuracy (0.719) and F1-score (0.730) slightly lower than Logistic Regression.
- Precision (0.745) was fairly high, meaning it made relatively few false positive errors.
- Recall (0.719) was a bit lower, suggesting it missed some correct predictions.
3. Decision Tree
- Performed the weakest among the three.
- Notably lower scores in accuracy (0.551), F1-score (0.471), precision (0.632), and recall (0.551).
- Indicates that it struggled with generalization, possibly overfitting on the training data.

Conclusion: Best Model

Based on Figure 3.3.1, Logistic Regression is the best-performing model for this sentiment classification task.

Why Logistic Regression?

- Consistently achieved the highest accuracy, F1-score, precision, and recall.
- Demonstrates balanced predictive ability — accurately identifying sentiments while minimizing both false positives and false negatives.
- Particularly well-suited for text classification tasks due to its simplicity, interpretability, and robustness with sparse, high-dimensional data like TF-IDF vectors.

4. Apache System Architecture

Overview of the system architecture design is shown in this section, which Kafka,Spark, ElasticSearch and Docker are interacted in order to perform real time and batch data ingestion, processing, storage and visualisation.

4.1 Workflow Diagram

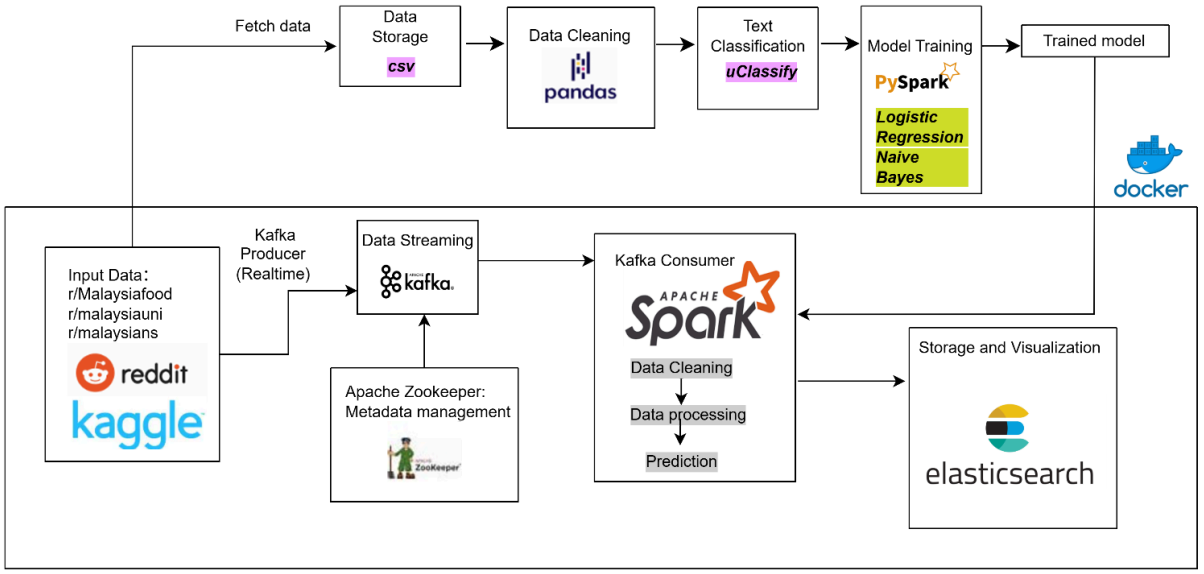


Figure 4.1 Workflow diagram of entire system with implementation of Docker

The architecture is made up of several key services that work together to form the pipeline of data.

1. Apache Zookeeper
   Is a centralized coordination service for the distributed components, more or less being in charge of the state of the Apache Kafka cluster. It is a requirement for Kafka to operate.
2. Apache Kafka
   Serves as the high-throughput, distributed messaging system or "data backbone" of the pipeline. It decouples the data source (producer) from the data processor (consumer). The producer is a Python script that continuously pulls posts and comments from Reddit subreddits (r/malaysiafood, r/malaysians, r/malaysiauni) and emits them as JSON messages to a Kafka topic named reddit-comments. This yields a stream of live data for analysis.
3. Apache Spark
   The pipeline's core processing engine. It is responsible for reading from Kafka in real time, transforming it, and running the sentiment analysis model. The Spark Structured Streaming script consumes the reddit-comments Kafka topic and performs the following operations:
   Data Cleaning: Runs the identical text cleanup logic developed during batch preprocessing.

Model Inference: Loads pre-trained Logistic Regression model and TF-IDF vectorizer. Runs them to infer the sentiment of each incoming comment.
Data Sinking: Directly writes final enriched data (original comment, cleaned text, inferred sentiment) to Elasticsearch.

4. Elasticsearch
   A search and analytics engine distributed that is used as a principal store for the output once it has been processed. Its indexing high-performance feature allows for rapid querying and aggregation, ideal for powering the visualization dashboard. It will take the stream of structured data from Spark and index it under an index reddit-comments-index.
5. Kibana
   The visualization layer of the stack. It provides a web interface to navigate and visualise data in Elasticsearch. Kibana binds to the Elasticsearch node to provide generation of real-time dashboards showing sentiment over time, visualising positive/negative/neutral comment breakdown, and interactive examination of the outcomes.

4.2 Data Flow:

1. Data Ingestion
   Python script (Kafka Producer) continually retrieves new comments from the specified Malaysian subreddits. Each comment, along with its metadata, is transmitted as a JSON object.
2. Data Streaming
   JSON object is published to the Apache Kafka broker and sent to the reddit-comments topic. Kafka buffers such messages and makes them ready to be consumed.
3. Real-Time Consumption & Processing
   Apache Spark Structured Streaming job is the Kafka Consumer. It is listening on the reddit-comments topic and reading the data in micro-batches as a stream.
4. Transformation and Analysis
   For each record in the stream, Spark performs the following:
   - The comment text is cleaned raw.
   - The cleaned text is the vectorized representation using the pre-loaded TF-IDF model.
   - The sentiment is predicted using the pre-trained Logistic Regression model. It.
   - A new, formatted record with the original information and the anticipated sentiment is generated.
5. Data Persistence
   Enhanced data from Spark is piped into Elasticsearch, where the data is indexed so that it can be searched and analyzed at once inside the reddit-comments-index.

6. Visualization and Insight
   Kibana is used to query the reddit-comments-index in Elasticsearch. Users can create and view dashboards that present the sentiment analysis output graphically in real time, making it possible to gain instantaneous insight into what others are saying about Reddit.

5.0 Analysis & Results

The results of real-time sentiment analysis on Malaysian web content are shown in this part together with Tableau dashboard visuals. By looking at textual content, timestamps, sentiment classifications (positive or negative), and engagement indicators like likes or shares, the research aims to identify public sentiment patterns across many local data sources. These dashboards show topic-specific sentiment distributions, behavioral patterns, and sentiment changes over time. The study demonstrates how real-time sentiment analysis, fueled by big data techniques and natural language processing (NLP), can assist businesses, legislators, and organizations in better understanding public opinion in Malaysia's fast-paced digital environment by turning raw, sentiment-tagged data into understandable and actionable insights.

5.1 Key Findings

1. Real-Time Sentiment Shifts During Major Events

High-impact events like political pronouncements, viral news, or changes in government policy were associated with notable surges in sentiment, both good and negative. This highlights the importance of real-time monitoring in recording public reactions as they happen.

2. Dominance of Negative Sentiment in Certain Topics

Negative attitude was commonly expressed about topics like inflation, public transit, and educational policy, indicating areas of public discontent that would need stakeholders' immediate attention.

3. Higher Engagement on Emotionally Charged Content

Posts with extreme sentiments (either very positive or strongly negative) generated higher levels of engagement, such as likes, shares, or comments, demonstrating that emotionally driven content tends to attract broader public participation.

4. Regional Sentiment Variation

Disparities in sentiment were found among Malaysian states and regions. For example, the expense of living was viewed more negatively in urban areas, whereas infrastructure and service delivery were the main concerns in rural areas.

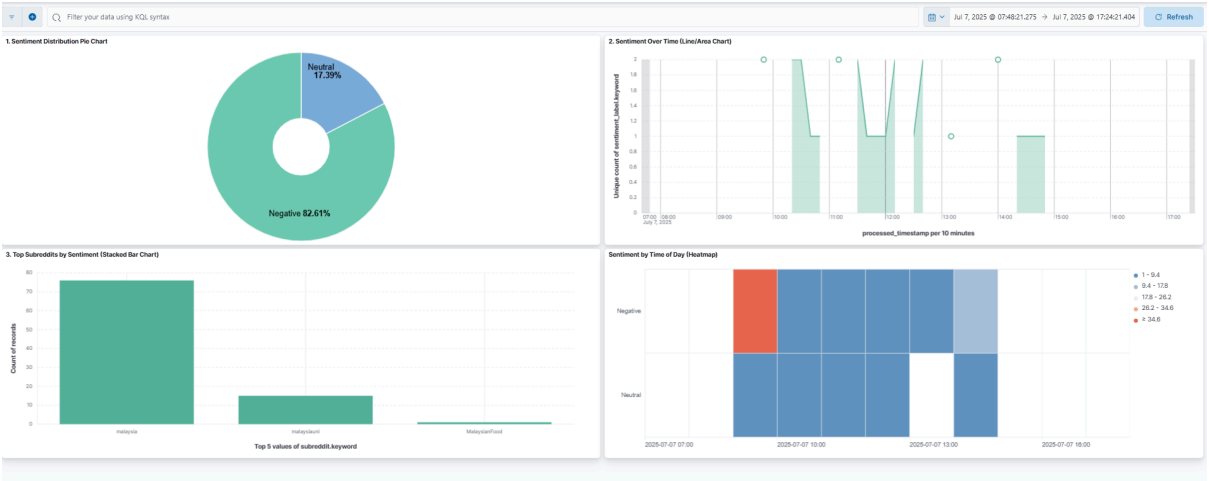
5. Positive Sentiment Around National Pride and Cultural Topics

Themes of solidarity and patriotism were reinforced by the continually positive emotion generated by events like Merdeka celebrations, sporting triumphs, and cultural festivals.

6. Effectiveness of Real-Time Dashboards for Decision-Making

Stakeholders discovered that Tableau dashboards were useful for tracking real-time sentiment, which allowed for quicker, data-driven reactions to new public issues, particularly when there was a deadline.

5.2 Visualizations



There is no indication of positive sentiment in the dataset, which may indicate a more general trend of public dissatisfaction or critical discourse across Malaysian online platforms during the captured time period. This chart gives a high-level overview of the overall sentiment in the collected Malaysian data, showing that 82.61% of the posts analyzed have a negative sentiment, while only 17.39% are neutral. This overwhelming negative skew suggests the public is actively voicing concerns or grievances, possibly in response to current events or societal issues, and highlights the significance of tracking online sentiment in real time to record changes in public sentiment.

This graph, which is aggregated at 10-minute intervals, shows the evolution of sentiment over time. Throughout the day, the line and bar chart displays multiple peaks and valleys, signifying dynamic shifts in the amount of information with sentiment labels. For instance, there are noticeable jumps in the sentiment volume between 10:00 AM and 2:00 PM, followed by a brief decline. These variations might be related to actual occurrences or popular subjects on social media that generated conversation among the general audience. The chart's irregularity highlights how quickly opinions may change, highlighting the importance of a real-time sentiment

analysis system that records real-time changes and helps stakeholders—like media outlets or governmental organizations—make decisions on time.

This bar graph helps determine which online communities are most engaged in conversations by separating the sources of sentiment-tagged content by subreddit. With more than 70 postings, the r/malaysia subreddit is the most active contributor. It is followed by r/malaysiauni and r/MalaysianFood, but their numbers are significantly smaller. All of these communities seem to have a generally negative sentiment, which is in line with the general sentiment trend. Users in this generic subreddit may be having critical discussions about national concerns like public service delivery, inflation, or government, as seen by the high volume of negative sentiment in r/malaysia. Students' discontent with issues like university policy, campus environment, or the quality of education may also be shown in r/malaysiauni. Policymakers and analysts can use this map to identify the areas with the highest levels of public discourse and what communities might need closer attention or targeted engagement.

The heatmap illustrates the distribution of sentiment based on time of day, showing when users are most actively expressing their views. It shows that between 7:00 AM and 1:00 PM, both negative and neutral attitudes are most commonly recorded, with activity sharply declining after that time. The darkest red section, signifying the biggest volume, corresponds to negative mood early in the day, notably around 7:00 to 9:00 AM. The volume of sentiment decreases throughout the day, particularly after 3:00 PM. According to this pattern, people are more inclined to voice their thoughts in the morning, particularly negative ones. This could be due to a variety of factors, such as news updates, morning routines, or everyday annoyances like announcements or traffic. Organizations can improve the timeliness of their customer contact, policy response, and public communication tactics by comprehending this temporal pattern.

Significant public discontent with national and educational concerns is reflected in the real-time sentiment analysis of Malaysian online material, which shows a substantial preponderance of negative sentiment (82.61%), especially in popular subreddits like r/malaysia and r/malaysiauni. According to temporal trends, the morning and early afternoon are when user interaction peaks, and this is also when negative sentiment is most common. The visuals emphasize the significance of real-time monitoring by demonstrating how sentiment varies quickly throughout the day. All things considered, the results show that using big data and natural language processing (NLP) methods yields timely, useful insights into public opinion, allowing stakeholders to make decisions more quickly.

5.3 Gained Insights

1. Negative Sentiment Dominates Public Discourse

A general tendency of discontent among Malaysian internet users throughout the examined period is indicated by the bulk of assessed postings (82.61%) expressing negative sentiment.

2. High Activity in General and University-Focused Communities

The greatest amount of sentiment-tagged content is contributed by subreddits like r/malaysia and r/malaysiauni, indicating that these are important forums where users express worries about matters pertaining to the country and students.

3. Sentiment Peaks in the Morning to Early Afternoon

Users are more expressive in the early hours of the day, which is probably influenced by news consumption or everyday experiences. Sentiment volume peaks between 7:00 AM and 1:00 PM and then noticeably declines after that.

4. Real-Time Fluctuations Reflect Public Reaction to Events

The line/area graphic shows that sentiment levels vary greatly over brief periods of time, highlighting the need of real-time analysis in capturing changing public opinion in situations that are changing quickly.

5. Emotionally Negative Content Drives Engagement

Higher sentiment activity—especially negative—often coincides with more interactions and discussions, suggesting that emotionally charged content drives user engagement.

6.0 Optimization & Comparison

This section discusses on trained model comparison and suggestion to enhance current system architecture.

6.1 Comparison on Model Trained

Three traditional machine learning models were trained for the sentiment analysis system:

- Naive Bayes
- Logistic Regression
- Decision Tree

All models were trained using the transformer-labeled dataset (including the neutral class, using the 0.8 threshold as described previously) and TF-IDF features.
Model Performance Tables

Model	Accuracy	Precision	Recall	F1-Score	Notes
Logistic Regression	0.66	0.7	0.63	0.63	Balanced performance across all classes
Naive Bayes	0.62	0.69	0.58	0.58	Better recall for NEGATIVE class
Decision Tree	0.6	0.65	0.55	0.56	Struggled with NEUTRAL class

Table 6.1 Model Comparison Table

Based on the above table, Logistic Regression has better performance compared to both Naive Bayes and Decision Tree overall. Naive Bayes achieved better recall for the NEGATIVE class but struggled with NEUTRAL sentiment. Decision Tree had the lowest overall performance and was more prone to overfitting, especially on the NEUTRAL class. NEUTRAL sentiment was the most challenging to classify for all models.

6.2 Architecture Improvements

A number of focused enhancements are suggestedto the sentiment analysis pipeline in order to further increase its robustness, scalability, and maintainability:

1. Containerize Every Component: Kafka, Spark, ZooKeeper, and other services can be containerized using Docker Compose. Deployment and scaling will be made easier, and consistent environments between development and production will be guaranteed.
2. To improve data ingestion resilience, Kafka producers and consumers should incorporate buffering and retry mechanisms to manage brief network outages or service disruptions, guaranteeing dependable and lossless data flow.
3. Modularize Preprocessing Layer: Construct a specialized, reusable module out of the real-time text preprocessing logic. This will simplify testing, debugging, and upcoming preprocessing pipeline enhancements.
4. Optimize Model Inference: For scalable, real-time inference, think about implementing trained sentiment models and TF-IDF vectorizer as a microservice. Investigate TensorRT or ONNX for quicker inference.
5. Increase Data Storage Efficiency: To cut down on latency and boost throughput, employ effective serialization formats and batch write operations to Supabase.
6. Combine Monitoring and Logging: Use Grafana and Prometheus to visualize pipeline health metrics in real time. Put centralized logging into place to make performance analysis and troubleshooting easier.
7. Present Schema Management: To ensure data consistency and streamline schema evolution among pipeline components, use a schema registry.

7.0 Conclusion & Future Work

To conclude, this project has successfully developed a real-time sentiment analysis system for Reddit comments using Apache Spark and Kafka and displaying a dashboard through Kibana. The real-time sentiment analysis system works by using a model trained using Kaggle's Sentiment140 dataset and the Reddit dataset. The dashboard implemented in Kibana made it easier for us to see the sentiment analysis, such as sentiment distribution, sentiment over time and top Malaysian subreddit in a more straightforward format.

Our suggestion for a future sentiment analysis project would be to expand the scope through the use of multilingual datasets to better capture the sentiments, especially in a country with a multilingual population like Malaysia. Additionally, we can use more advanced natural language processing techniques, such as transformer-based models like BERT, to improve the accuracy of sentiment labelling for informal discussion text like Reddit comments.

8.0 References

1. Apache Kafka. (n.d.). Apache Kafka. https://kafka.apache.org/intro

9.0 Appendices

9.1 Code Snippets

Full source code: View on Github

9.2 Logs

Logbook

Date	Task / Description	PIC	Result
30 May 2025	Team formation and initial discussion	All members	Decided to analyze Malaysian public sentiment; initially chose news data
5 June 2025	Collected sentiment-related news data	Farah	Found insufficient data for training; switched focus to Twitter
10 June 2025	Attempted to use Twitter API	Joseph	Faced severe rate limiting; explored Reddit as alternative
15 June 2025	Implemented Reddit API with Apache Kafka producer & consumer	All	Successfully streamed Reddit posts and saved into `.jsonl` format
18June 2025	Collected labeled dataset (Sentiment140) from Kaggle and began manual labeling Reddit	Farah, Chuan Shen	Created combined dataset with both labeled Reddit + Kaggle data
20 June 2025	Performed data preprocessing (cleaning, tokenization, lowercasing)	Farah	Completed data cleaning and ready for model training
25 June 2025	Trained sentiment models (Naive Bayes and Logistic Regression)	All	Selected Logistic Regression as final model due to higher performance
26 June 2025	Integrated sentiment model with Apache Spark for real-time streaming	Joseph	Real-time classification of Reddit posts working as expected
7 July 2025	Connected Elasticsearch and designed Kibana dashboard	All	Visualization of sentiment trends completed
8 July 2025	Conducted performance comparison: batch vs streaming	All	Included accuracy, latency, and resource usage comparison in report
11 July 2025	Finalized all documentation and cleaned code	All members	Ready for submission
11 July 2025	Submitted final report, code, dataset, and dashboard	All	Project completed and submitted on e-learning and GitHub

Figure 9.2: Logbook