MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Fifth Semester

BTech in CSE (AI & ML)

Department of Computer Science & Engineering

[Jul – Nov 2024]

Big Data Analytics Project

CSE 3145

Real-Time News Classification and Trend Analysis Using Big Data Analytics

---

Group Members:

NAME	REG. NO.	ROLL. NO.
Pratik Chakraborty	220962350	57
Shaik Nurul Ameen	220962320	51
Priyanka Pathak	220962276	42
Anoushka Smriti	220962300	46
Svadha Dey	220962450	80

Title: Real-Time News Classification and Trend Analysis Using Big Data Analytics

Abstract

In today's fast-paced digital environment, the ability to process and analyze news in real time is crucial for timely decision-making and staying ahead of emerging trends. This project proposes a framework for real-time news classification and trend analysis, leveraging big data analytics and machine learning algorithms. The system integrates Hadoop for scalable data storage, Kafka for high-throughput data streaming, Zookeeper for efficient coordination, and Apache Spark for rapid data processing and analysis.

Using machine learning models, the system classifies news articles into predefined categories and detects emerging trends as they develop. By continuously processing data streams, it provides timely insights into rapidly evolving news topics, empowering stakeholders like media organizations, financial analysts, and policymakers with actionable intelligence. This approach addresses the limitations of traditional methods, ensuring efficient data management, improved classification accuracy, and scalability for handling large volumes of news data. The real-time analysis of news offers a strategic advantage, enabling informed decisions based on the most current information.

Introduction

The exponential growth of digital news content presents challenges in managing and analyzing the vast influx of information. Traditional methods of news classification and trend analysis struggle to handle the scale, speed, and diversity of modern news data streams. This project introduces a framework for real-time news classification and trend analysis using big data technologies to meet the increasing demand for timely insights.

The framework leverages robust tools designed for big data processing. Hadoop provides scalable data storage across multiple nodes. Apache Kafka enables continuous collection and transmission of real-time news data with high throughput. Zookeeper manages synchronization and coordination among distributed components to ensure data consistency and fault tolerance. Apache Spark, with its rapid data processing capabilities, is used for real-time data transformations, machine learning operations, and analytics.

The streaming data, representing a continuous flow of news articles, is ingested by Kafka and distributed for parallel processing. Zookeeper coordinates the distributed components to maintain consistency. The data is then processed in real-time using Apache Spark, where it undergoes classification via machine learning algorithms. These algorithms categorize news into predefined topics such as politics, sports, finance, and technology. Additionally, trend

analysis identifies significant patterns and emerging themes, allowing stakeholders to stay updated on important developments.

This framework addresses the critical challenge of real-time data processing, which traditional batch processing systems cannot achieve. By utilizing stream processing, the system ingests, processes, and analyzes data as it arrives, providing timely insights. The system's scalability and low latency make it suitable for large-scale applications, ensuring responsiveness even under heavy loads.

Literature Review

Real-time news classification is an essential task in today's data-driven world, enabling the timely analysis of news articles and the extraction of actionable insights. The advent of big data technologies, combined with sophisticated machine learning models, has made it possible to efficiently process and classify the large volumes of news generated from various sources. This process involves the use of data streaming tools, such as Apache Kafka, and advanced natural language processing (NLP) pipelines, which transform raw text into structured data suitable for machine learning models. A variety of machine learning algorithms, including Naive Bayes, Support Vector Machines (SVM), Random Forests, and Decision Trees, are commonly used in text classification, each offering distinct advantages for handling real-time news streams.

Apache Kafka, introduced by Kreps et al. (2011), is a distributed messaging system that is designed to handle high-throughput, real-time data streams. It plays a critical role in enabling real-time analytics by providing a reliable pipeline through which news data can be ingested, processed, and analysed. Kafka's ability to scale horizontally across distributed systems makes it ideal for real-time news classification tasks, where large volumes of data must be processed continuously. Confluent Kafka, described by Narkhede et al. (2017), extends the capabilities of Kafka by adding features like Kafka Streams, Kafka Connect, and Schema Registry, further optimizing data stream management for real-time classification and trend analysis.

Once the news data is ingested via Kafka, it must be preprocessed using an NLP pipeline. NLP is a crucial step in transforming unstructured text into a format suitable for machine learning models. The pipeline includes steps such as tokenization, stopword removal, stemming, lemmatization, and vectorization. One of the most common methods of feature extraction in text classification is Term Frequency-Inverse Document Frequency (TF-IDF), which highlights the most important terms in a document relative to the corpus (Manning et al., 2008). By weighting words based on their frequency and significance, TF-IDF helps reduce the dimensionality of text data while retaining the most relevant information for classification.

Several machine learning algorithms are employed in news classification. Naive Bayes, as discussed by McCallum & Nigam (1998), is a simple yet effective classifier for text data. It is based on Bayes' theorem and assumes conditional independence between features, making it particularly efficient for large datasets like news articles. Despite its simplicity, Naive Bayes often delivers strong performance in text classification tasks, especially when paired with TF-IDF for feature extraction.

Support Vector Machines (SVM) are another popular algorithm for text classification, particularly in handling high-dimensional feature spaces typical of textual data. SVMs work by finding the optimal hyperplane that separates classes with maximum margin (Joachims, 1998). In news classification, SVMs excel in handling sparse and noisy text data, making them suitable for distinguishing between similar news topics or categories.

Random Forests and Decision Trees are frequently used in text classification due to their interpretability and versatility. Decision Trees build a model that splits data based on feature values, creating a hierarchical structure that is easy to interpret but prone to overfitting. Random Forests, introduced by Breiman (2001), address this issue by creating an ensemble of Decision Trees, which enhances model robustness and reduces variance. Random Forests are particularly effective in handling large, unstructured news datasets, as demonstrated by Lee et al. (2020) in their study on real-time news classification using big data frameworks. By leveraging the power of ensemble learning, Random Forests provide better generalization in diverse and evolving news environments.

Recent studies have explored hybrid approaches, combining different algorithms for improved accuracy in news classification. Sun & Du (2021) investigated the integration of Random Forests and SVMs, showing that these hybrid models, when used in conjunction with feature extraction techniques like TF-IDF, can outperform individual algorithms in terms of accuracy and efficiency. The combination of machine learning models with big data processing frameworks enables real-time classification systems to handle the scale and complexity of continuous news streams.

Big data frameworks like Apache Spark are also crucial in scaling real-time news classification. Spark's distributed computing capabilities allow it to process vast amounts of data quickly and efficiently. Azizoğlu (2018) highlighted Spark's utility in handling large-scale text classification tasks, making it a valuable tool for real-time applications. In a similar vein, Drakopoulos et al. (2020) demonstrated how Spark could be used for real-time trust analytics on social media data, showcasing the framework's applicability to real-time news classification and trend analysis.

Moreover, Pintye et al. (2018) proposed a cloud-based big data and machine learning framework for text classification, emphasizing the importance of scalability in real-time applications. Khan et al. (2018) extended this work by developing a two-stage big data analytics framework using Spark and Long Short-Term Memory (LSTM) networks, which could be adapted for real-time news classification. These frameworks offer significant advantages in terms of speed, scalability, and performance, making them indispensable for handling the continuous flow of news data in real-time systems.

From BERT Nugroho et.al. (2021)' s article stands for Bidirectional Encoder Representations from Transformers is a deep learning architecture that can be used for downstream NLP tasks. The architecture consists of a stacked encoder layer from the Transformer. BERT takes a distinctive approach to learning. Bidirectional means that BERT learns from the left and right sides of the token during learning. A bidirectional method is essential to understand the meaning of language. There are two main steps in BERT: pre-training and fine-tuning. During pretraining, BERT is trained in a large unlabeled corpus with two unsupervised tasks: masked language model (MLM) and next sentence prediction (NSP) to produce a pre-trained model. For fine-tuning, the model is initialized with the pre-trained parameters, and all the parameters are fine-tuned using labeled data for specific tasks such as classification.

Methodology

1.1 Installations and Initial Imports

a. pyspark: For distributed data processing with Apache Spark.
b. confluent_kafka: To interact with Kafka for streaming data.
c. newsapi-python: To fetch real-time news articles.
d. scikit-learn: For machine learning algorithms and tools.
e. pandas: For data manipulation and analysis.
f. numpy: For numerical operations and handling arrays.
g. matplotlib: For data visualization.
h. torch: For deep learning with PyTorch.
i. transformers: For pre-trained NLP models like BERT.
j. spacy: For advanced NLP tasks.
k. nltk: For text processing tasks

1.2 Setup for Apache Spark and Java

- Java Installation: Since Spark requires Java, we installed OpenJDK to provide the necessary runtime environment.

- Environment Variables: Java and Spark environment variables were set to ensure they are correctly initialized in the environment.
- FindSpark Setup: To easily locate and use the installed Spark package, we initialized FindSpark.
- Download and Install Apache Spark: We downloaded Spark 3.1.2 and unpacked it to enable distributed data processing.
- FindSpark Installation: We installed FindSpark to make sure PySpark runs in the Colab environment.

2. Overview of Kafka and NewsAPI Setup

1. Setup NewsAPI Client

The first part of the code focuses on initializing the NewsAPI client and fetching news articles:

1. Importing the NewsAPI Library: The NewsApiClient is imported to interact with the NewsAPI service, allowing the application to retrieve news articles from various sources.
2. Initializing the NewsAPI Client: The client is initialized with an API key, which is essential for authenticating requests to the NewsAPI. This key allows access to the service and is necessary for fetching news data.
3. Fetching a Single Article: A function (fetch_single_article) is defined to encapsulate the logic for retrieving a single news article. This function:
    1. Calls the NewsAPI to get the top headlines with a limit of one article.
    2. Extracts relevant details (title, description, content) from the response.
    3. Returns the article as a dictionary, which is useful for further processing or sending to Kafka.

This modular approach ensures that the fetching logic is reusable and can be easily modified if needed.

2. Configure Confluent Kafka Producer

About Confluent Kafka

Confluent Kafka is an enterprise-grade event streaming platform built on Apache Kafka, which provides additional tools and services to enhance the management, security, and scalability of Kafka. It allows organizations to harness the power of real-time data streams, with features like cloud-native deployments, connectors, schema

registry, and multi-cluster management. Confluent Kafka is particularly useful for implementing high-throughput and low-latency event-driven architectures, such as streaming applications, real-time analytics, and data pipelines.

The next part sets up a Kafka producer to send the fetched articles to a Kafka topic:

1. Importing the Kafka Producer: The Producer class from confluent_kafka is imported to facilitate the sending of messages to Kafka.
2. Configuring the Kafka Producer: A configuration dictionary (conf) is created, specifying essential parameters such as:
    a. Bootstrap Server: Indicates the Kafka server to connect to.
    b. Security Protocol: Sets the protocol for secure communication.
    c. SASL Mechanism: Defines the authentication method.
    d. API Key and Secret: These are used for authentication with Confluent Cloud.

    This configuration is crucial for establishing a secure connection to Kafka and enabling the application to produce messages.

    a. Creating the Producer: An instance of the Kafka producer is created using the configured parameters, which prepares the application for sending messages.
    b. Delivery Report Callback: A callback function (delivery_report) is defined to handle message delivery reports. This function provides feedback on the success or failure of message delivery, allowing for monitoring and debugging of the Kafka interaction.

3. Sending News Data to Kafka

This section focuses on sending the fetched news articles to a Kafka topic:

a. Importing JSON Library: The json library is imported to serialize the article data into a JSON format before sending it to Kafka.
b. Function to Send News to Kafka: The function (produce_news_to_kafka) is defined to handle the sending process. It:
    1. Fetches a single article using the previously defined function.
    2. Serializes the article to JSON format.
    3. Sends the JSON-encoded article to a specified Kafka topic.
    4. Calls producer.flush() to ensure that all pending messages are sent, maintaining data integrity.

This encapsulation enhances code reusability and clarity, allowing for easier modifications and testing.

4. Run the Producer Periodically

The next part of the code implements a loop to fetch and send news articles continuously:

a. Infinite Loop for Periodic Fetching: A while True loop is used to repeatedly fetch and send articles to Kafka. This enables real-time data ingestion, making it suitable for applications that require timely updates.

b. Time Delay: A time.sleep(10) statement is included to pause execution for 10 seconds between fetches. This interval controls the frequency of data ingestion, preventing excessive API calls and ensuring compliance with rate limits.

c. Graceful Shutdown: A try/except block is used to catch KeyboardInterrupt exceptions, allowing for a clean exit from the infinite loop when the user decides to stop the process.

`5. Run the Kafka Consumer:

The Kafka consumer is responsible for fetching and storing the news articles from the Kafka topic.

a) Polling for Messages: The consumer enters an infinite loop that continuously polls Kafka for new messages. This allows the application to receive real-time news articles as they are streamed to the Kafka topic.
b) Timeout Handling: If no new messages are received after 10 consecutive polls, the consumer stops. This ensures the application doesn't run indefinitely when no new data is available.
c) Message Processing: Each message received from the Kafka topic is decoded from JSON and stored in a list for further analysis.
d) Graceful Shutdown: The consumer is wrapped in a try/except block to allow for a clean exit when interrupted by the user, ensuring proper closure of the Kafka consumer.

3. Classify and Save Articles Using Zero-Shot Classification:

In this step, we utilize the Hugging Face Transformers library to classify the news articles into predefined categories based on their descriptions. This approach leverages the powerful zero-shot classification technique, which allows the model to categorize text without explicit prior training on the specific labels.

a. Extract Descriptions

Description Extraction: We retrieve the article descriptions from the JSON objects fetched from Kafka. Descriptions serve as the main input for classification, capturing the essence of each article. If any article lacks a description, a default message is used instead.

b. Initialize Zero-Shot Classifier

Zero-Shot Classification: This technique enables models to predict labels for text that the model has never seen during training. Using pre-trained NLP models like BERT or its variants (often RoBERTa for zero-shot tasks), the model is capable of understanding text in a more generalizable way.

Transformers Library: The transformers library by Hugging Face provides access to cutting-edge pre-trained NLP models. It simplifies the use of models like BERT, GPT, and RoBERTa. By using the pipeline function, we can easily load a zero-shot classification model that understands the input text and predicts the category that best matches the text, even if the specific categories were not part of the model's training data.

c. Classify Descriptions

Candidate Labels: We define a set of candidate labels—entertainment, politics, business, sports, climate, and science. These are the categories into which we want to classify each article.

Classification Process: The classifier assigns a confidence score to each label for each description. The label with the highest confidence is chosen as the category. This process is powered by the model’s ability to understand the context and meaning of the text, even without explicit training on these specific labels.

d. Save Classified Data

Storing Results: After classifying each description, the result (description and category) is saved into a CSV file. This step ensures that the classified data is organized and stored for further analysis, visualization, or reporting.

CSV File Handling: If the CSV file already exists, new rows are appended to it, making this step scalable as more articles are fetched and classified. If it does not exist, a new CSV file is created with the appropriate headers.

4. NLP Pipeline Overview

The NLP pipeline is designed to preprocess raw text data for effective machine learning analysis. It includes four main steps:

1. Text Cleaning

a) Remove Useless Characters: Unwanted elements like URLs, HTML tags, special characters, and numbers that don't add meaning to the text are removed. This helps focus on the core content.

b) Standardize Text: By converting all characters to lowercase, the function ensures that words like "Apple" and "apple" are treated the same during analysis.
c) Reduce Noise: Extra spaces and punctuation are removed, making the text more uniform and easier to process in further steps such as tokenization or vectorization

2. Tokenization

This initial step breaks text into smaller units called tokens, typically words. Tokenization removes unwanted characters and symbols, ensuring that only meaningful units are preserved. Using NLTK's word_tokenize, we accurately identify token boundaries, maintaining the context of the text.

3. Stopword Removal

Stopwords are common words (like "and" and "the") that offer little value in understanding the text's meaning. By eliminating these words, we reduce the dataset's dimensionality, making it more efficient for analysis. NLTK's stopwords corpus helps ensure consistency in removing non-informative words.

4. Lemmatization and Stemming

This step reduces words to their base forms. Lemmatization uses a dictionary approach, while stemming applies heuristic rules to remove prefixes or suffixes. Combining both techniques allows us to treat different forms of a word as the same (e.g., "running" and "ran"), leading to a more unified representation of the data.

5. Vectorization: TF-IDF Vector Embeddings

In this section, we implement a system to vectorize news articles using TF-IDF (Term Frequency-Inverse Document Frequency) embeddings. This approach captures the importance of words within a document and across the entire corpus, transforming textual data into numerical vectors that are suitable for machine learning algorithms.

a. Fetching Articles:

The articles are collected and preprocessed into tokens, such as lemmatized and stemmed tokens. The content of each article is prepared for vectorization by ensuring it is tokenized appropriately.

b. Vectorizing Articles:

The vectorize_articles function performs the TF-IDF vectorization on the column containing the preprocessed tokens (lemmatized and stemmed). The vectorization process is as follows:

- Custom Tokenizer: A custom identity tokenizer is used, which passes the pre-tokenized words directly to the vectorizer without additional tokenization.
- TF-IDF Vectorizer: The TfidfVectorizer from sklearn is initialized with the custom tokenizer. It calculates the TF-IDF values for each token in the corpus.

- Transformation: The function transforms the preprocessed tokens into a TF-IDF matrix, where each article is represented as a vector of numerical values corresponding to the importance of each word.

This transformation results in a matrix that represents the articles in numerical format, which is suitable for feeding into machine learning models.

c. Adding TF-IDF Vectors to DataFrame:

After vectorization, the resulting TF-IDF vectors are added as a new column (tfidf_vector) to the DataFrame for further use in downstream tasks like classification or clustering.

6. BERT Embedding Model

In this section, we utilize BERT (Bidirectional Encoder Representations from Transformers) to generate contextual embeddings that represent each article's description in a rich, semantically meaningful way. These embeddings are highly effective for machine learning tasks due to BERT's ability to understand the context of words in a sentence.

a. Load BERT Model and Tokenizer:

The load_bert_model function is used to initialize and load the pre-trained BERT base uncased model along with its tokenizer from Hugging Face's transformers library. The tokenizer handles converting raw text into the format that BERT can process, while the model generates the actual embeddings.

b. Generate BERT Embeddings:

The get_bert_embeddings function processes the descriptions and outputs contextual embeddings:

a) Tokenization and Encoding: Each description in the dataset is tokenized and encoded using the BERT tokenizer, with the text padded and truncated to a maximum length of 512 tokens.
b) Embedding Extraction: The encoded inputs are passed through the BERT model to obtain hidden states. The embedding corresponding to the [CLS] token is extracted for each description, as it captures the overall representation of the input sequence.
c) Batch Embeddings: The function aggregates these embeddings into a matrix, where each row corresponds to the embedding of a single description.

c. Adding BERT Embeddings to DataFrame

The resulting BERT embeddings are stored in a new column, bert_embeddings, in the DataFrame. Each row of this column contains a vector representing the semantic meaning of the corresponding description.

7. ML Pipeline

1. Setup for PySpark ML Pipeline with BERT and TF-IDF Embeddings:

   In this section, we integrate PySpark to build a machine learning pipeline using BERT and TF-IDF embeddings for classification tasks. PySpark allows for scalable and distributed processing, making it ideal for handling large datasets in real-time systems.

   a. Initialize Spark Session

   We begin by initializing a Spark session using SparkSession.builder.appName("ML Pipeline"). This session is necessary for managing data and performing distributed operations across a cluster.

   b. Convert Embeddings and Labels to PySpark DataFrame

   For TF-IDF Embeddings:

   The TF-IDF vectors are combined with the corresponding labels (i.e., categories of the news articles) to create a PySpark DataFrame. Each entry in the DataFrame consists of a label and a vector of features (TF-IDF values).

   Using the VectorAssembler, the TF-IDF vectors are assembled into a PySpark-compatible format (Vectors.dense), which is essential for the machine learning models.

   For BERT Embeddings:

   Similarly, the BERT embeddings, which capture richer semantic representations of the text, are also combined with the article labels to form a PySpark DataFrame.

   c. StringIndexer for Label Conversion

   In both cases (TF-IDF and BERT embeddings), the labels are initially in string format. To make them suitable for machine learning models, we use the StringIndexer to convert these string labels into numeric indices. The transformed DataFrame now contains both features (embedding vectors) and numeric labels.

   d. Train-Test Split

   To evaluate the models effectively, we split the data into training and testing sets using an 80-20 ratio. This ensures that the models are trained on the majority of the data and tested on unseen data for performance evaluation.

2. Naive Bayes:
   a) BERT Embeddings: We use Naive Bayes with BERT embeddings to perform tasks such as sentiment analysis or fake account detection. However, Naive Bayes generally assumes non-negative feature values, which may limit its compatibility with BERT embeddings due to the presence of negative values.

b) TF-IDF Embeddings: Naive Bayes works particularly well with TF-IDF vectors as they are non-negative, making it a suitable choice for this model

3. Random Forest:
    a) BERT Embeddings: The Random Forest classifier is an ensemble method that leverages decision trees to capture complex patterns in the data. Using BERT embeddings as input features, this method can improve classification accuracy by considering the rich contextual information captured by BERT.
    b) TF-IDF Embeddings: Random Forest is also effective with TF-IDF vectors, capturing the importance of different terms in the text and building a robust classifier
4. Support Vector Machine (SVM):
    a) BERT Embeddings: The SVM model can be applied with BERT embeddings for high-dimensional data classification. The use of cosine similarity between embeddings enhances its performance for tasks that require understanding the contextual relationships between words.
    b) TF-IDF Embeddings: With TF-IDF vectors, SVM remains an effective method, especially for tasks involving sparse data, where it can efficiently separate different categories by finding the optimal hyperplane.
5. Logistic Regression:

a) BERT Embeddings: Logistic Regression is a simple yet effective linear classifier, particularly suited for binary and multiclass classification tasks. When used with BERT embeddings, Logistic Regression leverages the rich contextual representations provided by BERT, making it a strong baseline model for text classification tasks.
b) TF-IDF Embeddings: Logistic Regression also performs well with TF-IDF vectors, using the importance of words (as captured by the TF-IDF scores) to predict the target class. It works well for both sparse and dense representations, and is a commonly used model for text classification tasks due to its interpretability and efficiency.

Results:

Output of Kafka Producer:

```
Fetching and sending news to Kafka...
Message delivered to topic_0 [5]
Fetching and sending news to Kafka...
Message delivered to topic_0 [5]
Fetching and sending news to Kafka...
Message delivered to topic_0 [1]
Fetching and sending news to Kafka...
Message delivered to topic_0 [4]
Fetching and sending news to Kafka...
Message delivered to topic_0 [3]
Fetching and sending news to Kafka...
Message delivered to topic_0 [3]
Fetching and sending news to Kafka...
Message delivered to topic_0 [5]
Stopped fetching and sending articles.
```

Description of the output

1) Fetching and Sending News to Kafka:

   The message "Fetching and sending news to Kafka..." is printed each time the script fetches an article and sends it to the Kafka topic. This message appears before the article is sent and repeats every time the produce_news_to_kafka() function is executed.

2) Message Delivery Confirmation:

   The line "Message delivered to topic_0 [X]" indicates that a message (news article) has been successfully delivered to the Kafka topic named topic_0. The number inside the square brackets [X] likely refers to the partition or an identifier related to message delivery.This confirmation shows that the message has been produced and sent to Kafka successfully, and it happens after each fetch/send operation.

3) Stopped Fetching and Sending Articles:

   The message "Stopped fetching and sending articles." is printed when the script is interrupted by the user (e.g., by pressing Ctrl + C). This indicates that the loop has been terminated gracefully, and no more articles are being fetched or sent to Kafka.

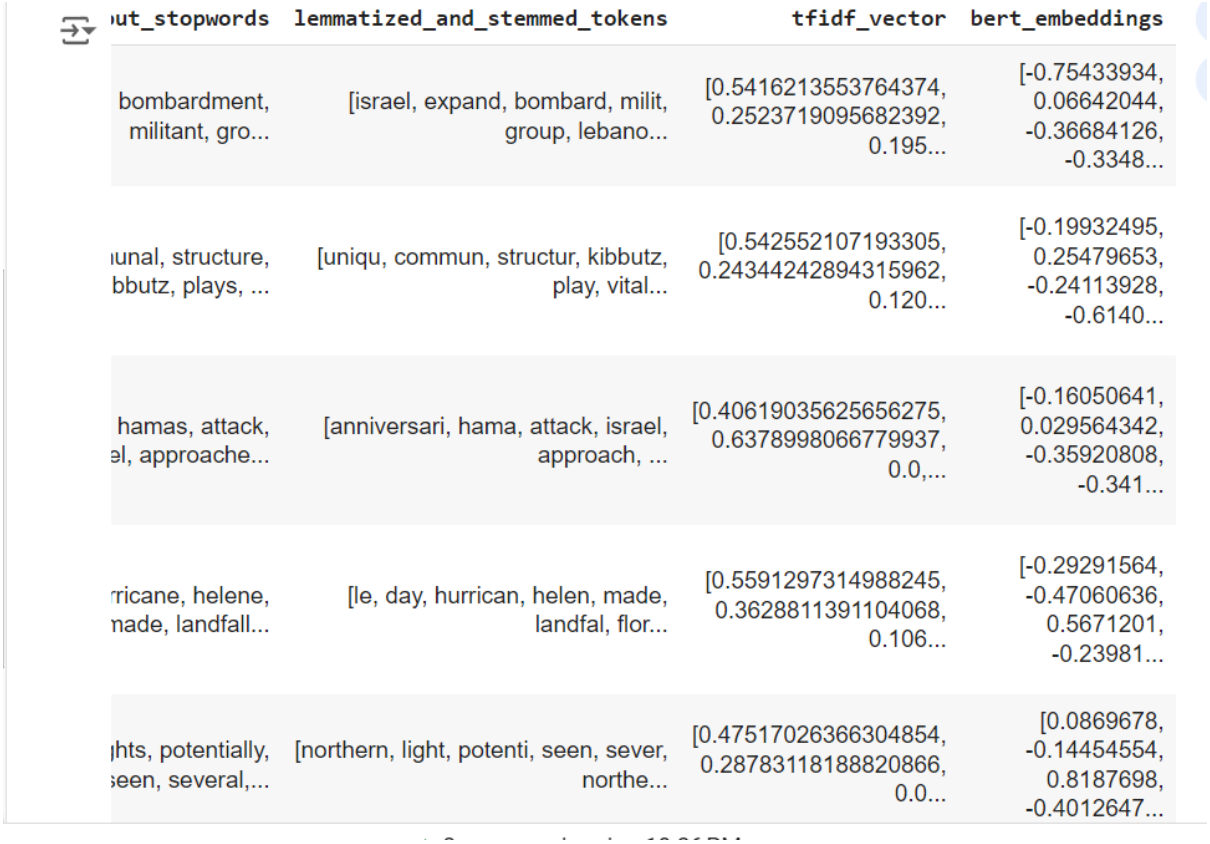
Output of Dataframe column containing article description and category given from RoBERTa Model::

	Description	Category
0	Israel is expanding its bombardment of militan...	politics
1	The unique communal structure of the kibbutz p...	climate
2	As the anniversary of the Hamas attack on Isra...	politics
3	Less than 10 days after Hurricane Helene made ...	climate
4	The Northern Lights can potentially be seen fr...	science

Output of article descriptions after going through the NLP Pipeline:

	Description	Category	cleaned_text	tokenized_text	tokenized_without_stopwords	lemmatized_and_stemmed_tokens
0	Israel is expanding its bombardment of militan...	politics	israel is expanding its bombardment of militan...	[israel, is, expanding, its, bombardment, of, ...	[israel, expanding, bombardment, militant, gro...	[israel, expand, bombard, milit, group, lebano...
1	The unique communal structure of the kibbutz p...	climate	the unique communal structure of the kibbutz p...	[the, unique, communal, structure, of, the, ki...	[unique, communal, structure, kibbutz, plays, ...	[uniqu, commun, structur, kibbutz, play, vital...
2	As the anniversary of the Hamas attack on Isra...	politics	as the anniversary of the hamas attack on isra...	[as, the, anniversary, of, the, hamas, attack,...	[anniversary, hamas, attack, israel, approache...	[anniversari, hama, attack, israel, approach, ...
3	Less than 10 days after Hurricane Helene made ...	climate	less than days after hurricane helene made lan...	[less, than, days, after, hurricane, helene, m...	[less, days, hurricane, helene, made, landfall...	[le, day, hurrican, helen, made, landfal, flor...
4	The Northern Lights can potentially be seen fr...	science	the northern lights can potentially be seen fr...	[the, northern, lights, can, potentially, be, ...	[northern, lights, potentially, seen, several,...	[northern, light, potenti, seen, sever, northe...

Output of TF-IDF vectors and BERT embeddings of the article descriptions:



ıut_stopwords	lemmatized_and_stemmed_tokens	tfidf_vector	bert_embeddings
bombardment, militant, gro...	[israel, expand, bombard, milit, group, lebano...	[0.5416213553764374, 0.2523719095682392, 0.195...	[-0.75433934, 0.06642044, -0.36684126, -0.3348...
ıunal, structure, bbutz, plays, ...	[uniqu, commun, structur, kibbutz, play, vital...	[0.542552107193305, 0.24344242894315962, 0.120...	[-0.19932495, 0.25479653, -0.24113928, -0.6140...
hamas, attack, el, approache...	[anniversari, hama, attack, israel, approach, ...	[0.40619035625656275, 0.6378998066779937, 0.0,...	[-0.16050641, 0.029564342, -0.35920808, -0.341...
rricane, helene, nade, landfall...	[le, day, hurrican, helen, made, landfal, flor...	[0.5591297314988245, 0.3628811391104068, 0.106...	[-0.29291564, -0.47060636, 0.5671201, -0.23981...
ghts, potentially, seen, several,...	[northern, light, potenti, seen, sever, northe...	[0.47517026366304854, 0.28783118188820866, 0.0...	[0.0869678, -0.14454554, 0.8187698, -0.4012647...

We have trained the ML models separately using the BERT embeddings and TF-IDF vectors.

The below table displays the different accuracy of the ML models using the 2 vector embedding models.

	TF-IDF	BERT
Naive Bayes	25%	Not included since Naive Bayes cannot take negative values
Random Forest	45.83%	70.83%
Support Vector Machine(SVM)	41.67%	79.17%
Logistic Regression	41.67%	79.17%

Since the BERT embeddings have negative values, training was not done using the Naive Bayes model. SVM and Logistic Regression with BERT has the highest accuracy of 79.17%.

Overall, BERT gave much better results than TF-IDF because it provides a more contextual and nuanced representation of the article descriptions, capturing the semantic relationships

between words and sentences more effectively than TF-IDF, which primarily relies on term frequency without considering word meaning or context.

Conclusion:

This project effectively illustrates the capabilities of big data analytics in transforming news processing into a real-time operation. Through the integration of powerful technologies like Apache Kafka and Apache Spark, the framework is able to process continuous streams of news articles while maintaining scalability and responsiveness. Kafka plays a pivotal role in ensuring efficient data ingestion and real-time streaming, while Spark provides the necessary speed for data transformations and classification.

The system employs machine learning models, including Naive Bayes, Support Vector Machines (SVM), Random Forests and Logistic Regression to classify news into relevant categories such as politics, sports, and climate. In parallel, trend analysis helps uncover emerging patterns, offering crucial insights that would otherwise go unnoticed in traditional batch-processing systems. The combination of Natural Language Processing (NLP) techniques like tokenization, stopword removal, and lemmatization refines the text data, allowing the models to perform more effectively on large-scale datasets.

There is focus on enhancing the accuracy and depth of classification through advanced language models such as BERT and exploring new methods for more granular trend detection. Additionally, deploying this framework in a cloud-based environment could further improve its scalability, making it ideal for handling even larger datasets while maintaining performance.

In conclusion, this project has demonstrated that with the right combination of big data technologies and machine learning algorithms, it is possible to manage, classify, and analyze vast amounts of real-time news data efficiently, providing stakeholders with the actionable insights they need in today's fast-moving digital landscape.

References

Azizoğlu, U. R. (2018). Text classification using apache spark.

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Drakopoulos, G., Kanavos, A., Paximadis, K., Ilias, A., Makris, C., & Mylonas, P. (2020, November). Computing massive trust analytics for Twitter using Apache Spark with account self-assessment. In WEBIST (pp. 403-414).

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. European Conference on Machine Learning.

Khan, M. A., Karim, M. R., & Kim, Y. (2018). A two-stage big data analytics framework with real world applications using Spark machine learning and long short-term memory network. Symmetry, 10(10), 485.

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. NetDB.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. AAAI-98 Workshop on Learning for Text Categorization.

Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka: The definitive guide: Real-time data and stream processing at scale. O'Reilly Media.

Pintye, I., Kail, E., Kacsuk, P., & Lovas, R. (2018). Big data and machine learning framework for clouds and its usage for text classification.

Sun, N., & Du, C. (2021). [Retracted] News Text Classification Method and Simulation Based on the Hybrid Deep Learning Model. *Complexity*, *2021*(1), 8064579.

K. Lee, D. Palsetia, R. Narayanan, M. M. A. Patwary, A. Agrawal and A. Choudhary, "Twitter Trending Topic Classification," *2011 IEEE 11th International Conference on Data Mining Workshops*, Vancouver, BC, Canada, 2011, pp. 251-258, doi: 10.1109/ICDMW.2011.171.

Kuncahyo Setyo Nugroho, Anantha Yullian Sukmadewa, and Novanto Yudistira. 2021. Large-Scale News Classification using BERT Language Model: Spark NLP Approach. In Proceedings of the 6th International Conference on Sustainable Information Engineering and Technology (SIET '21). Association for Computing Machinery, New York, NY, USA, 240–246. https://doi.org/10.1145/3479645.3479658

Contributions

NAME	WORK
AMEEN	Kafka, News API Setup
SVADHA	Support Vector Machines
PRATIK	NLP Pipeline, RoBERTa model
PRIYANKA	Naïve Bayes and Logistic Regression with BERT, TF-IDF
ANOUSHKA	Logistic Regression and Random Forest with TF-IDF