ARBAMINCH UNIVERSITY FACULTY OF COMPUTING AND SOFTWARE ENGINEERING

Course: Big Data Analytics & Business Intelligence Group F Assignment (Social media sentiment)

FINAL REPORT

Name	Id
Abigiya Solomon ---------	NSR/458/14
Betel Tarekegn ----------	NSR/3010/14
Elsa Husen ----------	NSR/852/13
Meklit Mathewos ----------	NSR/2460/14
Solomon Aragaw ----------	NSR/1998/14

Contents

Introduction

Social media platforms have emerged as powerful sources of public opinion, generating vast volumes of textual data daily. Platforms such as Twitter allow users to share thoughts, emotions, and reactions to events worldwide, making social media an invaluable resource for understanding sentiment trends. However, analyzing this data presents challenges due to its high volume, informal language, slang, abbreviations, emojis, and inconsistent structure.

Big Data Analytics and Business Intelligence techniques offer effective solutions for handling such complex datasets. Distributed processing frameworks like Apache Spark and storage systems such as Hadoop Distributed File System (HDFS) enable scalable data ingestion, cleaning, transformation, and analysis. When combined with machine learning algorithms, these tools allow automated sentiment classification at scale.

This project focuses on developing a complete end-to-end big data pipeline for social media sentiment analysis. The pipeline covers ingestion of structured and semi-structured datasets, ETL processing, data cleaning, feature engineering using TF-IDF, exploratory data analysis (EDA), and training multiple machine learning models. The selected models - Logistic Regression, Multinomial Naive Bayes, and Linear Support Vector Machine (SVM) - are compared to determine the most effective approach for sentiment prediction. The final system emphasizes scalability, reproducibility, and deployment readiness, providing a robust framework for realworld sentiment analysis applications.

Data Ingestion

Data ingestion is the process of collecting raw data from various sources and importing it into a storage system or data pipeline for further processing and analysis. It ensures that data from diverse formats - such as CSV files, databases, or APIs - is structured, accessible, and ready for downstream tasks like cleaning, transformation, and machine learning. In big data environments, tools like Spark and HDFS are commonly used to handle large-scale ingestion efficiently and reliably.

Structured Data

Ingesting raw social media data into a structured format is a critical first step in any data analytics pipeline. Raw CSV files often contain unorganized or inconsistently formatted records, which can complicate downstream analysis. In this project, COVID-19-related tweets are accepted from a local CSV file and read into a Spark DataFrame, allowing scalable processing even for large datasets. Once validated, the data is stored in a structured format using Parquet on HDFS at the path *hdfs://localhost:9000/raw/structured*. Parquet's columnar storage provides efficient compression and faster query performance, making the dataset easier to process for subsequent tasks such as data cleaning, feature extraction, exploratory data analysis, and machine learning. The ingestion process employs overwrite mode to ensure the HDFS location always contains the most up-to-date version of the data. The structured ingestion offers multiple benefits: scalability across distributed nodes, efficient storage and access, reproducibility for consistent downstream processing, and a solid foundation for analytics and predictive modeling.

Semi-structured Data

Semi-structured data, such as JSON files, contains organizational elements like keys and values but does not strictly follow a fixed schema like relational tables. Social media posts, logs, and API outputs often fall into this category. Efficient ingestion of semi-structured data ensures that it can be analyzed, cleaned, and transformed into formats suitable for machine learning and analytics. In this project, the dataset consists of semi-structured JSON files containing tweet text, user details, timestamps, and metadata. Spark's JSON reader is used to load these files into a DataFrame, automatically inferring the schema and handling nested structures to support scalable processing of large datasets. After loading, the DataFrame is written to HDFS in Parquet format

at *hdfs://localhost:9000/raw/semi* using overwrite mode, converting the semi-structured data into a columnar storage format for better query performance and reduced storage requirements. The benefits of this semi-structured ingestion include flexibility to handle varying JSON structures, scalability through distributed Spark processing, efficiency via Parquet compression and fast read/write operations, and preparation of the data in a structured format suitable for downstream cleaning, feature engineering, and machine learning workflows.

Build First ETL Pipeline

ETL (Extract, Transform, Load) is a fundamental process in data engineering that prepares raw data for analysis. Structured data, like CSV files, often contains inconsistencies such as extra whitespace, inconsistent casing, or missing metadata. An ETL pipeline cleans, standardizes, and enriches the data, making it ready for analytics and machine learning workflows.

ETL Structure

1. Extract

   The raw structured dataset is read from HDFS, where it was previously ingested. Spark efficiently loads large datasets distributed across nodes, allowing scalable processing.

2. Transform

   Several transformations are applied to ensure data quality:

   - Whitespace Cleaning and Lowercasing: All string columns are trimmed and converted to lowercase to maintain consistency.
   - Add Source Column: A new column source is added to identify the origin of the data, which is useful for auditing and downstream processing.

3. Load

   The transformed DataFrame is written back to HDFS in Parquet format under a clean, processed path. Using overwrite mode ensures that the latest transformations replace any previous data.

ETL Semi-Structure

Semi-structured data, such as JSON files from social media or APIs, often contains nested fields, inconsistent formatting, and variable key names. An ETL pipeline is essential to clean, standardize, and prepare this data for analysis and machine learning applications.

ETL Process for Semi-Structured Data

1. Extract

   The semi-structured dataset is read from HDFS, where it was previously ingested as Parquet. Spark efficiently handles large, distributed datasets, automatically managing schema inference and nested structures.

2. Transform

   The following transformations are applied:

   - Whitespace Cleaning and Lowercasing: All string columns are trimmed and converted to lowercase to ensure consistency.
   - Rename Column: The column body is renamed to text to standardize naming conventions across datasets.
   - Add Source Column: A source column is added to indicate the origin of the data (semi_structured), which aids in tracking and auditing.

3. Load

   The cleaned and transformed DataFrame is written back to HDFS in Parquet format using overwrite mode. This structured and standardized dataset is now ready for downstream tasks.

Data cleaning

Data cleaning is a critical step in any data processing or analytics workflow. Raw datasets, whether structured, semi-structured, or unstructured, often contain inconsistencies, missing values, noise, irrelevant characters, and duplicates. These issues can negatively impact statistical analysis, machine learning models, and overall data reliability. Cleaning ensures that the dataset is accurate, consistent, and ready for further processing.

Key aspects of data cleaning include handling missing values, correcting inconsistent formatting, removing irrelevant or noisy data, standardizing text, and eliminating duplicate records. By addressing these issues early, the dataset becomes more structured and reliable, which improves the effectiveness of downstream processes such as feature extraction, predictive modeling, and analytics. Proper cleaning also enhances reproducibility and ensures that insights drawn from the data are meaningful and trustworthy.

Clean Structured Data

Structured data, even after basic ETL transformations, may still contain issues that affect analysis and machine learning performance. Problems such as unwanted characters, inconsistent formatting, and duplicate records can reduce data quality and lead to unreliable results. Advanced cleaning ensures the dataset is fully prepared for downstream tasks.

Cleaning Process

- Text Standardization: All string columns are processed to remove non-alphanumeric characters, including punctuation and symbols. Text is converted to a consistent format to reduce noise and improve the effectiveness of feature extraction techniques such as TFIDF.
- Duplicate Removal: Duplicate records are identified and dropped to ensure that each row represents unique information. This prevents bias in analysis and improves the reliability of machine learning models.

Benefits

- Improved Data Quality: Noise and inconsistencies are minimized, making the dataset more accurate and reliable.
- Better Feature Extraction: Cleaned text allows algorithms to capture meaningful patterns in the data.
- Enhanced Model Performance: Unique and standardized data improves the accuracy and interpretability of models.
- Reproducibility: Clean and well-structured data ensures that analytics and machine learning workflows can be consistently repeated.

Clean Semi-Structured Data

Semi-structured data, such as JSON from social media or APIs, often contains inconsistent formatting, unwanted symbols, and duplicate records. Even after initial ETL processing, these issues can persist and affect the quality of analysis or machine learning tasks. Cleaning ensures the data is standardized, consistent, and ready for downstream processing.

Cleaning Process

The cleaning process involves removing special characters from all text columns to reduce noise and improve feature extraction, ensuring that text is easier to process for machine learning algorithms. Duplicate records are also eliminated to guarantee that each row represents unique information. This standardization makes the dataset more reliable, consistent, and suitable for analysis.

- Noise Reduction: Removing punctuation and symbols minimizes irrelevant information in the data.
- Unique Records: Dropping duplicates ensures unbiased analysis and model training.
- Enhanced Feature Extraction: Clean text improves the effectiveness of TF-IDF, sentiment analysis, and other NLP techniques.
- Data Consistency: Standardized semi-structured data is easier to process and analyze.

Merging

Combine cleaned structured and semi-structured datasets into a single unified dataset. Merging allows for comprehensive analysis across different data sources, enabling richer insights and more robust machine learning applications.

Merging Process

The merging process begins by identifying all unique columns across both datasets. Columns missing in either dataset are added with placeholder values (e.g., nulls) to ensure alignment. Once the columns are aligned, the datasets are combined using a union operation, preserving all rows from both sources. The structured data is loaded from "*hdfs://localhost:9000/processed/structured_clean*", and the semi-structured data is loaded from "*hdfs://localhost:9000/processed/semi_clean_final*". After merging, the combined dataset is

stored temporarily in HDFS at "*hdfs://localhost:9000/processed/merged_temp*" to be used in subsequent processing or final storage steps.

- Unified Dataset: Combines multiple data sources into one cohesive dataset for analysis.
- Consistency: Aligning columns ensures compatibility and prevents schema mismatches.
- Comprehensive Analysis: Enables richer insights by leveraging data from both structured and semi-structured sources.
- Ready for Downstream Tasks: The temporarily merged dataset can now be used for feature extraction, sentiment analysis, or predictive modeling.
- Traceability: The source of each dataset is preserved, ensuring clarity on where data originated and where it is temporarily stored.

Store Processed Data

Once the structured and semi-structured data have been cleaned, merged, and optionally enriched, it is essential to store the final dataset in a permanent location. This ensures that all downstream analytics, feature engineering, and machine learning tasks use a consistent and reliable version of the data.

Storage Process

The final processed dataset is loaded from the temporary or intermediate storage location, "*hdfs://localhost:9000/processed/final_enriched_data*". After verification, it is written to a permanent HDFS location at "*hdfs://localhost:9000/processed/final_dataset*". This step ensures that the dataset is persistently stored, organized, and ready for future use without needing to repeat previous cleaning or merging steps.

- Permanent Storage: Safely saves the processed dataset for consistent access.
- Reproducibility: Enables reliable re-use for analytics or machine learning pipelines.
- Efficiency: Avoids redundant processing since all cleaning, merging, and enrichment steps are already completed.
- Ready for Deployment: The final dataset can now be directly used for feature extraction, model training, or reporting.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the process of examining and understanding a dataset before performing advanced analysis or building machine learning models. It helps to discover patterns, spot errors or missing values, understand distributions, and summarize important characteristics of the data.

Dataset Loading and Schema Overview

The merged dataset is loaded from the temporary storage location "*hdfs://localhost:9000/processed/merged_temp*". The schema is displayed to understand column types, and a sample of rows is previewed to inspect the data content and format.

Row Count

The total number of rows in the merged dataset is calculated to assess the dataset size and confirm successful merging.

Null and Missing Value Analysis

Missing values are identified for both numeric and non-numeric columns. For numeric columns, nulls and NaNs are counted, while for non-numeric columns, only nulls are considered. This step helps detect incomplete or inconsistent data that may require imputation or removal.

Top Values in Text Columns

For textual data, the most frequent values in each column are analyzed. This provides insight into common entries, repeated patterns, or potential categorical variables. Examining frequent text values also helps identify noise, common phrases, or outliers that may affect subsequent analysis.

Basic Statistics for Numeric Columns

Numeric columns are summarized using descriptive statistics such as mean, minimum, maximum, and standard deviation. This analysis provides a high-level understanding of the distribution, scale, and variability of numeric features.

Early BI visuals

Load Final Dataset

The final merged dataset, which has been cleaned and processed, is loaded from HDFS at the path "*hdfs://localhost:9000/processed/final_dataset*". Inspecting the dataset begins with checking its schema to understand each column's data type and structure. A sample of rows is displayed to get a feel for the content and formatting of the data. Finally, the total number of rows is counted to understand the dataset's size and ensure all records have been successfully merged and stored.

EDA Missing

This step focuses on identifying data quality issues by checking for missing or empty values in the final processed dataset loaded from "*hdfs://localhost:9000/processed/final_dataset*". For each column, the code counts how many records contain either null values or empty strings, which are common indicators of incomplete data.

EDA Most Frequent Text

This step analyzes text-based columns in the final processed dataset loaded from "*hdfs://localhost:9000/processed/final_dataset*". First, all columns with a string data type are identified. From these, one text column is selected and grouped to count how often each unique value appears. The results are then sorted in descending order to display the most frequent values.

```
%run eda/eda_text_top.py
```

```
=== Most Frequent Values in 'clean_text' ===
+--------------------------------------------------------------------------------------------------------------------+------+
|clean_text                                                                                                          |count |
+--------------------------------------------------------------------------------------------------------------------+------+
|NULL                                                                                                                |156615|
|deleted                                                                                                             |142285|
|                                                                                                                    |5469  |
|twitter for android                                                                                                 |4601  |
|twitter web app                                                                                                     |4528  |
|tweetdeck                                                                                                           |618   |
|no                                                                                                                  |601   |
|yes                                                                                                                 |572   |
|covid                                                                                                               |555   |
|help slow the spread of covid and identify at risk cases sooner by selfreporting your symptoms daily even if y|467   |
+--------------------------------------------------------------------------------------------------------------------+------+
only showing top 10 rows
```

Figure 1 EDA top text

EDA Numeric Data

This step focuses on understanding the numeric features in the final processed dataset loaded from "*hdfs://localhost:9000/processed/final_dataset*". All columns with numeric data types are first identified. For these columns, descriptive statistics such as count, mean, standard deviation, minimum, and maximum values are computed.

```
%run eda/eda_numeric_summary.py
```

```
=== Numeric Columns Summary ===
+-------+------------------+------+--------------------+------+------------------+--------------------+
|summary|             score| downs|    controversiality|gilded|               ups|        retrieved_on|
+-------+------------------+------+--------------------+------+------------------+--------------------+
|  count|            850359|850359|              850359|850359|            850359|              850359|
|   mean|3.6510438532431597|   0.0|1.175973912194731...|   0.0|3.6510438532431597|1.4269040219341173E9|
| stddev|12.983954383203987|   0.0|0.001084423308581445|   0.0|12.983954383203987|  1147188.0999053488|
|    min|              -594|     0|                   0|     0|              -594|          1425904700|
|    max|              1812|     0|                   1|     0|              1812|          1435784177|
+-------+------------------+------+--------------------+------+------------------+--------------------+
```

Figure 2 EDA numeric

EDA Data Visualization

This step focuses on visually exploring the final processed dataset to better understand patterns and distributions that are not easily seen in tables. The dataset is first loaded from HDFS and converted from Spark DataFrame to Pandas because visualization libraries work efficiently with Pandas data. This conversion is done only when the dataset size is manageable.

For text data, a bar chart is created to show the top 10 most frequent values in a selected text column. This helps identify dominant categories, common terms, or repeated patterns in the data. For numeric data, histograms with density curves are plotted to visualize the distribution of values, making it easier to spot skewness, outliers, or unusual ranges.

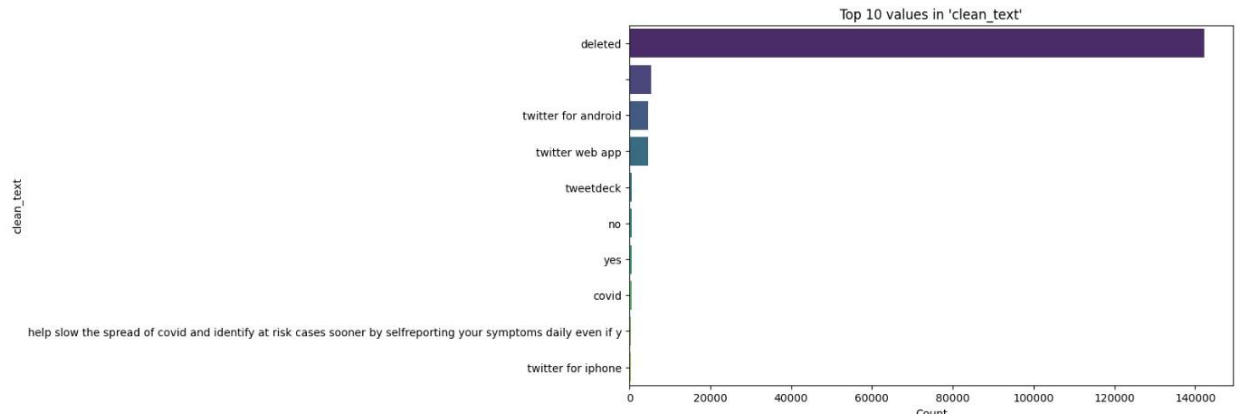

Figure 3 Top 10 values

Dashboard Report

Dashboard Report performs exploratory data analysis (EDA) and creates a basic dashboard-style analysis using the final processed dataset stored in HDFS. Its main goal is to understand the data clearly and generate visual outputs that can be used for reporting or submission.

Data Loading and Preparation

The final dataset is loaded from "*hdfs://localhost:9000/processed/final_dataset*" using Spark and then converted into a Pandas DataFrame for easier visualization. An output folder is created to store all generated plots, making the results organized and reusable.

Basic Dataset Overview

The script prints basic information such as the number of rows and columns, column names, and missing values per column. This gives a quick understanding of the dataset size, structure, and data quality.

Numeric Data Analysis

All numeric columns are identified and summarized using descriptive statistics like mean, minimum, maximum, and standard deviation. Histograms are then created for each numeric column to visualize data distribution and detect skewness or outliers. These plots are saved for later use.

Categorical and Text Data Analysis

Text or categorical columns are analyzed by showing the most frequent values. Bar charts are generated for the top values in each categorical column to highlight dominant categories or repeated patterns in the data. These visuals help in understanding trends and class imbalance.

Correlation Analysis

For numeric columns, a correlation heatmap is generated to show relationships between features. This helps identify strongly related variables, which is useful for feature selection and modeling decisions.

Output

All plots are saved into a dedicated folder, making this script suitable as a dashboard draft.

Select ML algorithms

Three machine learning algorithms were selected to perform sentiment analysis on social media text data. Social media data is usually short, informal, noisy, and high-dimensional after text vectorization. Therefore, the selected models needed to be efficient, reliable, and well-suited for text classification tasks. The three chosen models are Logistic Regression, Multinomial Naive Bayes, and Linear Support Vector Machine (SVM).

Logistic Regression

Logistic Regression is one of the most widely used algorithms for text classification and sentiment analysis. Although its name includes the word “regression,” it is actually a classification algorithm. It works by learning the relationship between input features (TF-IDF features extracted from social media text) and the sentiment labels.

Logistic Regression is effective because it handles high-dimensional sparse data very well. Text data converted into TF-IDF (Term Frequency – Inverse Document Frequency) vectors often contains thousands of features, and Logistic Regression can process these efficiently. The model estimates the probability that a given text belongs to a particular sentiment class (positive, negative, or neutral) and assigns the class with the highest probability. The influence of individual words on sentiment prediction can be understood by examining model weights. This

makes it useful for analyzing which terms contribute most to positive or negative sentiment in social media posts.

Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic machine learning model based on Bayes' Theorem. It is especially popular for text classification tasks such as spam detection and sentiment analysis. The model assumes that the presence of one word in a document is independent of the presence of other words, which is known as the "naive" assumption. Multinomial Naive Bayes performs well because it works naturally with word frequency and TF-IDF features. It is fast to train, requires less computational power, and performs efficiently even on large datasets. This makes it suitable for big data environments and real-time sentiment analysis.

One of the key strengths of Naive Bayes is its simplicity and speed. It can quickly learn patterns in text data and produce reliable results with minimal tuning. For social media data, where posts are short and vocabulary can be large, this efficiency is a major advantage.

Linear Support Vector Machine (SVM)

Linear Support Vector Machine is a powerful classification algorithm that aims to find the best decision boundary between different sentiment classes. It works by maximizing the margin between data points of different classes, which often leads to strong generalization performance. Linear SVM is highly effective because it handles high-dimensional text features extremely well. When text is transformed using TF-IDF, the feature space becomes very large, and Linear SVM is designed to work efficiently in such environments. one of the major advantages of Linear SVM is its ability to reduce overfitting. By focusing on the most important data points (support vectors), the model becomes robust to noise, which is very common in social media text such as slang, abbreviations, and misspellings.

Feature engineering

Feature engineering is a crucial step in any machine learning pipeline, especially when dealing with text data from social media platforms. Social media text is typically noisy, informal, and inconsistent, containing slang, abbreviations, hashtags, emojis, and misspellings. Without proper

preprocessing, these issues can negatively impact model performance. Feature engineering converts raw text into a structured format suitable for machine learning algorithms and improves the quality of predictions.

Data Cleaning

The first step in feature engineering is data cleaning. This involves several tasks to prepare raw social media text for analysis:

1. Lowercasing: All text is converted to lowercase to ensure that words like “Happy” and “happy” are treated as the same token. This reduces redundancy and improves model consistency.

2. Removing Non-Alphabetic Characters: Punctuation, numbers, special symbols, and emojis are removed, leaving only meaningful words. This step helps to reduce noise in the dataset while preserving the essential content.

3. Stopword Removal: Commonly used words such as “the,” “is,” “and,” and “a” do not carry significant sentiment information. These stopwords are removed from the text to focus on words that contribute to sentiment. Eliminating stopwords reduces feature dimensionality and improves the relevance of extracted features.

Feature Transformation: TF-IDF

Once the text is cleaned, the next step is to convert it into numerical features that machine learning algorithms can process. One of the most effective techniques for text data is Term Frequency-Inverse Document Frequency (TF-IDF).

TF-IDF captures how important a word is in a document relative to the entire corpus. Words that appear frequently in a single document but rarely in other documents are given higher importance. This helps the model focus on terms that are more likely to indicate sentiment rather than common words that appear across all texts.

- Positive words like “love,” “great,” or “amazing” gain higher scores in posts with positive sentiment.
- Negative words like “hate,” “bad,” or “terrible” are emphasized in negative posts.

- Neutral or common words are automatically downweighte, reducing their influence on predictions.

Sentiment Label Generation

The dataset requires sentiment labels. In this project, sentiment is determined based on the polarity of the cleaned text using natural language processing tools. Each post is classified into one of three categories:

- Positive sentiment: The text expresses favorable opinions or emotions.
- Negative sentiment: The text expresses discontent or unfavorable opinions.
- Neutral sentiment: The text neither strongly conveys positive nor negative emotion.

Storing Preprocessing Tools

To ensure reproducibility and consistency across training, testing, and deployment stages, the TF-IDF vectorizer used to transform text into numerical features is stored as a reusable object. This allows new social media posts to be converted using the same feature mapping, ensuring that models make predictions based on the same learned feature space.

Training Initial Models

After completing feature engineering and generating TF-IDF vectors from social media text, the next crucial step in the machine learning pipeline is training the models. Model training is the process by which algorithms learn patterns from the labeled dataset, allowing them to predict sentiment for unseen posts.

Preparing Data for Training

Before training the dataset is divided into two subsets:

1. Training Set: This portion of the data is used by the models to learn the relationship between input features (TF-IDF) and sentiment labels. The models examine patterns in word usage and how specific words are associated with positive, negative, or neutral sentiment.
2. Testing Set: A separate portion of the data is reserved for later evaluation. This ensures that the models are tested on unseen data, allowing assessment of their generalization

ability. Typically, an 80/20 split is used, with 80% of the data for training and 20% for testing.

Training Logistic Regression

Logistic Regression is trained on the TF-IDF features of the training set. During training, the model estimates weights for each word feature to maximize the probability that the text belongs to the correct sentiment class. Logistic Regression can efficiently handle thousands of features created by the TF-IDF transformation. The training process adjusts model parameters iteratively, ensuring that posts with positive, negative, or neutral sentiment are classified as accurately as possible. The result is a model capable of predicting sentiment probabilities for new posts.

Training Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic model that uses word frequencies to estimate the likelihood of each sentiment class. During training, the model calculates the probability of each word occurring in positive, negative, or neutral posts.

This approach is particularly well-suited for social media text, where some words strongly indicate sentiment while others are common across all posts. Multinomial Naive Bayes quickly learns these word-sentiment associations, resulting in a model that can efficiently predict sentiment for both small and large datasets. Its simplicity and speed make it an excellent choice for initial model development.

Training Linear Support Vector Machine (SVM)

Linear SVM takes a different approach by finding the optimal boundary (hyperplane) that separates different sentiment classes in the high-dimensional feature space. Using TF-IDF vectors, each post becomes a point in this space, and the model seeks the hyperplane that maximizes the margin between classes.

This training method allows Linear SVM to handle high-dimensional data effectively and reduces the risk of overfitting. The model learns which features (words) are most significant for distinguishing between positive, negative, and neutral sentiment. Although training Linear SVM can require more computational resources than Logistic Regression or Naive Bayes, it often achieves strong performance on text classification tasks.

Model Storage for Future Use

After training, the models are saved as reusable objects. Storing trained models allows them to be loaded later without retraining, making the pipeline efficient and reproducible. This is especially important for social media applications, where new posts are continuously generated and predictions need to be made in real time.

Each model - Logistic Regression, Multinomial Naive Bayes, and Linear SVM—is stored in a structured directory for easy access. This ensures that the machine learning pipeline can be executed seamlessly during deployment, with consistent preprocessing, feature extraction, and prediction steps.

Importance of Training Multiple Models

Training multiple models provides several benefits for social media sentiment analysis:

- Comparison: By training different algorithms, we can later compare performance and choose the most suitable model for the final pipeline.
- Robustness: Different models capture different patterns in the data, increasing the chance of finding an accurate solution.
- Efficiency: Some models, like Naive Bayes, are extremely fast, while others, like SVM, may provide higher accuracy. Having multiple trained models allows flexibility in balancing speed and performance.

Evaluating Performance

Proper evaluation is crucial to ensure that the models are reliable, accurate, and suitable for realworld social media applications.

Preparing for Evaluation

Before evaluation the dataset is divided into training and testing sets. The testing set consists of posts that the models have not seen during training. Using unseen data allows for a fair

assessment of the model's generalization ability its capacity to make correct predictions on new social media content.

The TF-IDF features created during feature engineering are applied to the testing data to maintain consistency. The trained models - Logistic Regression, Multinomial Naive Bayes, and Linear SVM—are then used to predict sentiment labels for each post in the testing set.

Compare algorithms

By evaluating multiple models on the same testing set, we can determine which model performs best overall and for each sentiment category:

- Logistic Regression offers a balance between interpretability and performance.
- Multinomial Naive Bayes is often very fast and performs well with sparse text data.
- Linear SVM typically provides strong classification boundaries in high-dimensional feature space, though it may require more computational resources.

Comparing Machine Learning Models

Comparing models allows us to understand the strengths and weaknesses of each approach and to select the most appropriate model for deployment in real-world applications.

Purpose of Model Comparison

Social media text data is diverse and challenging. Users often express opinions in informal language, including slang, abbreviations, emojis, and sarcasm. Therefore, different machine learning algorithms may perform differently depending on how they handle high-dimensional TF-IDF features and subtle sentiment cues. Comparing models helps to:

- Identify which model achieves the highest overall accuracy.
- Determine which model performs best for specific sentiment classes (positive, negative, neutral).
- Understand the trade-offs between simplicity, speed, and predictive performance.
- Decide which model is best suited for deployment and scalability.

Insights from Model Comparison

1. Logistic Regression

    - Provides consistent performance across all sentiment classes.
    - Balances simplicity with interpretability, allowing understanding of which words influence sentiment predictions.
    - May slightly underperform on more nuanced or highly imbalanced data.

2. Multinomial Naive Bayes

    - Extremely fast to train and predict, making it suitable for large-scale social media datasets.
    - Performs well when word frequencies strongly correlate with sentiment.
    - Its simplicity sometimes limits performance on posts with complex or subtle sentiment.

3. Linear Support Vector Machine (SVM)

    - Often achieves the highest accuracy and F1-scores due to its ability to separate classes with a clear margin.
    - Robust to high-dimensional TF-IDF features common in social media text.
    - Requires more computational resources and longer training time compared to simpler models.

Visualization of Comparison

Comparison is often visualized with:

- Bar charts showing accuracy and weighted F1-score for each model.
- Side-by-side confusion matrices for each sentiment class.

These visualizations make it easier to quickly understand which model performs best overall and for each type of sentiment.

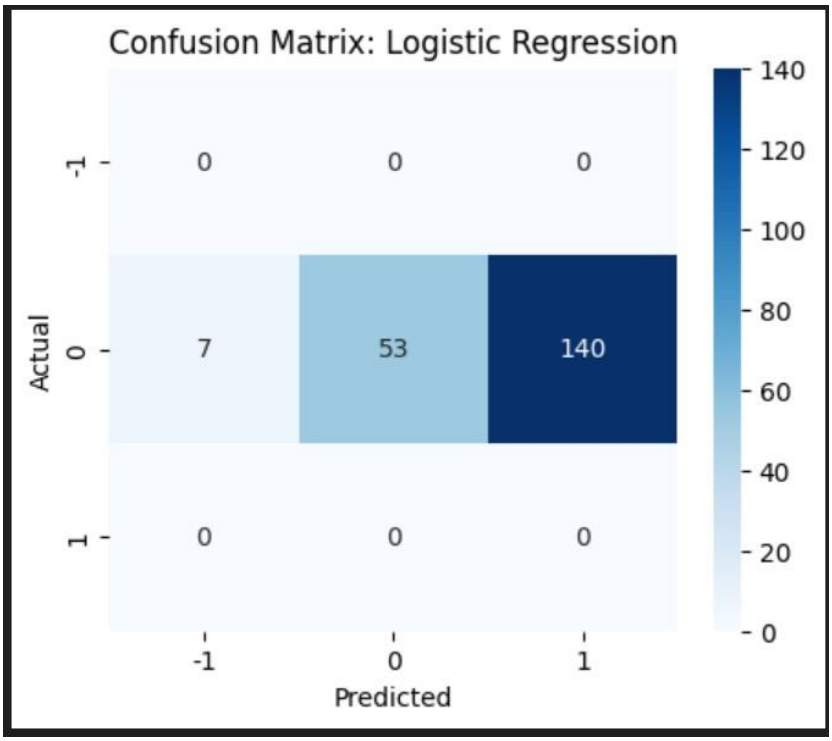

Figure 4 Logistic Regression Evaluation

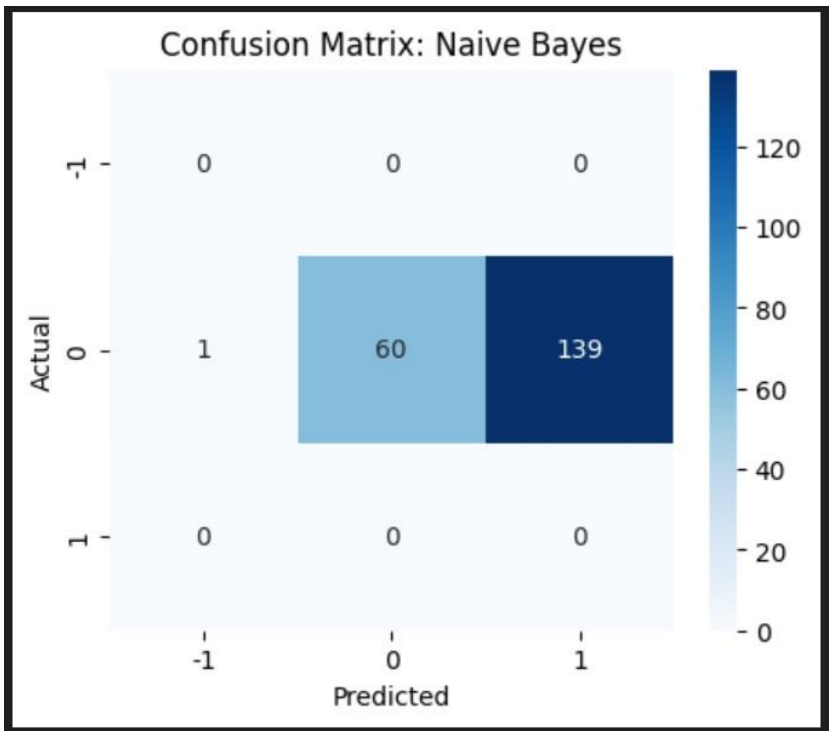

Figure 5 Naive Bayes Evaluation

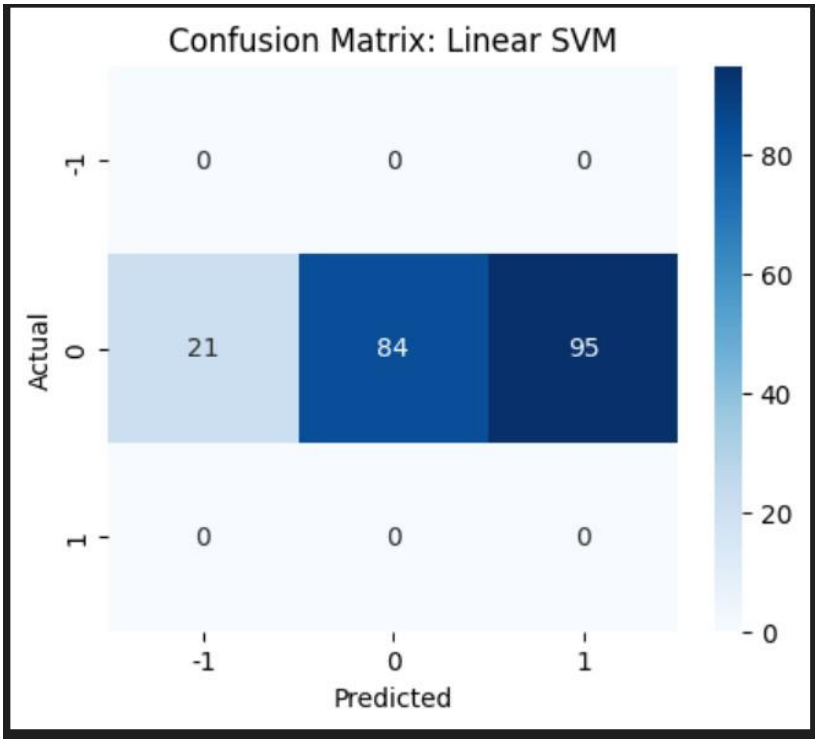

Figure 6 Linear SVM Evaluation

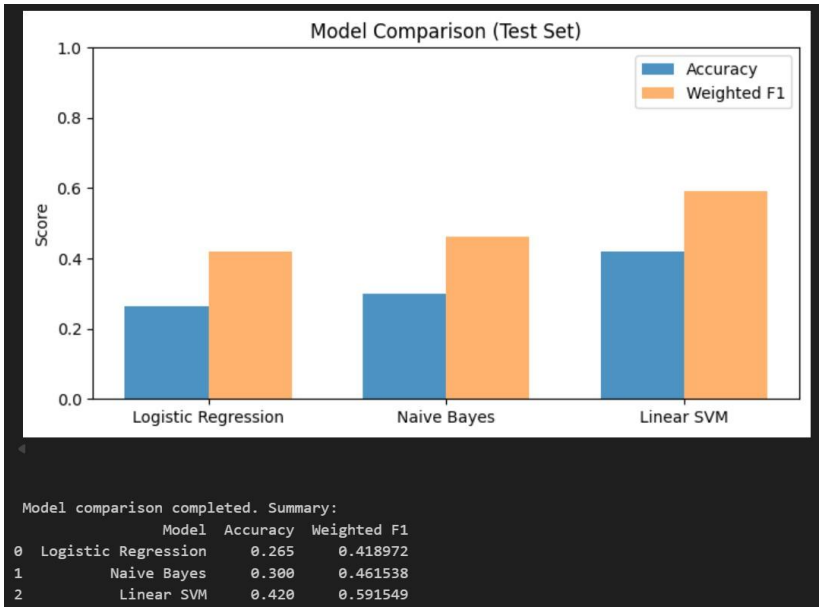

Figure 7 Comparison

Finalizing the Machine Learning Pipeline

The pipeline is a structured workflow that ensures the entire process - from raw social media text to sentiment prediction - is reproducible, efficient, and deployable.

Organizing Data and Artifacts

The finalized pipeline begins by organizing all input data and output artifacts. The cleaned social media dataset, along with computed sentiment labels, is stored in a centralized location, such as a distributed file system like HDFS. This ensures that all subsequent steps can access the same data consistently.

Saving intermediate datasets and features is important because:

- It allows the pipeline to be reused for new data without repeating initial processing steps.
- It improves reproducibility by maintaining a snapshot of the cleaned and preprocessed dataset.
- It facilitates collaboration, allowing multiple team members or systems to access the same structured data.

Feature Engineering Integration

Feature engineering is integrated into the pipeline to transform raw social media posts into numerical features suitable for machine learning. This involves:

- Text cleaning: Lowercasing, removing non-alphabetic characters, and eliminating stopwords.
- Vectorization: Converting cleaned text into TF-IDF features, which capture the importance of each word relative to the corpus.

The TF-IDF vectorizer is saved as a reusable object, ensuring that new posts can be transformed consistently using the same feature space. This guarantees that predictions on new social media data are accurate and aligned with the training process.

Model Training within the Pipeline

The finalized pipeline includes the training of multiple machine learning models. In this case:

- Logistic Regression: Efficient and interpretable, suitable for general sentiment classification.
- Multinomial Naive Bayes: Fast and effective for word frequency-based classification.
- Linear SVM: Robust for high-dimensional TF-IDF features and capable of handling complex decision boundaries.

Each model is trained on the processed dataset and then saved as an artifact. Storing models ensures that they can be reloaded for evaluation, prediction, or deployment without retraining.

Centralized Storage of Artifacts

All critical artifacts - processed datasets, TF-IDF vectorizers, and trained models - are saved in a centralized folder structure, typically in HDFS or a local storage system. Centralization ensures:

- Ease of deployment: Models and vectorizers are ready for integration into applications.
- Reproducibility: The same pipeline can be rerun with consistent results.
- Maintainability: Artifacts can be versioned and updated without affecting other parts of the pipeline.

Benefits of a Finalized Pipeline

1. Reproducibility: Every step, from preprocessing to model training, can be executed consistently.
2. Scalability: The pipeline can handle increasing volumes of social media data without manual intervention.
3. Efficiency: Automated storage and reuse of intermediate artifacts reduce computation time.
4. Deployment-Readiness: Trained models and vectorizers are ready to be deployed for realtime sentiment analysis.

User Interface (UI)

The sentiment analysis user interface (UI) was developed using Streamlit, providing an interactive platform for real-time prediction of social media text sentiment. The application

allows users to input text, such as comments or posts, into a text area and click a button to receive a sentiment classification. The underlying pipeline relies on a previously trained Linear Support Vector Machine (SVM) model, which uses TF-IDF vectorization to transform textual input into numerical features suitable for classification. The TF-IDF vectorizer is stored at *ml/submit_dashboard/models/tfidf_vectorizer.pkl*, and the trained SVM model is stored at *ml/submit_dashboard/models/model_svm_final.pkl*. Once a user enters text, the input is transformed using the vectorizer, and the model predicts one of three sentiment categories: Positive , Neutral , or Negative . The UI displays the prediction prominently using intuitive color-coded feedback (green for positive, gray for neutral, red for negative), enhancing user experience.

To run the application, first change the working directory to the folder containing the app (where app.py is located), then execute the command: streamlit run app.py

This launches the interactive Streamlit interface in a web browser, allowing users to enter text and see sentiment predictions instantly. This approach ensures a reproducible, efficient, and userfriendly interface for analyzing sentiment in social media data, bridging the gap between complex machine learning models and end-user interaction.

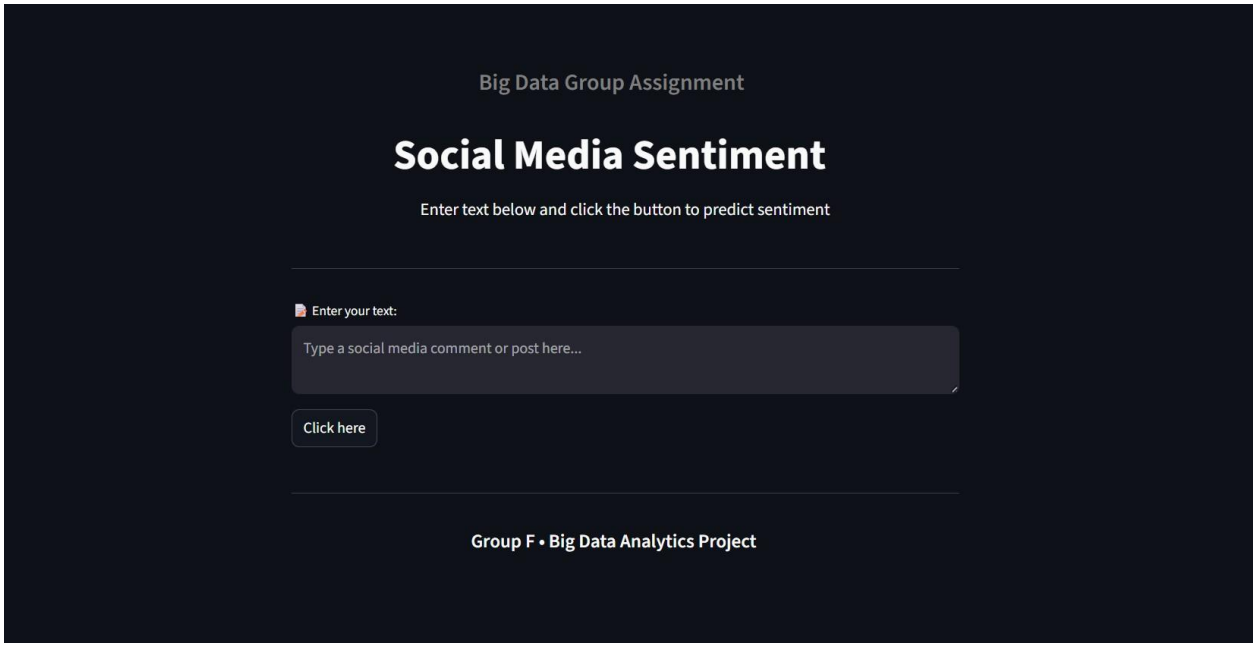

Figure 8 User Interface (UI)

Conclusion

This project successfully demonstrates the development of a comprehensive big data analytics and machine learning pipeline for social media sentiment analysis. Leveraging Apache Spark and HDFS, large-scale structured and semi-structured data was efficiently ingested, cleaned, transformed, and merged into a unified dataset suitable for analysis. Exploratory Data Analysis provided insights into data quality, distribution, and patterns, ensuring a reliable foundation for machine learning.

Feature engineering using TF-IDF transformed informal, noisy social media text into meaningful numerical features, enabling accurate model training. Three machine learning models - Logistic Regression, Multinomial Naive Bayes, and Linear SVM - were trained and evaluated. Logistic Regression offered interpretability and balanced performance, Multinomial Naive Bayes provided speed and efficiency, and Linear SVM demonstrated strong accuracy in highdimensional feature spaces.