BIG DATA ANALYTICS

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Real-Time Fake News Detection Using PySpark

Group 02

Tahiya Jahan Laboni (20240943)
Antonio Ramos (20240561)
Eden da Silva (20240740)
Joao Cardoso(20240529)

24/25 PROJECT STATEMENT

TABLE OF CONTENTS

1. Introduction(Problem statement)

Fake news is a growing problem in today's digital world. It spreads quickly on the internet, misleads people, and affects public trust. Detecting fake news at scale is difficult because of the large volume and speed of online content.

In this project, we built a system to detect fake news using Apache Spark on Databricks. We trained a machine learning model using labeled datasets of real and fake news. Then, we simulated a real-time environment where new statements are automatically checked and classified as real or fake. This project shows how big data tools can be used to solve real problems in a streaming setup.

2. Dataset Overview

We used two labeled datasets: Fake.csv (containing fabricated news articles) and True.csv (containing factual news). Each dataset included a text column representing the news content. Labels were assigned as **0** for fake and **1** for real.

After cleaning and merging, the final dataset included **23,481** fake samples and **21,417** real samples—providing a balanced and reliable foundation for training a supervised classification model.

3. Data Preprocessing WorkFlow

First, we implemented a data processing pipeline using PySpark in Databricks and involved multiple stages of preparation to ensure the textual data was clean, well-structured, and ready for machine learning.

3.1 Data Loading and Labeling

We began by loading two CSV files—**Fake.csv** and **True.csv**—from the *`/FileStore/tables/`* directory. The **Fake.csv** file contained fabricated news articles, while the **True.csv** file included factual news. To facilitate supervised learning, each dataset was assigned a binary label: fake news was labeled as **0** and real news as **1**. After labeling, the datasets were merged using the *`unionByName()`* function, and only the relevant columns, text and label, were retained for further processing. Additionally, any rows with null or missing values were removed using the *`.na.drop()`* method to ensure data quality.

3.2 Train-Test Splitting

To prepare the dataset for model training and evaluation, the combined data was randomly split into two subsets: 80% of the data was allocated to the training set, and the remaining 20% was reserved for testing. This split was performed using the **randomSplit([0.8, 0.2], seed=42)**

function in PySpark, which ensures consistent results across runs by setting a fixed random seed for reproducibility.

3.3 Text Preprocessing Pipeline

To prepare the textual data for machine learning, we developed a Spark ML pipeline with several sequential preprocessing stages. First, we used **Tokenizer** to split sentences into individual words by setting *inputCol='text'* and *outputCol='words'*. Next, the **StopWordsRemover** removed common stopwords to eliminate noise, with *inputCol='words'* and *outputCol='filtered'*. The filtered tokens were then transformed into fixed-length feature vectors using **HashingTF**, configured with *inputCol='filtered'* and *outputCol='rawFeatures'*. Finally, we applied **IDF** (Inverse Document Frequency) to scale the features based on their importance across the dataset, using *inputCol='rawFeatures'* and *outputCol='features'*. This pipeline ensured that the text data was appropriately vectorized for classification tasks.

3.4 Model Input Preparation

The final output from the preprocessing pipeline was a column named **features**, which represented the transformed text data in numerical format and served as the input for training machine learning models. Alongside this, the ***label*** column, which indicated whether a news sample was fake **(0)** or real **(1)**, was used as the target variable for classification.

4. ALGORITHM USED

To classify assertive statements as fake or real, we experimented with two supervised machine learning algorithms using PySpark MLlib: **Random Forest** and **Naive Bayes**.

4.1 Random Forest Classifier

We used the **Random Forest Classifier** to build multiple decision trees and aggregate their outputs to improve prediction accuracy and robustness. For our project, we configured the model with 10 trees and a maximum depth of 4 to avoid overfitting. We trained the model on an 80/20 split of the dataset and evaluated it using two metrics: **Area Under the ROC Curve (AUC)** and **accuracy**. The model achieved an AUC score of **0.8611** and an accuracy of **73.38%**, indicating reliable and consistent performance across both evaluation criteria. Due to its strong generalization capability and stable behavior, we ultimately selected the Random Forest as the final model for deployment in the real-time streaming environment.

4.2 Naive Bayes Classifier

We also tested a **Multinomial Naive Bayes** classifier, commonly used in text classification tasks. While it provided a very high test accuracy of **98.26%**, its AUC score was only **0.5428**, indicating poor ability to distinguish between fake and real news in probabilistic terms.

5. Results and insights

5.1 Offline Evaluation

To evaluate the performance of our classifiers, we used two standard metrics: Area Under the ROC Curve (AUC) and Accuracy.

Model	AUC	Accuracy
Random Forest	0.8611	0.7338
Naive Bayes	0.5428	0.9826*

Key Insights: The high accuracy of Naive Bayes is misleading due to biased predictions. **Confusion Matrix Analysis:** To understand model behavior more granularly, we analyzed the confusion matrix results:

- *True Positives (Real news correctly classified): 2190*
- *True Negatives (Fake news correctly classified): 4276*
- *False Positives (Fake news misclassified as real): 293*
- *False Negatives (Real news misclassified as fake): 2053*

This indicates the model performs well in identifying fake news but has more difficulty classifying real news accurately.

Model Selection: Given its robustness and consistent behavior across multiple trials, We ultimately selected Random Forest as the final model for deployment. Its stable AUC and balanced accuracy make it more suitable for real-world, real-time streaming scenarios, where interpretability and generalization are critical.

5.2 Real-Time Inference & Streaming Evaluation

To simulate real-time inference, we loaded the trained and saved Random Forest pipeline model using `*PipelineModel.load()*` from the Databricks file system. This model, previously trained and evaluated offline, was then applied to new incoming streaming data.

The streaming data was read from a CSV file **(stream1.csv)** stored in */FileStore/tables/*, containing unseen text news headlines. After ensuring data cleanliness by removing null entries, our model was applied to make predictions on each record. A timestamp was appended using `*current_timestamp()*` to mark the prediction time. The model predictions were stored in persistent storage as a CSV file at `*/FileStore/tables/stream_results*`.

To analyze streaming results, a temporary SQL view **(stream_results)** was created, and a Spark SQL query was executed to count predictions grouped by label.

6. Challenges and Future Improvements

6.1 Challenges Encountered

While deploying the final model on streaming data, we encountered a key challenge: the model consistently labeled all incoming text as fake. This behavior revealed a residual bias in the model, potentially caused by limitations in feature representation. Specifically, the use of **HashingTF** may have led to information loss and a lack of contextual understanding, prompting consideration of alternatives like *CountVectorizer* or more expressive classifiers such as **Logistic Regression**.

Another challenge was making sure the pipeline worked smoothly with real-time data. Since we were working with streaming inputs and large data, we had to make sure the system could handle the load, run without errors, and save the results properly.

Even with these problems, the system worked well overall and showed it could handle real-time predictions in a Big Data setup.

6.2 Planned Future Improvements

To enhance our model performance and accuracy, we plan to:

- Replace **HashingTF** with **CountVectorizer** for better text representation.
- Experiment with advanced models like **Logistic Regression** or **BERT**.
- Integrate **real-time data sources** (e.g., Twitter, RSS feeds).
- Develop **live dashboards** for monitoring predictions and alerts.