**Real-Time Streaming with Kafka, Bloom Filters, and Machine Learning**

**Overview**

This document describes the approach of integrating Kafka, Bloom Filters, and a Machine Learning (ML) model for real-time streaming and classification. The process involves training a Bloom Filter and an ML model using historical data stored in a CSV file and then applying them to classify new streaming data efficiently.

**1. Data Preprocessing**

**Step 1: Loading and Preparing Data**

* The dataset (EHR\_RISK\_ESTIMATION.csv) contains historical medical records, including various patient attributes and a CANCER label indicating whether a patient was diagnosed with cancer.
* Data preprocessing includes cleaning, handling missing values, and ensuring consistency.

**2. Training and Storing the Bloom Filter**

**Step 2: Initializing the Bloom Filter**

* A **Bloom Filter** is used for efficient membership checking, determining whether an incoming data point has been seen before.
* The filter is trained on the CSV dataset using only rows where CANCER = 1 (indicating positive cases).

**Step 3: Generating Unique Keys**

* Each row in the dataset is converted into a unique key using the following features:
* def generate\_key(rows):

return f"{rows['menopause']}\_{rows['age\_group']}\_{rows['density']}\_{rows['race']}\_{rows['Hispanic']}\_{rows['BMI']}\_{rows['Age\_First']}\_{rows['NRELBC']}\_{rows['BRSTPROC']}\_{rows['LASTMAMM']}\_{rows['SURGMENO']}\_{rows['HRT']}"

* The keys are then inserted into the Bloom Filter.

**Step 4: Saving the Bloom Filter**

* The trained Bloom Filter is stored using pickle for later use:
* with open("bloom\_filter.pkl", "wb") as f:

pickle.dump(bloomfilter, f)

**3. Training the Machine Learning Model**

**Step 5: Selecting the Best Model**

* Different ML models were evaluated to predict cancer risk based on historical data.
* **XGBoost** was determined to be the best-performing model.

**Step 6: Training and Saving the ML Model**

* The ML model is trained using the relevant features and labels.
* The trained model is saved using joblib:

joblib.dump(best\_model, 'model.pkl')

**4. Kafka-Based Real-Time Data Streaming**

**Step 7: Data Generation and Kafka Producer**

* A Kafka producer generates synthetic patient records or streams real data from the CSV file and sends it to a Kafka topic (ehr\_data\_topic).
* **Synthetic Data Generation**:

producer.send('ehr\_data\_topic', data)

**Step 8: Kafka Consumer and Spark Streaming**

* A **PySpark Streaming Application** reads data from the Kafka topic.
* The incoming messages are processed, converted to structured data, and checked against the Bloom Filter.

**Step 9: Applying the Bloom Filter and ML Model**

* The streaming data is checked against the **Bloom Filter**:
* if unique\_key not in bloom\_filter:

return 0 # Not found in the dataset

* If a possible match is found (1 returned by the Bloom Filter), the **ML model** is applied for further classification:

prediction = best\_model.predict([features])

* The final classification result is added as a new column and stored for analysis.

**Step 10: Writing Results to Console**

* The processed data (original message + prediction) is displayed in the console.

df.select("value", "final\_prediction").writeStream.outputMode("append").format("console").start()

**5. Conclusion**

* This approach efficiently handles real-time streaming data using **Kafka**.
* **Bloom Filters** optimize processing by filtering out irrelevant data, reducing computational overhead.
* **XGBoost** ensures high classification accuracy for cancer risk assessment.
* **Spark Streaming** enables scalable and real-time analytics, making this system ideal for real-world medical data processing.

This document serves as a comprehensive guide to implementing a real-time anomaly detection system using Kafka, Bloom Filters, and Machine Learning. 🚀