Fraud Detection System for Banking Transactions

1. Algorithm Justification: What, why, and Suitability

Overview of Chosen Algorithms

- **K-means Clustering (Unsupervised)**
- **Random Forest (Supervised)**
- **Logistic Regression (Supervised)**
- **Gradient Boosted Trees (Supervised)**

Why Only These Algorithms?

- **K-means Clustering:**
    - Identifies outliers and patterns in data without requiring fraud labels.
    - Enables the discovery of new, previously unseen types of fraud.
    - Works efficiently with large datasets in Apache Spark, making it scalable for banking data.
- **Random Forest:**
    - Offers high accuracy and resilience against overfitting due to assembling multiple decision trees.
    - Handles mixed types of variables and high-dimensional data, making it ideal for diverse banking transaction features.
    - Supports feature importance analysis, helping explain predictions to compliance teams.
- **Logistic Regression:**
    - Provides clear, interpretable probabilistic outputs—essential for regulatory compliance and explaining model behavior to stakeholders.
    - Fast training and inference, especially valuable for very large, sparse datasets.
    - Suitable as a baseline model for benchmarking more complex approaches.
- **Gradient Boosted Trees:**

    - Excels at detecting complex, nonlinear fraud patterns by sequentially combining weak learners.
    - Typically achieves very high predictive performance (precision, recall, AUC) on fraud datasets.
    - Well-supported and optimized in Spark MLlib for scalable model training.

Suitability for This Project

- All algorithms are available and optimized for distributed processing with Apache Spark, crucial for handling millions of transactions.
- The combination provides a full spectrum for both **anomaly detection** (K-means) and **fraud classification** (other three models).
- Real-time model scoring is possible via Spark Streaming and distributed ML pipelines.
- These models lend themselves well to feature importance analysis, threshold tuning, and integration with real-world production alerting systems.

2. Source Code Reference (Overview & Sample)

Architecture & Implementation Outline

- **Data Ingestion**
    - Spark or Hadoop reads data sources in parallel for spatial and temporal scalability.
    - Transaction features: date, code, amount, and engineered features (e.g., time since last transaction).
- **Preprocessing & Feature Engineering**
    - Handling missing/null values and standardizing formats.
    - Feature scaling and encoding (using Vector Assembler, StandardScaler in Spark).
    - Generating customer-centric features (frequency, transaction variance).
- **Model Training**
    - Modular scripts for model selection, training, and hyperparameter tuning.
    - K-means handles unsupervised clustering and assigns "anomaly" scores.

    - Supervised models (Random Forest, Logistic Regression, Gradient Boosted Trees) fit labels and predict fraud probabilities.
- **Evaluation**
    - Extensive metrics calculation: AUC, precision, recall, F1-score, confusion matrix.
    - Cross-model comparison via ROC curves.
    - Feature importance charts for interpretability.
- **Deployment & Inference**
    - Spark MLlib pipelines for quick retraining and inference.
    - Spark Streaming for live scoring in production.

Sample Spark ML Pipeline Snippet

```
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.clustering import KMeans
from pyspark.ml.classification import RandomForestClassifier, LogisticRegression,
GBTClassifier
from pyspark.ml import Pipeline

# 1. Feature Assembling and Scaling
assembler = VectorAssembler(inputCols=["amount", "hour", "day"], outputCol="features")
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")

# 2. Model Definitions
kmeans = KMeans(featuresCol="scaledFeatures", k=5)
rf = RandomForestClassifier(featuresCol="scaledFeatures", labelCol="label",
numTrees=100)
lr = LogisticRegression(featuresCol="scaledFeatures", labelCol="label")
gbt = GBTClassifier(featuresCol="scaledFeatures", labelCol="label")

# 3. Training Pipeline (example with Random Forest)
pipeline_rf = Pipeline(stages=[assembler, scaler, rf])
model_rf = pipeline_rf.fit(trainingData)
predictions_rf = model_rf.transform(testData)
```

Note: Replace `["amount", "hour", "day"]` with your actual feature names as per real dataset.

- For full source code, refer to the main GitHub repository: [github.com/karunakar00/Fraud-Detection-in-Banking-Transactions-Using-Hadoop/tree/main]

3. Comprehensive Project Documentation

Project Pipeline

Step 1: Data Loading & Exploratory Analysis

- Use Spark to ingest large-scale transaction data.
- Perform EDA to understand fraud rates, feature distributions, and temporal trends.
- Visualize data with histograms, scatter plots, and correlation matrices.

Step 2: Data Cleaning & Feature Engineering

- Remove or impute missing values.
- Standardize numeric features needed for K-means and regression.
- Engineer new features: transaction frequency, average amount per user, ratios, and time-based features.

Step 3: Model Development

Algorithm	Type	Core Strengths	Fraud Detection Role
K-means Clustering	Unsupervised	Fast, scalable. Outlier detection	Identifies novel/suspicious transactions
Random Forest	Supervised	Robust, accurate, interpretable	Main classifier for binary fraud labels
Logistic Regression	Supervised	Simple, fast, interpretable	Baseline, regulatory explanations
Gradient Boosted Trees	Supervised	Top AUC, nonlinear patterns	Captures complex, subtle fraud signatures

Step 4: Model Evaluation & Comparison

- Use cross-validation for benchmarking.
- Calculate precision, recall, F1-score, and plot ROC curves for each model.
- Interpret feature importances—highlighting which attributes most influence the fraud decision.

Step 5: Real-time Prediction & Deployment

- Integrate with Kafka or Spark Streaming for instant fraud detection.
- Send real-time alerts for flagged transactions to downstream systems (e.g., customer notification, security review).
- Monitor model performance and retrain regularly to address emerging fraud tactics.

4. Detailed Graphs & Visual Insights

- **Feature Importance Bar Chart (Random Forest):**
  Visualizes the top transaction features contributing to fraud prediction (e.g., amount, transaction time, transaction type).
- **ROC Curve Comparison:**
  Plots ROC curves for all algorithms, illustrating how well each model distinguishes fraudulent from legitimate transactions.
- **Cluster Scatter Plot (K-means):**
  Shows transaction clusters; outliers (far from centroids) may correspond to potential fraud.

5. Deployment Recommendations

- **Environment:**
  Running on Google Collab, Databricks, or any Spark/Hadoop-supported cluster.
- **Steps:**

  a. Clone the referenced repository and load the main pipeline scripts.
  b. Replace the sample dataset with real transaction data (e.g., Kaggle's `creditcard.csv`).
  c. Adjust feature engineering and pipeline stages as needed.
  d. Run the training and evaluation cells.
  e. For production, enable Spark Streaming and connect to Kafka for ingesting live transactions.

6. References

- Main Source:
  github.com/karunakar00/Fraud-Detection-in-Banking-Transactions-Using-Hadoop/tree/main

- Example Datasets:
    - Credit Card Fraud Detection, Kaggle
    - Synthetic Financial Datasets for Fraud Detection, Kaggle

P.M.D. NIHARIKA

24M11MC147

ADITYA UNIVERSITY