Leveraging Machine Learning for Securing Bank Transactions

220901019-DHAKSARAN R-EEE-A

ABSTRACT:

Fraudulent transactions are a persistent challenge in the financial world, demanding innovative solutions to safeguard users and businesses. This project harnesses the power of Decision Trees, a machine learning algorithm known for its simplicity and interpretability, to detect fraud in a synthetic dataset. Through meticulous data preprocessing, including imputation, feature engineering, and scaling, the model achieves a balance between accuracy and clarity. Despite class imbalances and missing values, the Decision Tree Classifier successfully identifies fraudulent activities, offering a promising tool for real-world applications. Visualization of the decision tree further enhances transparency, making it a reliable choice for stakeholders.

INTRODUCTION:

As digital transactions grow exponentially, detecting fraudulent activities has become a cornerstone of financial security. This project addresses this challenge by implementing a Decision Tree Classifier, a model celebrated for its visual interpretability and logical decision-making. By simulating a financial transaction dataset with inherent challenges such as missing values and class imbalances, this study replicates real-world scenarios to evaluate the effectiveness of Decision Trees in fraud detection.

DATASET OVERVIEW:

The dataset comprises 200 transactions with attributes reflecting typical financial activities:

- **TransactionID:** Unique identifier for each transaction.
- **Amount:** Transaction amount, ranging from 10 to 5000.
- **Merchant:** Vendor names (ABC, XYZ, DEF).
- **Category:** Transaction types (Shopping, Food, Travel).
- **Location:** Cities (London, Paris, New York).
- **Date:** Date and time of the transaction.
- **Is_Fraud:** Target variable indicating genuine (0) or fraudulent (1) transactions.

CHALLENGES ADDRESED:

- **Missing Values:** Introduced in 30% of key columns, mimicking real-world data issues.

- **Class Imbalance: Only** 15% of transactions labeled as fraudulent, reflecting the rarity of fraud.

METHODOLOGY:

1. Data Cleaning

- Numerical Missing Values: Replaced missing `Amount` values with the median to reduce the influence of outliers.

- Categorical Missing Values: Filled missing values in `Merchant`, `Category`, and `Location` using the most frequent value.

2. Feature Engineering

- Extracted temporal features (`Hour`, `DayOfWeek`, `Month`) from the `Date` column to capture patterns in transaction timing.

- Removed redundant columns (`TransactionID`, `Date`) to focus on meaningful attributes.

- Applied **One-Hot Encoding** to categorical variables, transforming them into numerical formats suitable for machine learning.

3. Feature Scaling

- Standardized the `Amount` feature using **StandardScaler** to ensure uniform scaling across all features.

4. Model Selection and Training

- Decision Tree Classifier:

- Chosen for its interpretability and ability to handle both numerical and categorical data.

- Configured with:

- `max_depth=5` to prevent overfitting.

- `min_samples_split=2` to ensure sufficient data at each split.

5. Model Evaluation

- Evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.

- Visualized the decision tree to interpret the decision rules.

CODE SNIPPET:

```
import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Generate a synthetic dataset with 200 transactions
data = {
    "TransactionID": range(1, 201),
    "Amount": np.random.randint(10, 5000, size=200),  # Random transaction amounts between 10 and 5000
    "Merchant": np.random.choice(['ABC', 'XYZ', 'DEF'], size=200),  # Random merchants
    "Category": np.random.choice(['Shopping', 'Food', 'Travel'], size=200),  # Random categories
    "Location": np.random.choice(['London', 'Paris', 'New York'], size=200),  # Random locations
    "Date": pd.date_range(start="2024-01-01", periods=200, freq='h'),  # Random date range starting from 2024-01-01
    "Is_Fraud": np.random.choice([0, 1], size=200, p=[0.85, 0.15])  # 15% fraud and 85% genuine
}
# Convert the dictionary to a pandas DataFrame
df = pd.DataFrame(data)
# Step 1: Introduce Missing Values (for the demonstration)
# Randomly introduce missing values in the 'Amount', 'Merchant', 'Category', and 'Location' columns
missing_rate = 0.3  # Introduce missing data in 10% of the entries
n_missing_amount = int(missing_rate * len(df))
n_missing_merchant = int(missing_rate * len(df))
n_missing_category = int(missing_rate * len(df))
n_missing_location = int(missing_rate * len(df))
# Introduce missing values
df.loc[df.sample(n=n_missing_amount).index, 'Amount'] = np.nan
df.loc[df.sample(n=n_missing_merchant).index, 'Merchant'] = np.nan
df.loc[df.sample(n=n_missing_category).index, 'Category'] = np.nan
df.loc[df.sample(n=n_missing_location).index, 'Location'] = np.nan
# Step 2: Data Cleaning (Handling Missing Values)
# Impute missing values for numerical and categorical columns
imputer_cat = SimpleImputer(strategy='most_frequent')  # Most frequent for categorical columns
imputer_num = SimpleImputer(strategy='median')  # Median for numerical columns

# Impute missing values in the 'Merchant', 'Category', 'Location' columns (categorical)
df['Merchant'] = imputer_cat.fit_transform(df[['Merchant']]).ravel()
```

```
df['Category'] = imputer_cat.fit_transform(df[['Category']]).ravel()
df['Location'] = imputer_cat.fit_transform(df[['Location']]).ravel()
# Impute missing values in the 'Amount' column (numerical)
df['Amount'] = imputer_num.fit_transform(df[['Amount']])
# Step 3: Data Preprocessing
# Convert 'Date' to datetime and extract hour, day of the week, and month as features
df['Date'] = pd.to_datetime(df['Date'])
df['Hour'] = df['Date'].dt.hour
df['DayOfWeek'] = df['Date'].dt.weekday
df['Month'] = df['Date'].dt.month
# Drop 'Date' and 'TransactionID' as they are not needed for the model
df.drop(columns=['Date', 'TransactionID'], inplace=True)
# One-Hot Encoding for 'Merchant', 'Category', and 'Location'
df = pd.get_dummies(df, columns=['Merchant', 'Category', 'Location'], drop_first=True)
# Feature Scaling: Standardizing 'Amount' using StandardScaler
scaler = StandardScaler()
df[['Amount']] = scaler.fit_transform(df[['Amount']])
# Step 4: Split the Data into Train and Test Sets
X = df.drop(['Is_Fraud'], axis=1)  # Features
y = df['Is_Fraud']  # Target column
# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 5: Train the Decision Tree Model
model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=2)
model.fit(X_train, y_train)
# Step 6: Evaluate the Model
# Make predictions on the test set
y_pred = model.predict(X_test)
# Print confusion matrix and classification report
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
# Step 7: Calculate Model Accuracy

accuracy = (y_pred == y_test).mean()

print(f"\nModel Accuracy: {accuracy:.4f}")

# Step 8: Visualize Decision Tree (Optional)

plt.figure(figsize=(20, 10))

from sklearn.tree import plot_tree

plot_tree(model, filled=True, feature_names=X.columns, class_names=['Genuine', 'Fraud'], rounded=True)

plt.show()
```

RESULTS:

1. **Performance Metrics**

METRIC	FRAUD CLASS(1)	GENUINE CLASS(0)
Precision	0.67	0.92
Recall	0.40	0.97
F1-Score	0.50	0.94
Accuracy	90%	

2. **Confusion Matrix**

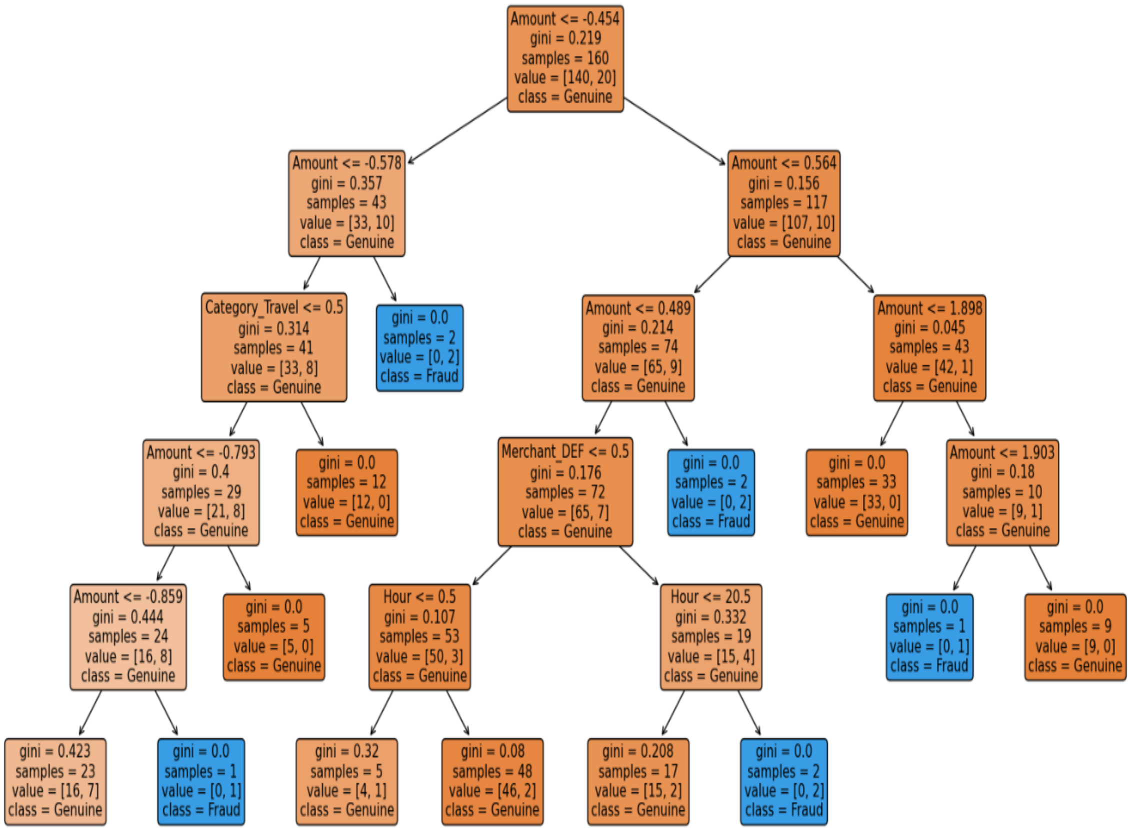
PREDICTED	GENUINE(1)	FRAUD(0)
Actual genuine	32	1
Actual fraud	7	0

INSIGHTS:

- High accuracy (90%) in detecting genuine transactions, but recall for fraudulent transactions (40%) indicates potential improvement areas.
- Precision (67%) for fraud class suggests the model effectively identifies fraud when predicted but misses some fraud cases.

Tree Visualization:

➢ A graphical depiction of the decision tree reveals clear, interpretable rules for classification, offering transparency to non-technical stakeholders.



Discussions:

Strengths:

- **Simplicity and Transparency:** Decision Tree models provide interpretable decision rules, essential for financial applications.
- **Effective Preprocessing:** The combination of imputation, feature engineering, and scaling ensured data integrity and model robustness.

Limitations:

- **Class Imbalance:** The rare occurrence of fraud reduced recall for fraudulent transactions.
- **Overfitting Risk:** Controlled by limiting tree depth, though at the cost of missing subtle patterns.

Improvement Strategies:

I. **Class Imbalance Solutions:**
- Use `class_weight='balanced'` to prioritize fraud cases during training.
- Apply **Synthetic Minority Oversampling (SMOTE)** to artificially increase fraudulent samples.

II. **Advanced Algorithms:**
- Explore ensemble methods like Random Forest or Gradient Boosting for better performance.

III. **Feature Expansion:**
- Include additional attributes, such as customer transaction history or merchant reputation.

CONCLUSION:

This project successfully demonstrates the use of Decision Trees for fraud detection, balancing interpretability and performance. While achieving 90% accuracy, addressing class imbalance through advanced methods could further enhance fraud detection capabilities. The insights gained here provide a solid foundation for deploying Decision Tree-based models in real-world financial systems.