Decision Tree Fraud Detection Program

Welcome to the Decision Tree Fraud Detection Program! This tool is designed to help detect fraudulent transactions by using a decision tree classifier. Below, you will find a comprehensive explanation of the components and functionality of the program.

Key Features

- **Dynamic Feature Selection**: Users can specify which features to include in the analysis.
- **Data Balancing**: The program uses SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance.
    - Generates nearest neighbors of fraudulent transactions to overcome limited dataset
- **Model Training and Evaluation**: Splits the dataset into training and testing sets, trains a decision tree, and evaluates its performance.
- **Decision Explanation**: Provides insights into how the decision tree made its predictions.

Columns in the Dataset

General Information

- `Unnamed: 0`: Row index, not useful as a feature.
- `accountNumber`: Unique identifier for the account, not recommended
- `customerId`: Unique customer identifier, may be useful for customer-level analysis

Financial Information

- `creditLimit`: Credit limit of the account, good for understanding account capacity
- `availableMoney`: Remaining credit, useful for analyzing spending behavior
- `currentBalance`: Current balance of the account, useful for financial state analysis

Transaction Details

- `transactionDateTime`: Timestamp of the transaction, may need feature engineering
- `transactionAmount`: Amount of the transaction, highly relevant for fraud detection

- **`transactionType`**: Type of transaction (e.g., purchase, withdrawal), relevant for context

Merchant Information

- **`merchantName`**: Name of the merchant, might be categorical and high cardinality
- **`acqCountry`**: Country of the acquiring bank, useful for geographic analysis
- **`merchantCountryCode`**: Merchant's country code, can be useful for fraud patterns
- **`merchantCity`**: City of the merchant, may need geographic aggregation
- **`merchantState`**: State of the merchant, similar use as `merchantCity`
- **`merchantZip`**: ZIP code of the merchant, less commonly used unless for location-specific insights

POS Information

- **`posEntryMode`**: Mode of transaction entry (e.g., chip, swipe), relevant for detecting anomalies
- **`posConditionCode`**: POS condition at the time of transaction, good for pattern analysis
- **`cardPresent`**: Indicates if the card was present during the transaction, highly relevant for fraud
- **`posOnPremises`**: POS on-premises indicator, useful for transaction context

Additional Features

- **`recurringAuthInd`**: Indicates if the transaction is part of a recurring authorization, relevant for patterns
- **`expirationDateKeyInMatch`**: Match status of entered vs. stored expiration date, useful for mismatches
- **`cardCVV`**: CVV code of the card, could be useful if not encrypted
- **`enteredCVV`**: CVV entered by the customer, relevant for mismatch detection
- **`cardLast4Digits`**: Last four digits of the card, not usually useful on its own

Target Variable

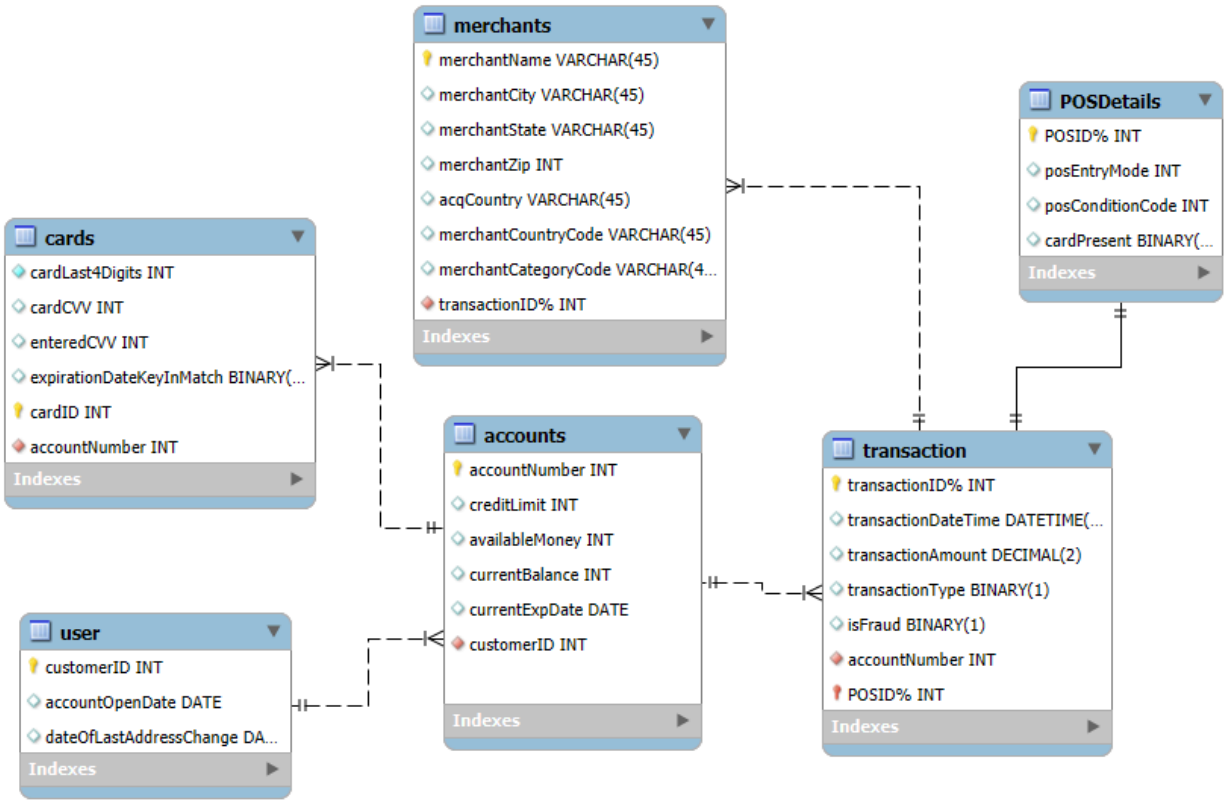
- **`isFraud`**: Indicates whether the transaction is fraudulent (1 for fraud, 0 for not fraud)

Explanation of Imports

- **numpy**: Numerical computations and array manipulations
- **pandas**: Data manipulation and analysis using DataFrames
- **scikit-learn**:
    - `SelectKBest`, `chi2`: Select top features based on statistical tests.
    - `KBinsDiscretizer`: Discretizes continuous features into bins.
    - `DecisionTreeClassifier`, `plot_tree`: Builds and visualizes decision tree models.
    - `train_test_split`: Splits the dataset into training and testing subsets.
    - `accuracy_score`, `classification_report`, `confusion_matrix`: Evaluates model performance.
    - `GridSearchCV`: systematically search for the best combination of hyperparameters for a given model
- **imbalanced-learn**:
    - `SMOTE`: Addresses class imbalance by generating synthetic samples
        - Generates nearest neighbors of fraudulent transactions to overcome limited dataset
- **matplotlib.pyplot**: Generates visualizations like confusion matrices and decision trees
- **seaborn**: Creates enhanced visualizations
- **joblib**: Saves and loads Python objects, including trained models. Used for reusing model in authentication of transactions
- **solcx**:
    - `install_solc`, `set_solc_version`, `compile_source`: Manages and compiles Solidity contracts for blockchain interaction.
        - Had to install a specific version because some existing ones are not supported with existing software
- **web3**: Interacts with the Ethereum blockchain, including deploying and managing smart contracts.
    - Documentation listed at bottom
- **Harhat**: development environment to compile, deploy, test, and debug Ethereum software. Used for authentication of transactions
    - Documentation listed at bottom

Relational model of the Dataset

This dataset was found on kaggle and represents different credit card transactions. In this dataset, I focused on predicting if transactions were fraudulent or not based on external factors. Link to data: [Fraud Detection Dataset]



How to Use

Step 1: Install Dependencies

Ensure you have the required Python libraries installed (when you run the script, it will prompt you to install libraries you don't already have):

- `numpy`
- `pandas`
- `scikit-learn`
- `imbalanced-learn`
- `matplotlib`
- `seaborn`
- `web3`

Step 2: Prepare Your Dataset

Place your dataset (`transactions.csv`) in the same directory as the script. Ensure it contains the columns listed above.

Step 3: Run the Jupyter Notebook / Python Script

1. Open the script in a Python IDE.
2. Modify the `selected_features` list to include the features you want to use.
3. Run the script to train the decision tree model and evaluate its performance.

Step 4: Interpret the Results

The script will:

- Print the model's accuracy, classification report, and confusion matrix.
- Visualize the decision tree structure.
- Display feature importances and provide an explanation of the decision-making process.
- Authenticate a transaction on the Ethernet network based on criteria of decision tree.

How to enable hardhat for ETH transactions

Run the following in command line:

- `mkdir AIBlockchainProj`
- `cd AIBlockchainProj`

- `npm init -y`
- `npm install --save-dev hardhat`
- `npx hardhat`
- `npx hardhat node`

Sample Results

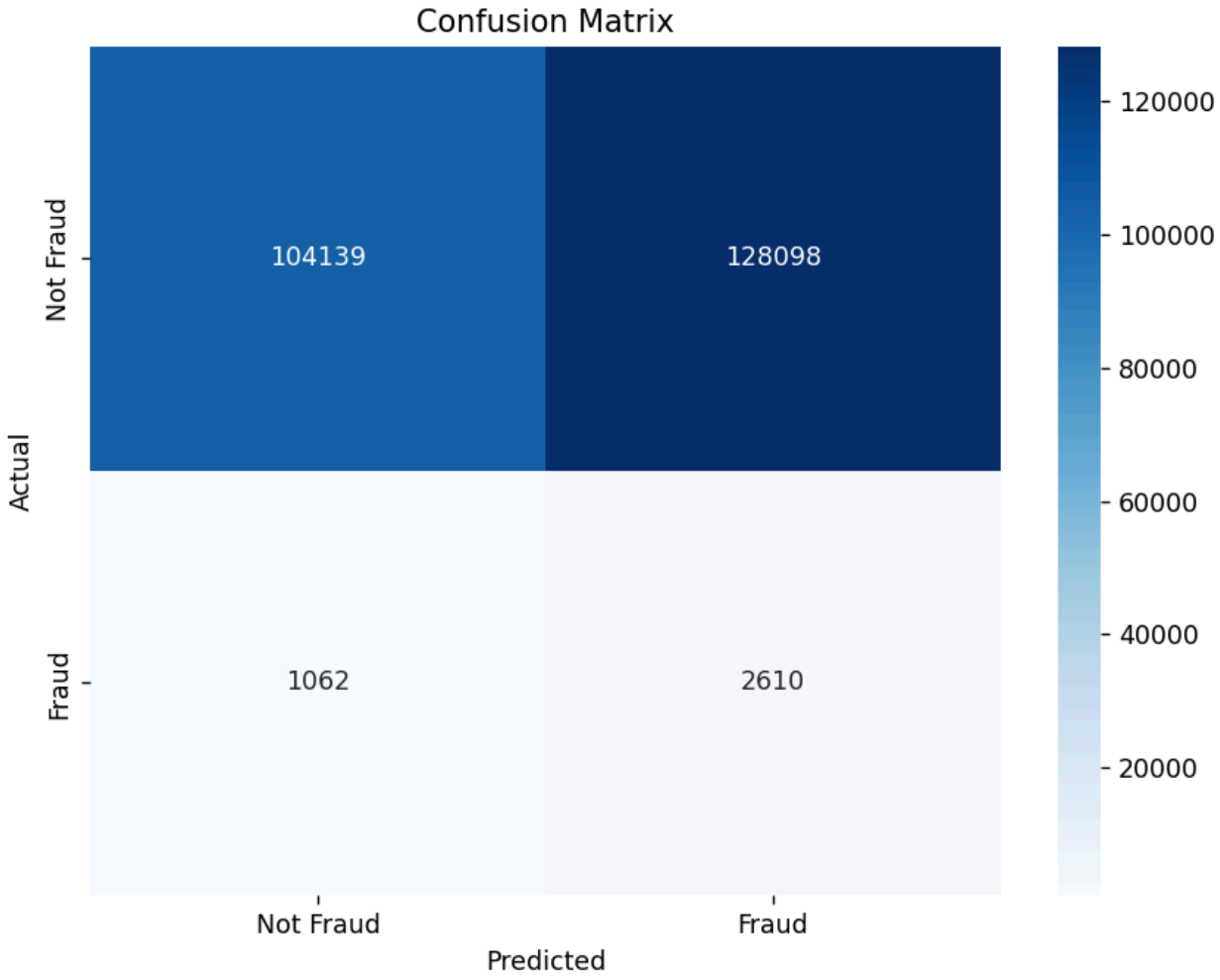
Data Preview:

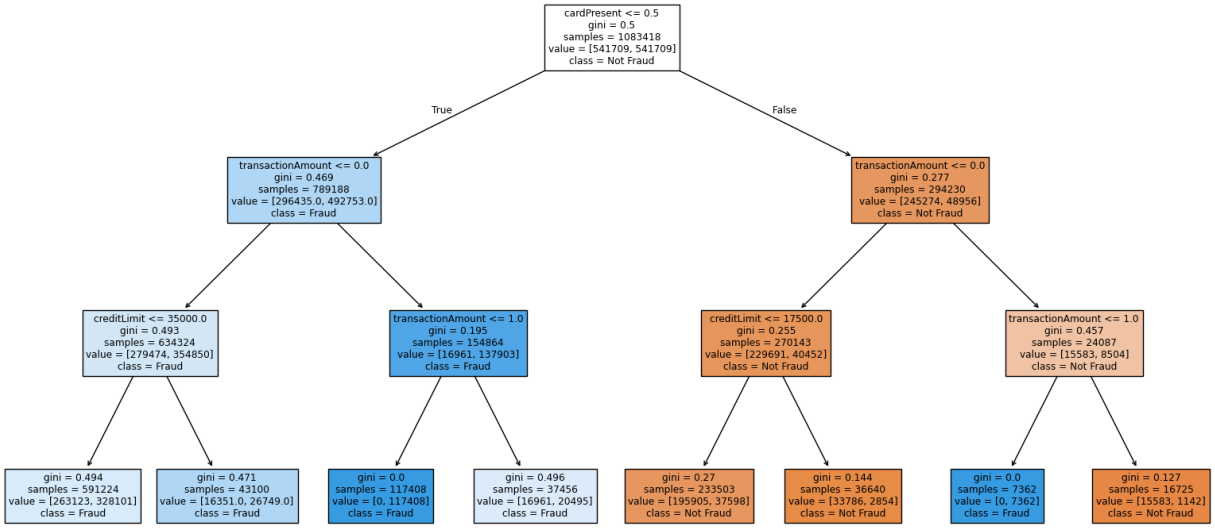
```
   Unnamed: 0  accountNumber  customerId  creditLimit  availableMoney  \
0           0     737265056   737265056         5000          5000.0
1           1     737265056   737265056         5000          5000.0
2           2     737265056   737265056         5000          5000.0
3           3     737265056   737265056         5000          5000.0
4           4     830329091   830329091         5000          5000.0

    transactionDateTime  transactionAmount         merchantName acqCountry  \
0  2016-08-13T14:27:32              98.55              Uber         US
1  2016-10-11T05:05:54              74.51         AMC #191138         US
2  2016-11-08T09:18:39               7.47         Play Store         US
3  2016-12-10T02:14:50               7.47         Play Store         US
4  2016-03-24T21:04:46              71.18  Tim Hortons #947751         US

  merchantCountryCode  ...  echoBuffer  currentBalance merchantCity  \
0                  US  ...         NaN             0.0          NaN
1                  US  ...         NaN             0.0          NaN
2                  US  ...         NaN             0.0          NaN
3                  US  ...         NaN             0.0          NaN
4                  US  ...         NaN             0.0          NaN

  merchantState merchantZip cardPresent  posOnPremises  recurringAuthInd  \
0           NaN         NaN       False            NaN               NaN
1           NaN         NaN        True            NaN               NaN
...
3                    False   False
4                    False   False
```





Contract deployed at 0x5FbDB2315678afecb367f032d93F642f64180aa3

Funds released to payee.

More Information about Packages

Hardhat:

- GitHub - NomicFoundation/hardhat: Hardhat is a development environment to compile, deploy, test, and debug your Ethereum software.
- Hardhat | Ethereum development environment for professionals by Nomic Foundation

Web3:

- gm — web3.py 7.6.0 documentation

Imports:

- SelectKBest — scikit-learn 1.6.0 documentation
- ML 101: Feature Selection With SelectKBest Using Scikit-Learn (Python) » EML
- KBinsDiscretizer — scikit-learn 1.6.0 documentation
- Building and Implementing Decision Tree Classifiers with Scikit-Learn: A Comprehensive Guide - GeeksforGeeks
- Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier | DataCamp

I also used some past class notes from my ‘Foundations Business Analytics’ course where we used seaborn, pandas, and similar imports.

Overview of the Algorithm

The algorithm used to solve the problem is a Decision Tree Classifier, which is a supervised learning algorithm used for classification tasks. The Decision Tree algorithm works by splitting the data into subsets based on the values . Each node of the tree represents a decision based on a feature, each branch represents an outcome and each leaf node represents a class label (being fraudulent or not).

Analysis of the Solution

Strengths

Decision Trees are easy to visualize and understand, making it straightforward to grasp the decision-making process and also capturing non-linear relationships between features and the target variable. They also provide key insights into the importance of different features in making predictions.

Weaknesses

Decision trees are prone to overfitting, especially when the tree has increased complexity, allowing for small changes in the data causing a completely different tree structure.

Relation to Class

The use of a Decision Tree Classifier aligns with the supervised learning techniques covered in class in which agents are trained on data to take a specific action. In this case, the agents are trained on financial data trained to determine if transactions are fraudulent or not.

Results

Overall, the model performed well, achieving a good balance between precision and recall, as indicated by the F1 score. The confusion matrix visualization provided insights into the model's performance, highlighting areas where it excelled and where it could be improved.