Problem Statement: -

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We have to build a classification model to predict whether a transaction is fraudulent or not.

Purpose:

The goal of this project is to develop a machine learning model capable of identifying fraudulent credit card transactions. This helps financial institutions prevent fraud, minimize financial losses, and protect customers.

Dataset Information-

Source:

A dataset containing credit card transactions made in September 2013 by European cardholders.

Size: 284,807 transactions, 492 fraudulent transactions (0.172% of total).

Features-

Time: Time elapsed since the first transaction.

V1 to V28: Principal components.

Amount: Transaction amount.

Class-

Target variable (0 for non-fraud, 1 for fraud)

Methodology

Understand the Dataset

- The dataset consists of 284,807 transactions with 492 fraudulent ones.
- Features: Most likely anonymized features due to privacy concerns (e.g., PCA-transformed variables, amounts, timestamps).
- Target: Binary classification (0 for legitimate transactions, 1 for fraud).

Exploratory Data Analysis

Load and Explore the Data

Steps:

1. Load the dataset (usually a .csv file).
2. Check for null or missing values.
3. Analyze the class distribution to confirm the imbalance.
4. Visualize patterns (e.g., transaction amounts, time of transactions).

Handle Class Imbalance

Since the dataset is highly imbalanced:

1. **Resampling Techniques:**
    - **Oversampling:** Use techniques like SMOTE (Synthetic Minority Oversampling Technique) to increase the minority class.
    - **Undersampling:** Randomly reduce the size of the majority class.
2. **Cost-sensitive Learning:**
    - Modify the loss function to penalize misclassifying the minority class more heavily.
3. **Ensemble Methods:**
    - Models like Random Forest or XGBoost often handle imbalanced datasets well.

Data Cleaning

Check for Missing Values

If there are missing values in the dataset, they need to be handled. For example:

- Impute missing values with the mean/median (for numerical features) or mode (for categorical features).
- Drop rows/columns with excessive missing values if necessary.

Standardization/Normalization

Since many ML algorithms are sensitive to feature scaling, we standardize the numerical features.

Outlier Detection and Handling

- Detect outliers using statistical methods (e.g., z-score, IQR).
- Decide whether to remove or transform the outliers.

Dealing with Imbalanced data

Techniques for Balancing Imbalanced Data

Oversampling the Minority Class

- **Synthetic Minority Oversampling Technique (SMOTE):** Generates synthetic samples for the minority class by interpolating between existing instances.
- **Random Oversampling:** Simply duplicates existing instances of the minority class.

Undersampling the Majority Class

- Reduces the number of majority class samples to balance the dataset. This can result in loss of valuable data if not done carefully.

Combining Oversampling and Undersampling

- A hybrid approach that balances the classes without significantly increasing the dataset size or losing critical information.

Cost-sensitive Learning

- Modifies the model to assign a higher penalty to misclassifying the minority class (fraudulent transactions).

Feature Engineering:

Transform Existing Features

- **Log Transformation:** Apply a log transformation to skewed numerical features like Amount to reduce the effect of outliers.
- **Time Features:**
    - The Time feature may represent seconds elapsed since the first transaction. Transform it into more meaningful features like hour_of_day or day_part (e.g., morning, afternoon).
- **Interaction Terms:** Create interaction features between existing variables to capture hidden patterns.

Create New Features

- **Transaction Patterns:**
    - Create features like the **rolling mean** or **standard deviation** of transaction amounts over time.
    - Create flags for unusually large transactions based on historical data.
- **Transaction Frequency:**
    - Count the number of transactions in the last hour/day/week to detect abnormal activity.

Encode Categorical Features

If any categorical features exist, use:

- **One-Hot Encoding** for low-cardinality features.
- **Frequency Encoding** for high-cardinality features.

Normalize/Standardize Features

Ensure that all numerical features are scaled to avoid dominance of one feature due to its range.

Feature Engineering

Model Options

a. Baseline Models

- **Logistic Regression:** A simple, interpretable model that works well as a baseline for binary classification.
- **Decision Tree:** Handles class imbalance reasonably well and is easy to interpret.

b. Advanced Models

- **Random Forest:** A robust ensemble method that works well on imbalanced datasets by handling overfitting and feature importance.
- **Gradient Boosting (e.g., XGBoost, LightGBM, CatBoost):**
    - Popular choices for tabular data.
    - Can handle imbalanced data using hyperparameters like scale_pos_weight.
- **Support Vector Machine (SVM):** Effective for small datasets and imbalanced data but can be computationally expensive for larger datasets.

c. Deep Learning Models

- **Neural Networks:** Useful for large datasets but may require more tuning and computational resources.
- **Autoencoders:** Often used for anomaly detection by learning a compressed representation of normal transactions.

Model Training

To train a model and estimate the best parameters, follow these steps:

1. Import Required Libraries.
2. Load and Explore Data.
3. Preprocessing.
4. Split Data into Train and Test Sets.
5. Train the Model.
6. Evaluate the Model.
7. Hyperparameter Tuning.

Model Validation Process

Model validation helps assess how well a trained model generalizes to unseen data. Here's a step-by-step approach:

1. Import Libraries.
2. Split Data into Training and Testing Sets.
3. Train the Model.
4. Evaluate on Test Set.
5. Cross-Validation.
6. Check for Overfitting.
7. Residual Analysis.

Dependencies:

Install the required Python libraries:

pip install pandas, numpy, scikit-learn, imbalanced-learn, matplotlib, seaborn, xgboost, sagemaker