Enhancing Credit Card Fraud Detection

Using Naive Bayes algorithm in Machine Learning

Includes: full Python script and README

README

Enhancing Credit Card Fraud Detection using Naive Bayes
Contents:
 - fraud_detection_naive_bayes.py : A Python script with full pipeline
 - Instructions to run (see bottom of script)
Summary:
This repository contains a baseline implementation using Gaussian Naive Bayes. The Kaggle 'creditcard.csv'
dataset (originally from European card transactions) is expected. Place it in the repository root.
Improvements to consider:
 - Use SMOTE (imblearn) for smarter oversampling
 - Try tree-based models (RandomForest, XGBoost) and compare results
 - Tune thresholds for precision-recall trade-offs
 - Use cross-validation and grid search for robust estimates

fraud_detection_naive_bayes.py

```
"""
Enhancing Credit Card Fraud Detection using Naive Bayes algorithm in Machine Learning
File: fraud_detection_naive_bayes.py
Brief:
This script demonstrates a complete pipeline for detecting credit card fraud using a Naive Bayes
classifier.
It includes:
 - Loading the dataset (expects 'creditcard.csv' in the working directory)
 - Basic exploratory checks
 - Preprocessing (scaling of 'Amount', dropping 'Time' optionally)
 - Handling class imbalance by simple upsampling of the minority class
 - Training a Gaussian Naive Bayes model
 - Evaluating performance with confusion matrix, classification report and ROC AUC
 - Saving the trained model to disk
Usage:
 - Place 'creditcard.csv' (Kaggle's Credit Card Fraud Detection dataset) in the same folder or
adjust the path.
 - Install required packages: scikit-learn, pandas, numpy, matplotlib (for plotting if desired)
 - Run: python fraud_detection_naive_bayes.py
Note:
 - This is intended as an educational baseline. For production, consider:
   * More advanced preprocessing
   * Feature selection / dimensionality reduction
   * More robust imbalance handling (SMOTE, class-weighted models)
   * Cross-validation and hyperparameter tuning
"""
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (confusion_matrix, classification_report,
                             roc_auc_score, roc_curve)
import joblib
import matplotlib.pyplot as plt
def load_data(path='creditcard.csv'):
    if not os.path.exists(path):
        raise FileNotFoundError(f"Dataset not found at {path}. Please place 'creditcard.csv' in the
working directory.")
    df = pd.read_csv(path)
    return df
def preprocess(df, drop_time=True, scale_amount=True):
    df = df.copy()
    if drop_time and 'Time' in df.columns:
        df = df.drop(columns=['Time'])
    # Scale Amount
    if scale_amount and 'Amount' in df.columns:
        scaler = StandardScaler()
        df['Amount_scaled'] = scaler.fit_transform(df[['Amount']])
        df = df.drop(columns=['Amount'])
    else:
        scaler = None
    # Features and labels
    X = df.drop(columns=['Class'])
    y = df['Class']
    return X, y, scaler
def upsample_minority(X, y, random_state=42):
    # Simple upsampling using pandas (duplicate minority samples)
    df = X.copy()
    df['Class'] = y.values
    counts = df['Class'].value_counts()
    if counts.min() == counts.max():
        return X, y
    majority = df[df['Class'] == 0]
    minority = df[df['Class'] == 1]
    minority_upsampled = minority.sample(n=len(majority), replace=True, random_state=random_state)
    df_upsampled = pd.concat([majority, minority_upsampled]).sample(frac=1,
random_state=random_state).reset_index(drop=True)
```

```
    y_res = df_upsampled['Class']
    X_res = df_upsampled.drop(columns=['Class'])
    return X_res, y_res
def train_and_evaluate(X_train, X_test, y_train, y_test):
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = None
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:,1]
    cm = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)
    auc = None
    if y_proba is not None:
        try:
            auc = roc_auc_score(y_test, y_proba)
        except Exception:
            auc = None
    return model, cm, report, auc, y_proba
def plot_roc(y_test, y_proba, outpath='roc_curve.png'):
    if y_proba is None:
        print("Model does not provide probabilities; cannot plot ROC.")
        return
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    plt.figure()
    plt.plot(fpr, tpr)
    plt.plot([0,1],[0,1], linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.savefig(outpath)
    plt.close()
def main(dataset_path='creditcard.csv'):
    print("Loading data...")
    df = load_data(dataset_path)
    print("Dataset shape:", df.shape)
    print("Class distribution:\n", df['Class'].value_counts())
    X, y, scaler = preprocess(df)
    print("After preprocessing, features:", X.shape)
    # Handle imbalance by upsampling minority class
    print("Upsampling minority class to balance dataset...")
    X_res, y_res = upsample_minority(X, y)
    print("Resampled class distribution:\n", y_res.value_counts())
    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2,
random_state=42, stratify=y_res)
    print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
    model, cm, report, auc, y_proba = train_and_evaluate(X_train, X_test, y_train, y_test)
    print("Confusion Matrix:\n", cm)
    print("Classification Report:\n", report)
    if auc is not None:
        print(f"ROC AUC: {auc:.4f}")
        plot_roc(y_test, y_proba, outpath='roc_curve.png')
        print("ROC curve saved to 'roc_curve.png'")
    # Save model
    model_path = 'naive_bayes_fraud_model.joblib'
    joblib.dump({'model': model, 'scaler': scaler}, model_path)
    print(f"Model and scaler saved to {model_path}")
if __name__ == '__main__':
    try:
        main('creditcard.csv')
    except Exception as e:
        print("Error:", e)
```