1.Credit Card Fraud Detection Using Machine Learning

Introduction

With the rapid growth of online and digital payments, credit card fraud has become a serious concern in the financial industry. Fraudulent transactions cause significant financial losses and reduce customer trust. Since fraud cases are very rare compared to normal transactions, traditional rule-based systems often fail to detect them accurately. Therefore, there is a strong need for an intelligent and automated fraud detection system that can work in real time and adapt to changing fraud patterns.

Dataset Description

This project uses the Credit Card Fraud Detection Dataset obtained from Kaggle. The dataset contains real and anonymized credit card transactions made by European cardholders. It includes 284,807 transactions, out of which only 492 are fraudulent, making it a highly imbalanced dataset. The features consist of Time, Amount, and 28 anonymized variables (V1–V28) obtained using PCA for privacy protection. The target variable Class indicates whether a transaction is fraudulent (1) or legitimate (0).

Proposed Methodology

Initially, the dataset is preprocessed by handling missing values and scaling the numerical features. To address the issue of class imbalance, the SMOTE (Synthetic Minority Over-sampling Technique) is applied to balance fraudulent and non-fraudulent samples. An XGBoost classifier is then trained to learn complex transaction patterns. The model performance is evaluated using ROC-AUC score, confusion matrix, and classification report. In addition, a threshold-based logic is implemented to simulate real-time fraud detection, generating alerts for suspicious transactions.

Conclusion

The proposed machine learning-based fraud detection system effectively identifies fraudulent credit card transactions despite severe class imbalance. By combining data preprocessing, imbalance handling, and a powerful classification model, the system achieves reliable detection performance. The inclusion of real-time fraud simulation makes the approach practical for real-world financial applications.

```
!pip install imbalanced-learn xgboost
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.12/dist-packages (0.14.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.1.2)
Requirement already satisfied: numpy<3,>=1.25.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (2.0.2)
Requirement already satisfied: scipy<2,>=1.11.4 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (1.16.3)
Requirement already satisfied: scikit-learn<2,>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (1.6.1)
Requirement already satisfied: joblib<2,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (1.5.3)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (3.6.0
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.28.9)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve

from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
```

```
df = pd.read_csv('/content/creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.11
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.10
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.90
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.19
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.13

5 rows × 31 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49610 entries, 0 to 49609
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    49610 non-null  int64  
 1   V1      49610 non-null  float64
 2   V2      49610 non-null  float64
 3   V3      49610 non-null  float64
 4   V4      49609 non-null  float64
 5   V5      49609 non-null  float64
 6   V6      49609 non-null  float64
 7   V7      49609 non-null  float64
 8   V8      49609 non-null  float64
 9   V9      49609 non-null  float64
 10  V10     49609 non-null  float64
 11  V11     49609 non-null  float64
 12  V12     49609 non-null  float64
 13  V13     49609 non-null  float64
 14  V14     49609 non-null  float64
 15  V15     49609 non-null  float64
 16  V16     49609 non-null  float64
 17  V17     49609 non-null  float64
 18  V18     49609 non-null  float64
 19  V19     49609 non-null  float64
 20  V20     49609 non-null  float64
 21  V21     49609 non-null  float64
 22  V22     49609 non-null  float64
 23  V23     49609 non-null  float64
 24  V24     49609 non-null  float64
 25  V25     49609 non-null  float64
 26  V26     49609 non-null  float64
 27  V27     49609 non-null  float64
 28  V28     49609 non-null  float64
 29  Amount  49609 non-null  float64
 30  Class   49609 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 11.7 MB
```

```
df['Class'].value_counts()
```

	count
Class	
0.0	49461
1.0	148

dtype: int64

```
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
df['Time'] = scaler.fit_transform(df[['Time']])
```

```
df.isna().sum()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	1
V5	1
V6	1
V7	1
V8	1
V9	1
V10	1
V11	1
V12	1
V13	1
V14	1
V15	1
V16	1
V17	1
V18	1
V19	1
V20	1
V21	1
V22	1
V23	1
V24	1
V25	1
V26	1
V27	1
V28	1
Amount	1
Class	1

dtype: int64

```
df['Class'].unique()
```

```
array([ 0.,  1., nan])
```

```
df = df.dropna(subset=['Class'])
```

```
df['Class'] = df['Class'].fillna(0)
```

```
df.isna().sum()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
Amount	0
Class	0

dtype: int64

```
df['Class'].unique()
```

```
array([0., 1.])
```

```
from sklearn.model_selection import train_test_split

X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)
```

```
X.isna().sum().sort_values(ascending=False).head()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	0

dtype: int64

```
X = X.fillna(0)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

print("After SMOTE:", y_train_res.value_counts())
```

```
After SMOTE: Class
0.0    39569
1.0    39569
Name: count, dtype: int64
```

```
from xgboost import XGBClassifier

model = XGBClassifier(
    n_estimators=150,
    max_depth=4,
    learning_rate=0.1,
    eval_metric='logloss',
    random_state=42
)

model.fit(X_train_res, y_train_res)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=150, n_jobs=None,
```

```
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:,1]
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      9892
         1.0       0.84      0.90      0.87        30

    accuracy                           1.00      9922
   macro avg       0.92      0.95      0.94      9922
weighted avg       1.00      1.00      1.00      9922
```

```
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
```

```
ROC-AUC: 0.9865177247607495
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[9887,    5],
       [   3,   27]])
```

```
def detect_fraud(transaction):
    prob = model.predict_proba(transaction)[:,1]
    if prob > 0.35:
        return "🚨 FRAUD ALERT", prob
    else:
        return "✅ NORMAL", prob
```

```
sample = X_test.sample(1)
status, probability = detect_fraud(sample)

print(status)
print("Fraud Probability:", probability)
```

```
✅ NORMAL
Fraud Probability: [3.4016693e-06]
```

2.Customer Churn Prediction for Telecom Using Machine Learning

Introduction

In the telecommunications industry, customer churn refers to customers discontinuing their services and switching to competitors. Retaining existing customers is more cost-effective than acquiring new ones, making churn prediction a critical business problem. Due to the large volume of customer data and complex usage patterns, manual analysis is inefficient. Hence, machine learning techniques are used to automatically predict customers who are likely to churn, enabling telecom companies to take preventive actions.

Dataset Description

The Telco Customer Churn Dataset from Kaggle is used in this project. The dataset contains customer demographic information, service usage details, and billing data. Key features include gender, tenure, contract type, monthly charges, total charges, and payment method. The target variable Churn indicates whether a customer has left the service (Yes) or continues (No). The dataset includes both numerical and categorical features, making it suitable for classification and feature importance analysis.

Proposed Methodology

The dataset is first cleaned by handling missing values and converting categorical variables using one-hot encoding. Exploratory Data Analysis (EDA) is performed to identify key churn-influencing factors such as contract type, tenure, and monthly charges. A Gradient Boosting / Decision Tree classifier is trained to predict customer churn. Model performance is optimized using cross-validation and hyperparameter tuning. Feature importance analysis is carried out to understand which factors contribute most to churn prediction.

Innovation / Key Contribution

Instead of treating churn prediction as a black-box model, this project emphasizes interpretability through feature importance visualization. This allows telecom providers to clearly identify high-risk customers and the reasons behind churn, enabling targeted retention strategies such as personalized offers and contract modifications.

Conclusion

The proposed machine learning-based churn prediction system effectively identifies customers likely to leave a telecom service. By combining data preprocessing, model optimization, and feature importance analysis, the system provides both accurate predictions and actionable business insights. This approach supports proactive decision-making and improves customer retention in the telecom industry.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
```

```
df = pd.read_csv('/content/WA_Fn-UseC_-Telco-Customer-Churn.csv')
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No

5 rows × 21 columns

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df = df.dropna()
```

```
df['Churn'] = df['Churn'].map({'Yes':1, 'No':0})
```

```
df_encoded = pd.get_dummies(df, drop_first=True)
```

```
X = df_encoded.drop('Churn', axis=1)
y = df_encoded['Churn']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

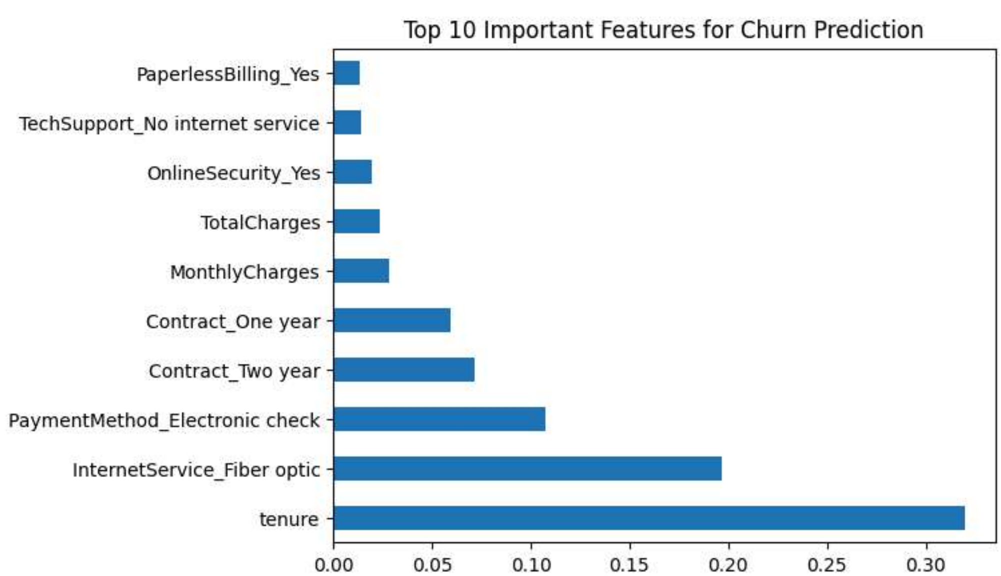
```
model = GradientBoostingClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

```
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7945984363894811
[[932 101]
 [188 186]]
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1033
           1       0.65      0.50      0.56       374

    accuracy                           0.79      1407
   macro avg       0.74      0.70      0.71      1407
weighted avg       0.78      0.79      0.79      1407
```

```
importances = model.feature_importances_
features = X.columns

feat_imp = pd.Series(importances, index=features)
top_features = feat_imp.sort_values(ascending=False).head(10)

top_features.plot(kind='barh')
plt.title("Top 10 Important Features for Churn Prediction")
plt.show()
```



3.Sales Forecasting for Retail Using Time-Series Analysis

Introduction

Accurate sales forecasting is essential for retail businesses to manage inventory, reduce losses, and improve customer satisfaction. Poor forecasting can lead to overstocking or stock shortages, directly impacting revenue. With the availability of historical sales data, machine learning and time-series models can be used to predict future sales trends effectively. This project focuses on forecasting retail sales using historical Walmart sales data.

Dataset Description

The project uses the Walmart Sales Forecasting Dataset obtained from Kaggle. The dataset contains historical weekly sales records of multiple Walmart stores along with related economic factors such as fuel price, consumer price index, and unemployment rate. The time-series nature of the dataset makes it suitable for identifying trends and seasonal patterns in retail sales.

Proposed Methodology

Initially, the dataset is cleaned and aggregated to obtain total weekly sales across stores. Time-series preprocessing is performed by converting dates into a standard format and sorting them chronologically. The Prophet forecasting model is used to capture trend and seasonality in sales data. Future sales are predicted for upcoming weeks, and model performance is evaluated using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

Conclusion

The proposed sales forecasting system successfully predicts future retail sales by learning historical trends and seasonal patterns. Such forecasts help retailers optimize inventory planning and improve operational efficiency. The model provides a practical and data-driven solution for decision-making in the retail industry.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prophet import Prophet
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# Load dataset
df = pd.read_csv('/content/train.csv')

# Convert Date column
df['Date'] = pd.to_datetime(df['Date'])

# Check data
df.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

```
# Aggregate weekly sales by date
sales_ts = df.groupby('Date')['Weekly_Sales'].sum().reset_index()

# Rename columns for Prophet
sales_ts.columns = ['ds', 'y']

sales_ts.head()
```

	ds	y
0	2010-02-05	49750740.50
1	2010-02-12	48336677.63
2	2010-02-19	48276993.78
3	2010-02-26	43968571.13
4	2010-03-05	46871470.30

```
# Last 12 weeks for testing
train = sales_ts.iloc[:-12]
test = sales_ts.iloc[-12:]
```

```
model = Prophet(
    yearly_seasonality=True,
    weekly_seasonality=False,
    daily_seasonality=False
)

model.fit(train)
```

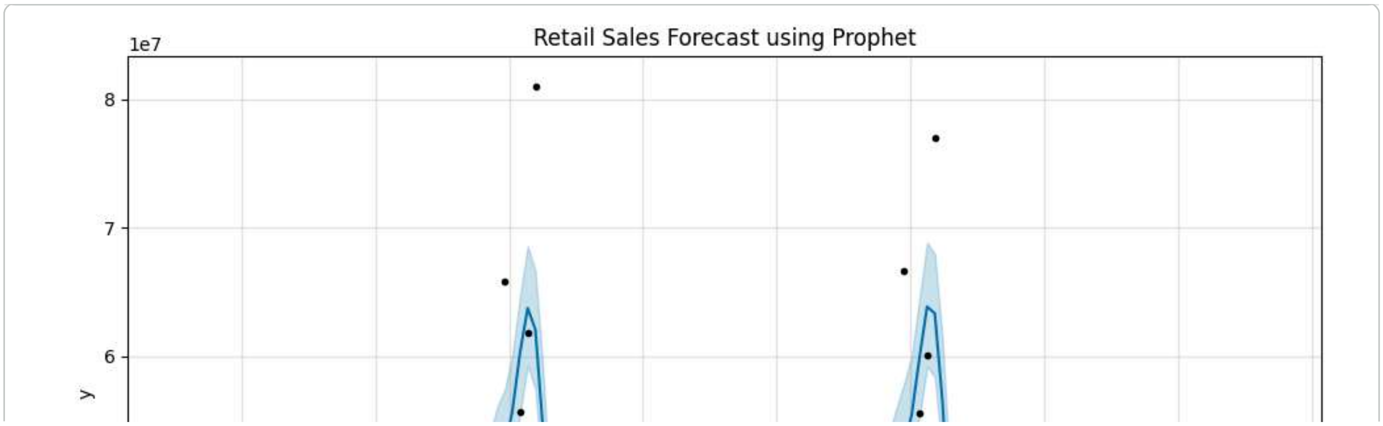
```
<prophet.forecaster.Prophet at 0x7e17add18440>
```

```
future = model.make_future_dataframe(periods=12, freq='W')
forecast = model.predict(future)
```

```
predicted = forecast.iloc[-12:]['yhat']

mae = mean_absolute_error(test['y'], predicted)
rmse = np.sqrt(mean_squared_error(test['y'], predicted))

print("MAE:", mae)
print("RMSE:", rmse)
```

```
MAE: 947863.5174702244
RMSE: 1217281.2001961966
```

```
# Forecast plot
model.plot(forecast)
plt.title("Retail Sales Forecast using Prophet")
plt.show()

# Trend & Seasonality
model.plot_components(forecast)
plt.show()
```


Retail Sales Forecast using Prophet
1e7
8
7
6
y