



Credit Card Fraud Detection Using Adaptive Machine Learning and Behavioral Analytics

Prepared By

Name: Ahmad Mostafa Abutalib

ID No.: 243002097

Submitted in fulfillment of the requirements for the degree Master of
Science in Artificial Intelligence

At

Collage of Informatics

Midocean University

UAE

Supervisor Name: Dr.Amjad Qatish

Collage of Informatics, Midocean University, Comoros

Full Pipeline Overview — End-to-End Workflow

1. Data Loading

Load the Kaggle Credit Card Fraud dataset from Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/creditcard.csv')
```

2. Data Inspection

Check structure, types, and missing values.

```
df.info()
df.isnull().sum()
```

Description (1 line):

Dataset contains 31 features with *no missing values*, suitable for immediate processing.

3. Removing Missing or Invalid Entries

(If any existed — but this dataset has none.)

```
df = df.dropna()
```

Description:

Ensures data quality and prevents training issues caused by incomplete rows.

4. Feature Normalization

Normalize Time and Amount because they operate on different scales.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df["norm_time"] = scaler.fit_transform(df[["Time"]])
df["norm_amount"] = scaler.fit_transform(df[["Amount"]])
df = df.drop(["Time", "Amount"], axis=1)
```

Description:

Creates consistent feature scaling to improve ML/DL model stability.

5. Splitting into Train / Validation / Test

70% train — 15% validation — 15% test (stratified).

```
from sklearn.model_selection import train_test_split

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

# 70% train, 30% temp
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)

# 15% val, 15% test
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, random_state=42, stratify=y_temp
)
```

Description:

Prevents data leakage and creates fair evaluation sets.

6. Handling Imbalance Using SMOTE

Apply oversampling *only* on training data.

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)
```

Description:

Balances fraud vs. non-fraud transactions to prevent bias.

7. Machine Learning Models (ML Section)

7.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_resampled, y_train_resampled)
```

7.2 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42
)
rf_model.fit(X_train_resampled, y_train_resampled)
```

7.3 XGBoost

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.9,
    colsample_bytree=0.9,
    eval_metric='logloss'
)
xgb_model.fit(X_train_resampled, y_train_resampled)
```

Description for ML block:

These models learn decision boundaries based on tabular data; ideal for structured fraud detection.

8. Deep Learning Models (DL Section)

8.1 Deep Neural Network (DNN)

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Input

dnn_model = Sequential([
    Input(shape=(X_train_resampled.shape[1],)),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

dnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
dnn_model.fit(
    X_train_resampled, y_train_resampled,
    validation_data=(X_val, y_val),
    epochs=10, batch_size=256
)
```

8.2 LSTM (Sequential Transactions Learning)

Reshape to 3D

```
import numpy as np

X_train_lstm = np.expand_dims(X_train_resampled.values, axis=1)
X_val_lstm   = np.expand_dims(X_val.values, axis=1)
X_test_lstm  = np.expand_dims(X_test.values, axis=1)

Model
from keras.layers import LSTM

lstm_model = Sequential([
    Input(shape=(1, X_train_resampled.shape[1])),
    LSTM(32, return_sequences=False),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

lstm_model.fit(
    X_train_lstm, y_train_resampled,
    validation_data=(X_val_lstm, y_val),
    epochs=10, batch_size=256
)
```

Description DL block:

DL models capture deeper patterns, with LSTM learning temporal transaction behavior.

9. Unified Evaluation Function

(Used for ML + DL)

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

def evaluate(model, X_test, y_test, name, proba=True):
```

```

start = time.time()

if proba:
    y_prob = model.predict(X_test)
    y_prob = y_prob.flatten() if hasattr(y_prob, "flatten") else y_prob
else:
    y_prob = model.predict(X_test).flatten()

y_pred = (y_prob > 0.5).astype(int)
latency = time.time() - start

return {
    "model": name,
    "accuracy": accuracy_score(y_test, y_pred),
    "auc": roc_auc_score(y_test, y_prob),
    "precision": classification_report(
        y_test, y_pred, output_dict=True
    )["1"]["precision"],
    "recall": classification_report(
        y_test, y_pred, output_dict=True
    )["1"]["recall"],
    "f1": classification_report(
        y_test, y_pred, output_dict=True
    )["1"]["f1-score"],
    "confusion": confusion_matrix(y_test, y_pred),
    "latency": latency
}

```

10. Model Comparison Table

```

import pandas as pd

comparison_table = pd.DataFrame([
    # Fill with your final values
    {"Model": "XGBoost", "Accuracy": 0.9985, "AUC": 0.9679, "Precision":
0.55, "Recall": 0.82, "F1": 0.66, "FP": 50, "FN": 13},
    {"Model": "DNN", "Accuracy": 0.9987, "AUC": 0.9628, "Precision":
0.59, "Recall": 0.78, "F1": 0.67, "FP": 40, "FN": 16},
    {"Model": "LSTM", "Accuracy": 0.9991, "AUC": 0.9665, "Precision":
0.71, "Recall": 0.82, "F1": 0.76, "FP": 25, "FN": 13},
])

```

comparison_table

11. Visualization (Latency, AUC, F1)

```
import matplotlib.pyplot as plt

def plot_latency(results):
    names = [r["model"] for r in results]
    latency = [r["latency"] for r in results]

    plt.figure(figsize=(8,4))
    plt.bar(names, latency)
    plt.title("Model Latency Comparison")
    plt.ylabel("Seconds")
    plt.show()

def plot_auc_f1(results):
    names = [r["model"] for r in results]
    aucs = [r["auc"] for r in results]
    f1s = [r["f1"] for r in results]

    plt.figure(figsize=(12,4))

    plt.subplot(1,2,1)
    plt.bar(names, aucs, color="steelblue")
    plt.title("ROC-AUC Comparison")

    plt.subplot(1,2,2)
    plt.bar(names, f1s, color="orange")
    plt.title("F1 Score Comparison")

    plt.show()
```

12. Adaptive Learning (Incremental Updates)

```
from sklearn.linear_model import SGDClassifier

adaptive_model = SGDClassifier(loss="log_loss")

# initial training
adaptive_model.partial_fit(X_train, y_train, classes=[0,1])

# incremental updates
for i in range(0, len(X_val), 5000):
    adaptive_model.partial_fit(X_val[i:i+5000], y_val[i:i+5000])
```

Description:

Allows model updates without full retraining to adapt to evolving fraud patterns.

