# **Machine Learning Pipeline**

Binary Classification Project Workflow: Bank Fraud Detection

**1. Data Loading and Initial Exploration**

I began by importing the dataset using **pandas**:

df = pd.read\_csv('fraud\_detection\_bank\_dataset.csv')

To understand the structure of the dataset, I checked its shape, previewed the first few rows, and explored the distribution of the target column targets. This gave me a quick idea about the class balance:

df.shape

df.head()

df['targets'].value\_counts()

I then performed basic **Data Analysis** to check for data types, summary statistics, and missing values:

df.info()

df.describe()

df.isnull().sum()

Although some rows had missing values, I dropped them using:

df.dropna(axis=0)

**2. Feature Scaling**

Next, I used **StandardScaler** to scale the features. This is especially useful for models like SVM and Logistic Regression that are sensitive to feature magnitudes:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x\_scaled = scaler.fit\_transform(df.drop('targets', axis=1))

**3. Train-Test Split**

I separated the dataset into features X and target y, and then used an 80-20 split for training and testing:

from sklearn.model\_selection import train\_test\_split

X = df.drop('targets', axis=1)

y = df['targets']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_scaled, y, test\_size=0.2, random\_state=42)

**4. Model Training**

I trained four different models for comparison:

* Logistic Regression (with class balancing)
* Decision Tree
* Random Forest
* Support Vector Machine (SVM)

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

models = {

'Logistic Regression': LogisticRegression(class\_weight='balanced'),

'Decision Tree': DecisionTreeClassifier(),

'Random Forest': RandomForestClassifier(n\_estimators=100),

'SVM': SVC(probability=True),

}

for name, model in models.items():

model.fit(X\_train, y\_train)

**5. Model Evaluation**

For evaluation, I used key performance metrics: **Accuracy, Precision, Recall, F1 Score**, and **ROC-AUC Score**. I also printed the **classification report** and **confusion matrix** for each model.

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score

def evaluate(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba))

for name, model in models.items():

print("\n=== Evaluating:", name)

evaluate(model, X\_test, y\_test)

Then, I wrote a function to compute and store the metrics for easy comparison:

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

def get\_metrics(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

return {

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred),

'Recall': recall\_score(y\_test, y\_pred),

'F1 Score': f1\_score(y\_test, y\_pred),

'ROC AUC': roc\_auc\_score(y\_test, y\_proba)

}

results = {}

for name, model in models.items():

results[name] = get\_metrics(model, X\_test, y\_test)

**6. Model Comparison and Visualization**

To better compare the results, I stored them in a DataFrame and sorted them by **F1 Score**:

import pandas as pd

results\_df = pd.DataFrame(results).T

print(results\_df.sort\_values(by='F1 Score', ascending=False))

I also visualized the F1 Scores using a bar chart to clearly see which model performed best:

import matplotlib.pyplot as plt

results\_df['F1 Score'].plot(kind='bar', color='skyblue', title='F1 Score Comparison')

plt.ylabel('F1 Score')

plt.grid(True)

plt.show()

Bottom of Form