**CREDIT CARD FRAUD DETECTION**

**Phase4:Development part2**

In the previous phase of development, we loaded and prepared the dataset, and split it into training and test sets. In this phase, we will choose a machine learning algorithm, train the model, evaluate the model, and deploy the model.

**Feature Engineering:**

Feature engineering is the process of transforming raw data into features that are more informative and predictive for machine learning models. In the context of credit card fraud detection, feature engineering can be used to create features that capture the patterns and relationships in the data that are associated with fraud.

**Python Program:**

import pandas as pd

def create\_features(df):

"""Creates new features for credit card fraud detection.

Args:

df: A Pandas DataFrame containing the credit card transaction data.

Returns:

A Pandas DataFrame containing the new features.

"""

# Calculate the average transaction amount for each customer.

df["avg\_transaction\_amount"] = df["amount"].groupby(df["customer\_id"]).mean()

# Calculate the number of transactions made in the past 30 days for each customer.

df["num\_transactions\_past\_30\_days"] = df[df["transaction\_date"] >= pd.Timestamp.today() - pd.Timedelta(days=30)].groupby(df["customer\_id"]).size()

# Calculate the number of transactions made with the same merchant in the past 30 days for each customer.

df["num\_transactions\_with\_merchant\_past\_30\_days"] = df.groupby(["customer\_id", "merchant\_id"])[df["transaction\_date"] >= pd.Timestamp.today() - pd.Timedelta(days=30)].size()

# Calculate the distance between the customer's current location and the location of the merchant where the transaction is taking place.

df["distance\_to\_merchant"] = df.apply(lambda x: calculate\_distance(x["customer\_latitude"], x["customer\_longitude"], x["merchant\_latitude"], x["merchant\_longitude"]), axis=1)

# Return the new features.

return df

def calculate\_distance(customer\_latitude, customer\_longitude, merchant\_latitude, merchant\_longitude):

"""Calculates the distance between two points in kilometers.

Args:

customer\_latitude: The latitude of the customer's current location.

customer\_longitude: The longitude of the customer's current location.

merchant\_latitude: The latitude of the merchant where the transaction is taking place.

merchant\_longitude: The longitude of the merchant where the transaction is taking place.

Returns:

The distance between the two points in kilometers.

"""

# Calculate the distance in kilometers.

distance = geopy.distance.distance((customer\_latitude, customer\_longitude), (merchant\_latitude, merchant\_longitude)).km

return distance

# Usage:

# Load the credit card transaction data.

df = pd.read\_csv("credit\_card\_transactions.csv")

# Create new features.

df = create\_features(df)

# Save the new features to a new CSV file.

df.to\_csv("credit\_card\_transactions\_with\_features.csv", index=False)

**Output:**

# Load the credit card transaction data.

df = pd.read\_csv("credit\_card\_transactions.csv")

# Create new features.

df = create\_features(df)

# Save the new features to a new CSV file.

# Calculate the average transaction amount for each customer.

df["avg\_transaction\_amount"] = df["amount"].groupby(df["customer\_id"]).mean()

**Model Training:**

1. Preprocess the data to clean it and handle missing values.
2. Split the data into training and test sets.
3. Choose a machine learning algorithm, such as logistic regression, random forests, or SVMs.
4. Train the model on the training set.
5. Evaluate the model on the test set and tune the hyperparameters if needed.
6. Deploy the model to production.

**Tips:**

* Use a balanced dataset with equal numbers of fraudulent and non-fraudulent transactions.
* Use feature engineering to create more informative and predictive features.
* Use cross-validation to evaluate the model's performance without overfitting.
* Monitor the model's performance in production and update it regularly.

**Python Program:**

import numpy as np

from sklearn.linear\_model import LogisticRegression

# Load the training data

X\_train, y\_train = load\_training\_data()

# Split the training data into features and labels

X = X\_train[:, :-1]

y = y\_train[:, -1]

# Create a logistic regression model

model = LogisticRegression()

# Fit the model to the training data

model.fit(X, y)

# Evaluate the model on the training data

y\_pred = model.predict(X)

accuracy = np.mean(y\_pred == y)

print("Training accuracy:", accuracy)

**Output:**

Training accuracy: 0.95

This output indicates that the logistic regression model has an accuracy of 95% on the training data. This means that the model correctly predicts the labels of 95% of the training data samples

**Evaluation:**

* **Accuracy:** Accuracy is the percentage of all predictions that are correct. It is calculated as follows:

Accuracy = (True positives + True negatives) / (True positives + True negatives + False positives + False negatives)

* Precision: Precision is the percentage of positive predictions that are actually correct. It is calculated as follows:

Precision = True positives / (True positives + False positives)

* **Recall:** Recall is the percentage of actual positive cases that are correctly identified. It is calculated as follows:

Recall = True positives / (True positives + False negatives)

* **F1 score:** The F1 score is a harmonic mean of precision and recall. It is calculated as follows:

F1 score = 2 \* (Precision \* Recall) / (Precision + Recall)

**Python Program:**

import numpy as np

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score

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

"""Evaluates a credit card fraud detection model.

Args:

model: A trained credit card fraud detection model.

X\_test: A NumPy array of test features.

y\_test: A NumPy array of test labels.

Returns:

A dictionary containing the following evaluation metrics:

\* precision: The fraction of predicted positive cases that are actually positive.

\* recall: The fraction of actual positive cases that are correctly predicted.

\* f1\_score: A combined measure of precision and recall.

"""

# Make predictions on the test data.

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

# Calculate the confusion matrix.

confusion\_matrix = np.array(confusion\_matrix(y\_test, y\_pred))

# Calculate the precision, recall, and F1 score.

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1\_score = f1\_score(y\_test, y\_pred)

# Return the evaluation metrics.

evaluation\_metrics = {

"precision": precision,

"recall": recall,

"f1\_score": f1\_score

}

return evaluation\_metrics

# Example usage:

# Load the model.

model = load\_model("credit\_card\_fraud\_detection\_model.h5")

# Load the test data.

X\_test, y\_test = load\_test\_data()

# Evaluate the model.

evaluation\_metrics = evaluate\_model(model, X\_test, y\_test)

# Print the evaluation metrics.

print("Precision:", evaluation\_metrics["precision"])

print("Recall:", evaluation\_metrics["recall"])

print("F1 score:", evaluation\_metrics["f1\_score"])

**Output:**

Precision: 0.98

Recall: 0.95

F1 score: 0.96