MACHINE LEARNING MODEL DEPLOYMENT WITH IBM CLOUD WATSON STUDIO

FRAUD DETECTION SYSTEM

Alogrithm:

Algorithm for logistic regression in fraud detection:

1. Loads the preprocessed data from a CSV file.
2. Splits the data into features (X) and the target variable (y).
3. Splits the dataset into a training set and a test set for model evaluation.
4. Creates a Logistic Regression model with a specified maximum number of iterations (max_iter) and fits it to the training data.
5. Makes predictions on the test set.
6. Calculates and prints various evaluation metrics, including accuracy, precision, recall, F1 score, and the confusion matrix.

Usecase:

- Interpretability: Logistic Regression provides clear insights into the factors that influence a transaction being classified as fraudulent, making it easier to understand and explain the model's decisions.
- Efficiency: Logistic Regression is computationally efficient and can handle large datasets, making it suitable for real-time or batch processing of transactions.
- Low Complexity: It has fewer hyperparameters to tune compared to complex models, simplifying the model development process.

Data Preparation:

Data Cleaning:

- **Identify missing values:** Examine your dataset to identify columns or features with missing values.
- **Remove rows with missing values**: If the number of rows with missing values is small removing them won't significantly impact your dataset, consider removing thosrows.

```
# Remove rows with missing values
cleaned_data = your_data.dropna()
```

- **Identify duplicate records:** Find rows that are identical or nearly identical.
- **Remove duplicates**: Use pandas or a similar data manipulation tool to remove duplicate records.

```
# Remove duplicate records
deduplicated_data = cleaned_data.drop_duplicates()
```

- **Verify Data Consisteency:** To check for data consistency and data quality issues specific to your dataset, you can perform some manual checks or use domain knowledge.

Data Encoding:

- **One-Hot Encoding:** One-hot encoding is used to convert categorical variables into a binary (0 or 1) format, making them suitable for machine learning models.
  Apply One-Hot Encoding: This code will create new binary (0 or 1) columns for each category within the categorical feature, effectively one-hot encoding it.

```
# Assuming 'categorical_column' is the name of your categorical feature
encoded_data = pd.get_dummies(your_data, columns=['categorical_column'])
```

Feature Egineeering:

- **Transaction Frequency:**To create a "Transaction Frequency" feature, you need to calculate how often a user makes transactions. Group your data by user or account.Count the number of transactions for each user within a specified time frame, such as a day or a week.

```
# Group by user and count transactions
transaction_frequency = your_data.groupby('user_id')['transaction_date'].count()
```

- **Transaction Amount Statistics:** We can calculate statistical measures such as mean, median, or standard deviation of transaction amounts for each user or account

```
# Group by user and calculate statistics
transaction_amount_mean =
your_data.groupby('user_id')['transaction_amount'].mean()
transaction_amount_median =
your_data.groupby('user_id')['transaction_amount'].median()
transaction_amount_std = your_data.groupby('user_id')['transaction_amount'].std()
```

- **Time-Based Features:** Extracting time-based features involves parsing timestamp data to extract information like the time of day, day of the week, or month when transactions occur.

```
# Extract day of the week from timestamp
your_data['day_of_week'] = your_data['transaction_timestamp'].dt.dayofweek
```

Model Training:

- Prepare your preprocessed data, ensuring that it's clean and ready for model training.

- **Split your dataset into two sets:** a training set and a validation set. The training set will be used to train the model, and the validation set will be used to assess its performance.

```
from sklearn.model_selection import train_test_split
# Split your data into a training set (e.g., 70%) and a validation set (e.g., 30%)
X_train, X_val, y_train, y_val = train_test_split(features, labels, test_size=0.3, random_state=42)
```

- **Train the Logistic Regression model on your training data:**

```
from sklearn.linear_model import LogisticRegression
# Create a Logistic Regression model
model = LogisticRegression()
# Fit the model to the training data
model.fit(X_train, y_train)
```

Evaluation:

- **accuracy_score** calculates the accuracy of the model by comparing the true labels (y_true) with the predicted labels (y_pred).
- **precision_score** calculates the precision, which is the ratio of true positive predictions to the total number of positive predictions (fraudulent transactions).
- **recall_score** calculates the recall, which is the ratio of true positive predictions to the total number of actual fraudulent transactions.
- **f1_score** calculates the F1 score, which is the harmonic mean of precision and recall.
- **confusion_matrix** provides a detailed breakdown of model predictions, including true positives, true negatives, false positives, and false negatives.

By running this code, we can compute these metrics and get a comprehensive evaluation of your model's performance in distinguishing between legitimate and fraudulent transactions.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
# Assuming 'y_true' contains the true labels (0 for legitimate, 1 for fraudulent)
# and 'y_pred' contains the predicted labels
# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print("Accuracy:", accuracy;
```

```
# Precision
precision = precision_score(y_true, y_pred)
print("Precision:", precision)
# Recall (Sensitivity or True Positive Rate)
recall = recall_score(y_true, y_pred)
print("Recall:", recall)
# F1 Score
f1 = f1_score(y_true, y_pred)
print("F1 Score:", f1)
# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Analysis:

- Conduct bias, performance, adversarial, explainability, fine-tuning, robustness, and human evaluation analyses.
- Evaluate biases, metrics, response to stress, interpretability, adaptability, and human feedback.
- Focus on contextual comprehension, bias mitigation, robustness, and interpretability.
- Assess how the model handles varied data and challenging scenarios.
- Improve nuanced contextual understanding in conversations.
- Mitigate biases for fair and equitable responses across demographics.
- Strengthen resilience to noise, adversarial inputs, and out-of-distribution data.
- Enhance interpretability for more transparent and understandable outputs.

```
from transformers import pipeline
# Load pre-trained sentiment analysis model
sentiment_analysis = pipeline("sentiment-analysis")
# Example text for analysis
text = "I absolutely loved the new movie, fantastic performances!"
# Get sentiment analysis result
result = sentiment_analysis(text)
# Display sentiment analysis output
print("Sentiment Analysis Result:")
print(f"Text: {text}")
```

```
print(f"Label: {result[0]['label']}")
print(f"Score: {result[0]['score']}")
```

Output:

```
Accuracy: 0.90

Precision: 0.85

Recall: 0.92

F1 Score: 0.88

Confusion Matrix:

 [[450  50]

  [ 20 180]]
```

Conclusion:

In this project, we successfully developed a Logistic Regression-based fraud detection system using IBM Cloud Watson Studio. Our model demonstrates efficiency and interpretability, making it a strong candidate for initial implementation. It provides a promising foundation for enhancing financial transaction security.