Data Science HW1 - 信用卡詐騙偵測系統

環境要求

Python 版本：

- 建議 Python 3.7 或更新版本。

套件需求：

- **Pandas**: 用於資料操作和處理。
- **NumPy**: 用於數值運算。
- **Scikit-learn**: 用於資料預處理、模型評估等。
- **LightGBM**: 用於訓練梯度提升模型。

如何執行程式

1. **準備資料**：
   - 確保有 `train_data.csv` 和 `test_data.csv` 兩個檔案，並放置在程式相同的目錄下。
   - `train_data.csv` 需要包含訓練所需的特徵（如 `att1`, `att3`, `att4`, ...）和目標欄位 `fraud`。
   - `test_data.csv` 則是需要進行預測的資料，並包含 `Id` 欄位用來識別資料。
2. **安裝必要的環境和套件**： 需要使用 Python，並安裝以下套件：

   ```
   pip install pandas numpy scikit-learn lightgbm
   ```

3. **執行程式**：
   - 在終端機或命令列執行：

     ```
     python fraud_detection.py
     ```

   - 程式將會：
     1. 加載資料。
     2. 進行特徵工程。
     3. 訓練 LightGBM 模型。
     4. 評估模型的性能。
     5. 生成 `submission.csv`，包含預測結果。

程式架構

主要模組與函式

1. **`load_data`**

- **目的**: 載入訓練與測試資料。
- **輸入**: CSV 檔案。
- **輸出**: pandas DataFrame 格式的資料。

```
def load_data():
    train_data = pd.read_csv('train_data.csv')
    test_data = pd.read_csv('test_data.csv')
    return train_data, test_data
```

2. `feature_engineering`
   - **目的**: 提取並新增有意義的特徵。
   - 方法：
     - 提取時間屬性並判斷是否為夜晚（22:00-05:00）。
     - 使用 haversine 公式計算商家與用戶之間的地理距離。
   - **輸入**: DataFrame。
   - **輸出**: 加工後的 DataFrame。

```
def feature_engineering(df):
    df['hour'] = pd.to_datetime(df['att1'], format='%H:%M').dt.hour
    df['is_night'] = (df['hour'] >= 22) | (df['hour'] <= 5)

    def haversine_distance(lat1, lon1, lat2, lon2):
        R = 6371
        phi1 = np.radians(lat1)
        phi2 = np.radians(lat2)
        delta_phi = np.radians(lat2 - lat1)
        delta_lambda = np.radians(lon2 - lon1)

        a = np.sin(delta_phi/2)**2 + np.cos(phi1) * np.cos(phi2) *
np.sin(delta_lambda/2)**2
        c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

        return R * c

    df['merchant_distance'] = haversine_distance(
        df['att12'], df['att13'],
        df['att15'], df['att16']
    )

    return df
```

3. `create_preprocessor`
   - **目的**: 構建數值與類別資料的前處理流程。

- 方法：
  - 數值資料：使用中位數填補缺失值，並進行標準化。
  - 類別資料：使用最頻繁值填補缺失值，並進行 One-Hot 編碼。
- **輸出**: ColumnTransformer。

```
def create_preprocessor():
    numeric_features = ['att4', 'att5', 'att10', 'merchant_distance', 'hour']
    categorical_features = ['att3', 'att6', 'att7', 'att8', 'att9']

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', Pipeline([
                ('imputer', SimpleImputer(strategy='median')),
                ('scaler', StandardScaler())
            ]), numeric_features),
            ('cat', Pipeline([
                ('imputer', SimpleImputer(strategy='most_frequent')),
                ('onehot', OneHotEncoder(handle_unknown='ignore'))
            ]), categorical_features)
        ])

    return preprocessor
```

4. `train_model`
   - **目的**: 訓練 LightGBM 模型。
   - 方法：
     - 使用前處理器處理資料。
     - 建立 LightGBM 資料集並訓練模型。
   - **輸出**: 訓練好的模型及前處理器。

```
def train_model(X_train, y_train, preprocessor):
    # Preprocess the data
    X_train_processed = preprocessor.fit_transform(X_train)

    # Create LightGBM dataset
    train_data = lgb.Dataset(X_train_processed, label=y_train)

    # Parameters
    params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.05,
        'feature_fraction': 0.9,
```

```
        'is_unbalance': True
    }

    # Train model
    model = lgb.train(
        params,
        train_data,
        num_boost_round=200
    )

    return preprocessor, model
```

5. `evaluate_model`
    - **目的**: 評估模型效能。
    - 方法：
        - 輸出分類報告與混淆矩陣。
    - **輸出**: 評估結果。

```
def evaluate_model(preprocessor, model, X_test, y_test):
    # Preprocess test data
    X_test_processed = preprocessor.transform(X_test)

    # Predict
    y_pred = (model.predict(X_test_processed) > 0.5).astype(int)

    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

def predict_submission(preprocessor, model, test_data):
    test_data = feature_engineering(test_data)

    # Preprocess test data
    X_test_processed = preprocessor.transform(test_data)

    # Predict
    predictions = (model.predict(X_test_processed) > 0.5).astype(int)

    submission = pd.DataFrame({
        'Id': test_data['Id'],
        'fraud': predictions
    })

    submission.to_csv('submission.csv', index=False)
    return submission
```

6. `predict_submission`
    - **目的**: 預測測試資料並產生提交檔案。
    - 方法：
        - 進行特徵工程與前處理。
        - 輸出預測結果到 CSV 檔案。

```
def predict_submission(preprocessor, model, test_data):
    test_data = feature_engineering(test_data)

    # Preprocess test data
    X_test_processed = preprocessor.transform(test_data)

    # Predict
    predictions = (model.predict(X_test_processed) > 0.5).astype(int)

    submission = pd.DataFrame({
        'Id': test_data['Id'],
        'fraud': predictions
    })

    submission.to_csv('submission.csv', index=False)
    return submission
```

7. `main`
    - 整合執行上述所有步驟，從資料載入到最終提交。

```
def main():
    # Load Data
    train_data, test_data = load_data()

    # Feature Engineering
    train_data = feature_engineering(train_data)

    # Split Features and Target
    X = train_data.drop('fraud', axis=1)
    y = train_data['fraud']

    # Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Create Preprocessor
    preprocessor = create_preprocessor()
```

```
    # Train Model
    preprocessor, model = train_model(X_train, y_train, preprocessor)

    # Evaluate Model
    evaluate_model(preprocessor, model, X_test, y_test)

    # Generate Submission
    submission = predict_submission(preprocessor, model, test_data)

if __name__ == "__main__":
    main()
```

資料前處理

步驟與目的

1. **缺失值處理**
   - **數值資料**: 使用中位數填補，避免極端值對結果的影響。
   - **類別資料**: 使用最頻繁值填補，確保類別資料一致性。
2. **特徵標準化**
   - 使用 Z-Score 標準化將數值資料轉換為均值為 0、標準差為 1 的分佈，提升模型收斂速度與穩定性。
3. **特徵編碼**
   - 使用 One-Hot 編碼將類別特徵轉換為二元向量，避免類別順序性的誤解。
4. **特徵工程**
   - **新增 `hour` 與 `is_night` 特徵**: 提取時間資訊以捕捉交易時間與夜晚欺詐行為的相關性。
   - **計算地理距離**: 使用 Haversine 公式計算用戶與商家的距離，作為地理特徵。

演算法流程

1. **資料載入**
   - 從檔案中讀取資料並檢查格式。
2. **資料清理與特徵工程**
   - 對每筆資料執行清理，並根據需求新增特徵。
3. **前處理管線建立**
   - 使用 `ColumnTransformer` 和 `Pipeline` 統一處理數值與類別特徵。
4. **資料分割**
   - 使用 `train_test_split` 將資料分為訓練集與測試集。
5. **模型訓練**

- 使用 LightGBM 訓練模型，並調整超參數以提升效能。

6. **模型評估**
    - 計算分類報告與混淆矩陣，檢查模型的準確率、精確率、召回率等指標。
7. **生成提交檔案**
    - 將測試資料進行預測並輸出為指定格式的提交檔案。