# Low-Level Design (LLD) Report for Fraud Transaction Detection

Author: Lokesh Bapte
Email: lokesh18.ml@gmail.com

# Table of Contents

## Introduction

This report outlines the Low-Level Design (LLD) for a Fraud Transaction Detection system. The system aims to classify transactions as fraudulent or non-fraudulent using machine learning models. This document details the design and implementation of the system, including data processing, feature engineering, model training, and evaluation.

## Objective

The objective of this project is to build a machine learning model that can accurately detect fraudulent transactions. The model will be trained on historical transaction data and evaluated based on various performance metrics.

## Architecture Overview

The architecture of the Fraud Transaction Detection system consists of the following main components:

1. Data Ingestion
2. Data Pre-processing
3. Exploratory Data Analysis (EDA)
4. Feature Engineering
5. Data Balancing
6. Model Training
7. Model Evaluation

Each component plays a critical role in ensuring the accuracy and robustness of the fraud detection model.

## Data Flow

1. **Data Ingestion:** Load the training and testing datasets from CSV files.
2. **Data Pre-processing:** Clean and preprocess the data to handle missing values, encode categorical variables, and extract relevant features.
3. **Exploratory Data Analysis (EDA):** Perform EDA to understand the data distribution and relationships between features.
4. **Feature Engineering:** Create new features based on existing data to improve model performance.
5. **Data Balancing:** Down-sample the majority class to address class imbalance.

6. **Model Training:** Train various machine learning models on the preprocessed data.
7. **Model Evaluation:** Evaluate the trained models using appropriate metrics and select the best model.

## Detailed Component Design

### Data Ingestion

**Description:** Load the training and testing datasets from CSV files.

```
from google.colab import drive

drive.mount('/content/drive')

file_path1 = '/content/drive/My Drive/Colab/fraudTrain.csv'

file_path2 = '/content/drive/My Drive/Colab/fraudTest.csv'

train_df = pd.read_csv(file_path1, index_col='Unnamed: 0')

test_df = pd.read_csv(file_path2, index_col='Unnamed: 0')
```

### Data Pre-processing

**Description:**

1. Convert transaction dates to datetime objects.
2. Extract hour and month from transaction dates.
3. Drop non-useful columns.
4. Encode categorical features using Weight of Evidence (WOE)

```
# Convert transaction date to datetime

train_df['trans_date_trans_time'] =
pd.to_datetime(train_df['trans_date_trans_time'], format='mixed')

test_df['trans_date_trans_time'] =
pd.to_datetime(test_df['trans_date_trans_time'], format='mixed')

# Extract hour and month from transaction date

train_df['hour'] = train_df['trans_date_trans_time'].dt.hour

test_df['hour'] = test_df['trans_date_trans_time'].dt.hour

train_df['month'] = train_df['trans_date_trans_time'].dt.month

test_df['month'] = test_df['trans_date_trans_time'].dt.month
```

```
# Remove non-useful columns

columns_to_drop = ['first', 'unix_time', 'dob', 'cc_num', 'zip', 'city',
'street', 'state', 'trans_num', 'trans_date_trans_time']

train_df = train_df.drop(columns_to_drop, axis=1)

test_df = test_df.drop(columns_to_drop, axis=1)

# Clean merchant column

train_df['merchant'] = train_df['merchant'].apply(lambda x: x.replace('fraud_',
''))

# Encoding categorical features

train_df['gender'] = train_df['gender'].map({'F': 0, 'M': 1})

for col in ['job', 'merchant', 'category', 'lat', 'last']:

                 train_df[col]    =    WOEEncoder().fit_transform(train_df[col],
train_df['is_fraud'])
```

### Exploratory Data Analysis

**Description:** Perform EDA to understand the data distribution and relationships between features.

```
# Gender distribution visualization

fig, axb = plt.subplots(ncols=2, nrows=1, figsize=(15, 8))

explode = [0.1, 0.1]

train_df.groupby('gender')['is_fraud'].count().plot.pie(explode=explode,
autopct="%1.1f%%", ax=axb[0])

ax = sns.countplot(x="gender", hue="is_fraud", data=train_df, ax=axb[1])

for p in ax.patches:

         ax.annotate(f'{p.get_height()}',  (p.get_x()  +  p.get_width()  /  2.,
p.get_height()), ha='center', va='center', xytext=(0, 10), textcoords='offset
points')

plt.title("Distribution of Gender with Fraud Status")

plt.xlabel("Gender")

plt.ylabel("Count")
```

```
plt.show()


# Fraud vs non-fraud pie chart
is_fraud = train_df["is_fraud"].value_counts()
plt.figure(figsize=(10, 6))
plt.pie(is_fraud, labels=["No", "YES"], autopct="%0.0f%%")
plt.title("is_fraud Counts")
plt.tight_layout()
plt.show()
```

**Feature Engineering**

**Description:** Create new features such as hour and month from transaction dates.

```
# Extract hour and month from transaction date
train_df['hour'] = train_df['trans_date_trans_time'].dt.hour
test_df['hour'] = test_df['trans_date_trans_time'].dt.hour
train_df['month'] = train_df['trans_date_trans_time'].dt.month
test_df['month'] = test_df['trans_date_trans_time'].dt.month
```

**Data Balancing**

**Description:** Down-sample the majority class to address class imbalance.

```
# Down-sample the majority class
No_class = train_df[train_df["is_fraud"] == 0]
yes_class = train_df[train_df["is_fraud"] == 1]
No_class = resample(No_class, replace=False, n_samples=len(yes_class))
down_samples = pd.concat([yes_class, No_class], axis=0)
```

### Model Training

**Description:** Train various machine learning models on the preprocessed data.

```
# Split the data into training and test sets

X = down_samples.drop("is_fraud", axis=1)

y = down_samples["is_fraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=65)

# Scale the features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Initialize models

models = {

    "Logistic Regression": LogisticRegression(),

    "Decision Tree": DecisionTreeClassifier(),

    "Random Forest": RandomForestClassifier(),

    "XGBoost": XGBClassifier(),

    "SVM": LinearSVC(),

    "Naive Bayes": GaussianNB()

}

# Train and evaluate each model

for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    report = classification_report(y_test, y_pred)

    matrix = confusion_matrix(y_test, y_pred)

    print(f"Model: {name}")
```

```
print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{report}")

print(f"Confusion Matrix:\n{matrix}\n")
```

**Model Evaluation**

**Description:** Evaluate the trained models using appropriate metrics and select the best model. **Evaluation Metrics:**

- **Accuracy:** Proportion of correctly classified instances among the total instances.
- **Precision:** Proportion of true positive instances among the instances predicted as positive.
- **Recall (Sensitivity):** Proportion of true positive instances among the actual positive instances.
- **F1-Score:** Harmonic mean of precision and recall, providing a balance between the two.
- **Confusion Matrix:** A table showing the counts of true positives, true negatives, false positives, and false negatives.

## Classes and Functions

1. **Data Ingestion:**
    - `load_data(file_path)`: Load data from the specified file path.
2. **Data Pre-processing:**
    - `preprocess_data(df)`: Preprocess the data including datetime conversion, feature extraction, and encoding.
3. **EDA:**
    - `plot_gender_distribution(df)`: Plot gender distribution.
    - `plot_fraud_distribution(df)`: Plot fraud vs non-fraud distribution.
4. **Feature Engineering:**
    - `extract_features(df)`: Extract new features from existing data.
5. **Data Balancing:**
    - `balance_data(df)`: Down-sample the majority class to address class imbalance.
6. **Model Training:**
    - `train_models(X_train, y_train)`: Train various machine learning models.

- `evaluate_models(X_test, y_test, models)`: Evaluate trained models using appropriate metrics.

## Libraries and Dependencies

- `pandas`: For data manipulation and analysis.
- `numpy`: For numerical operations.
- `sklearn`: For machine learning algorithms and evaluation metrics.
- `xgboost`: For the XGBoost model.
- `imblearn`: For handling class imbalance.
- `matplotlib` and `seaborn`: For data visualization.

## Testing Strategy

1. **Unit Testing:** Test individual functions such as `load_data`, `preprocess_data`, and `extract_features` to ensure they work as expected.
2. **Integration Testing:** Test the entire pipeline from data ingestion to model evaluation to ensure all components work together seamlessly.
3. **Performance Testing:** Evaluate the performance of different models using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.

## Conclusion

This Low-Level Design (LLD) report provides a comprehensive guide for implementing a fraud transaction detection system. The system includes data ingestion, pre-processing, feature engineering, data balancing, model training, and evaluation. The detailed component design, code snippets, and testing strategy ensure the robustness and accuracy of the fraud detection model.