Credit Card Anomaly Detection: Project Report

Development Team

December 26, 2025

1 Introduction

This report documents the development and restructuring journey of the Credit Card Anomaly Detection project. The primary objective was to organize an existing codebase into a professional, modular structure, verify its integrity, and enhance its performance using advanced Deep Learning techniques.

2 Project Restructuring

The initial phase involved transforming a flat script-based project into a scalable architecture. The files were organized as follows:

- **config/**: Centralized configuration (`config.yaml`) for hyperparameters and paths.
- **src/**: Core modules including `data_loader.py`, `preprocessor.py`, `autoencoder.py`, and `trainer.py`.
- **models/**: Directory for saving trained artifacts (`.keras` models and scalers).
- **Root**: Entry points `main.py` (training) and `predict.py` (inference).

3 Integrity Verification & Fixes

During the initial audit, several critical issues were identified and resolved:

3.1 Configuration Management

The original scripts had hardcoded paths. We introduced a centralized loading mechanism in `src/utils.py` using `PyYAML` to ensure consistency across `main.py` and `predict.py`.

3.2 Prediction Pipeline Repair

The `predict.py` script was found to be non-functional due to:

1. Import errors (referencing non-existent functions).

2. Path mismatches (looking for `.h5` instead of `.keras`).

3. Missing preprocessing (predicting on raw instead of scaled data).

These were fixed by rebuilding the script to load the saved `StandardScaler` and ensuring the input data shape matched the model's expected input dimension (29 features).

4 Technical Approach

4.1 Autoencoder Architecture

We utilized an Autoencoder, an unsupervised neural network, to learn the representation of *normal* transactions.

- **Encoder**: Compresses input data into a lower-dimensional latent space.
- **Decoder**: Reconstructs the input from the latent representation.
- **Loss Function**: Mean Squared Error (MSE). High reconstruction error indicates an anomaly.

4.2 Efficiency Improvements

To improve robustness and detection capability, the following enhancements were implemented:

- **Robust Scaling**: Switched from `StandardScaler` to `RobustScaler` to handle outliers in the input features effectively.
- **Regularization**: Added `Dropout` (0.2) and `BatchNormalization` layers to prevent overfitting and improve training stability.
- **Dynamic Thresholding**: Implemented a recall-based strategy to automatically calculate the anomaly threshold, targeting 90% capture of known fraud cases during evaluation.

5 Results

The system was successfully verified with an end-to-end test.

- **Training**: The model converged with decreasing validation loss.
- **Inference**: The `predict.py` demo successfully distinguished between normal and fraud samples in a simulated live environment.
- **Documentation**: A comprehensive `README.md` and project walkthrough were created to ensure ease of use.

6 Conclusion

The project successfully transitioned from a prototype to a robust, well-structured application. The inclusion of automated integrity checks and sophisticated preprocessing techniques ensures high reliability for future development.