BATTERY FAILURE PREDICTION PROJECT REPORT

Prepared by: SHERIN SAMUEL
Date: May 08, 2025

Introduction

The project predicts battery failure using the NASA battery dataset, encompassing data processing, feature engineering, preprocessing, exploratory data analysis (EDA), class balancing with SMOTE, model development, hyperparameter tuning, SHAP analysis for interpretability, and deployment through a Flask API, Streamlit dashboard, and Power BI dashboard. The goal is to deliver a robust, interpretable solution for battery management.

Data Processing and Feature Engineering

The project starts by processing NASA .mat files (B0005, B0055, B0056) to extract discharge cycle data. Features like average voltage, current, temperature, and capacity are computed, alongside derived metrics: State of Charge (SOC) = (voltage - 3.0) / (4.2 - 3.0), State of Health (SOH) = (capacity / 2.0) * 100, internal resistance (voltage drop / discharge current), and a failure flag (capacity < 1.4 Ah). Ambient temperatures are set (24°C for B0005, 4°C for B0055/B0056). The data is saved as nasa_battery_data_combined.csv.

Data Preprocessing

Preprocessing ensures data quality by filling missing numerical values with medians, removing outliers using the Interquartile Range (IQR) method, encoding battery_id with LabelEncoder, and standardizing features with StandardScaler. The resulting dataset, nasa_battery_data_preprocessed.csv, has 370 rows and 12 columns with no missing values, ready for modeling.

Exploratory Data Analysis (EDA)

EDA generates visualizations including capacity degradation plots, correlation heatmaps, and failure label distributions. Key insights show capacity/SOH strongly correlate negatively with failure (-0.882548), while current correlates positively (0.778305). Plots are saved in eda_plots, revealing degradation trends and feature relationships.

Class Balancing with SMOTE

To address class imbalance in the dataset (more failure instances: 66.49% failure vs. 33.51% no-failure), SMOTE is applied during training of Random Forest and XGBoost models. SMOTE oversamples the minority class ("No Failure") by generating synthetic samples, ensuring balanced training data and improving model performance on minority class predictions, as seen in the improved recall scores (e.g., 0.9796 for Random Forest "Failure").

Model Development

Multiple models are developed:

- **Random Forest**: Achieves 98.65% accuracy with SMOTE, with top features time (28.51%), capacity (27.09%), and voltage (22.49%).
- **XGBoost**: Also uses SMOTE, reaching 99% accuracy post-tuning.

- **One-Class SVM**: Detects anomalies with 96% accuracy, trained on non-failure data.
- **LSTM**: Handles sequential data (sequence length=20), achieving 97% accuracy with class weights.
- **Ensemble**: Combines XGBoost, One-Class SVM, and LSTM (weights: 0.5 LSTM, 0.3 XGBoost, 0.2 SVM), achieving 100% accuracy, though evaluation indicates potential overfitting (no "No Failure" predictions).
  Predictions are saved in predictions (e.g., ensemble_predictions.csv), and models in models (e.g., rf_model.joblib).

Hyperparameter Tuning

Grid Search enhances model performance:

- **XGBoost**: Tunes max_depth (3, 5, 7), learning_rate (0.01, 0.1, 0.3), n_estimators (100, 200), selecting max_depth=3, learning_rate=0.01, n_estimators=100 (F1-score 0.99 for "Failure").
- **One-Class SVM**: Tunes nu (0.05, 0.1, 0.2), gamma (scale, auto, 0.1), selecting nu=0.05, gamma=scale (F1-score 0.97).
- **LSTM**: Manually tunes sequence_length (10, 20) and units (50, 100), selecting sequence_length=10, units=100 (F1-score 1.0).
  Tuned models are saved (e.g., lstm_model_tuned.h5).

SHAP Analysis for Interpretability

SHAP (SHapley Additive exPlanations) analysis is applied to the Random Forest and XGBoost models to interpret predictions. For Random Forest, SHAP values confirm time, capacity, and voltage as key drivers of failure predictions, with high time values increasing failure likelihood. For XGBoost, SHAP highlights similar features, showing how lower capacity values contribute to higher failure probabilities. This interpretability ensures the models' decisions are transparent and actionable for battery management.

Flask App Development

A Flask app (app.py) enables real-time predictions on localhost:5000 via a /predict endpoint. It accepts JSON input (features like cycle, voltage), loads pre-trained models (XGBoost, One-Class SVM, LSTM), normalizes data with MinMaxScaler, and generates ensemble predictions (0.5 LSTM, 0.3 XGBoost, 0.2 SVM). The app includes logging, error handling, and returns JSON results (e.g., predicted_failure, ensemble_prob).

Streamlit Dashboard

A Streamlit dashboard (dashboard.py) integrates with the Flask API, sending 25 sample rows for prediction and displaying results in a table. Titled "Battery Failure Prediction Dashboard," it provides an interactive interface with error handling for failed API requests.

Power BI Dashboard

The Power BI dashboard visualizes failure probabilities:

- **Heatmap**: Shows ensemble_prob across battery_id and cycle, with conditional formatting (green to red). An outlier (0.52 at cycle 42, battery_id 2) is highlighted.

- **Column Chart**: Displays average probability per battery_id with data labels.
- **Line Chart**: Plots probability over cycles 1–78 for clarity.
- **Slicers**: Filters for battery_id and cycle (1–78).
- **KPI Card**: Shows average ensemble_prob in dark red.
  The dashboard, titled "Battery Failure Prediction Dashboard," uses the "Executive" theme and is exported as Battery_Failure_Dashboard.pdf.

Conclusion

This project delivers a robust battery failure prediction solution with high accuracy (ensemble: 100%), enhanced by SMOTE, hyperparameter tuning, and SHAP interpretability. The Flask API, Streamlit, and Power BI dashboards ensure practical deployment. Future work could address ensemble overfitting, integrate real-time data, and explore additional failure indicators.