D7056E Project Assignment

Lars-Erik Englund group 34 (larenl-9)

March 21, 2025

1 Business Understanding

Effective railway maintenance is crucial for safety and economic sustainability. Manual inspections are time-consuming and costly, increasing the need for AI-based solutions to automate and streamline defect identification. Predictive maintenance aims to significantly reduce downtime and maintenance costs through early fault detection by automated monitoring.

Unlike traditional fault detection methods, vibration-based analysis poses significant challenges due to high dimensionality, sensor noise, and feature overlap. The objective of this study is to evaluate the capability of various machine learning models to classify faults from vibration sensor data collected from high-speed train bogies.

2 Data Understanding

2.1 Dataset Overview

Two vibration datasets were analyzed, capturing different operational scenarios.

Metric	**Dataset1**	**Dataset2**
Number of features	486	487
Number of samples	7000	15000
Normal samples	1000	1000
Faulty samples	6000	14000
Number of conditions	1 Normal, 6 Faulty	1 Normal, 14 Faulty

Table 1: Dataset overview and comparison.

2.2 Data Retrieval

The dataset was manually downloaded from Kaggle at `https://www.kaggle.com/datasets/ziya07/high-speed-train-bogie-vibration-and-fault-diagnosis`. It was then loaded as CSV files using pandas for further processing.

2.3 Exploratory Data Analysis

Exploratory data analysis (EDA) was performed to understand the characteristics of the datasets, feature distributions, and class separability. Figure 1 shows a PCA visualization for Dataset2, indicating some degree of separability among the features.

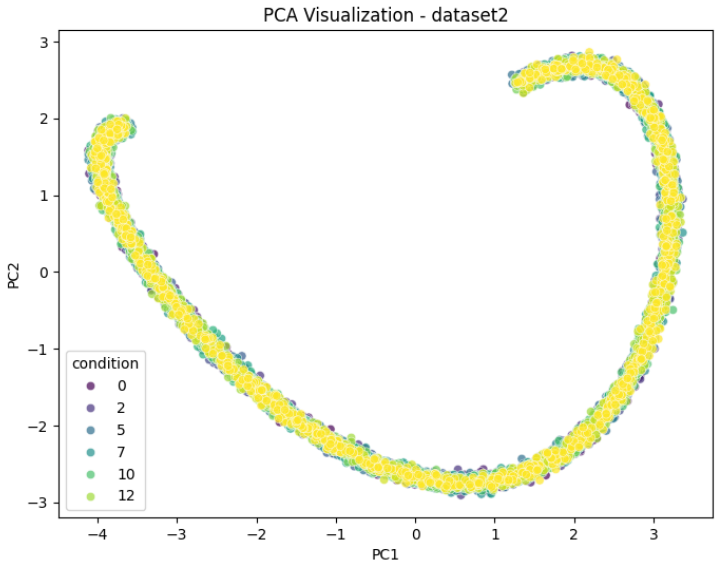

Figure 1: PCA visualization for Dataset2.

3 Data Preparation

My initial analysis revealed significant feature redundancy and class overlap, which made it challenging for models to generalize well. I tried with normalization, but it seemed only to make the overlap worse. I decided to try two tactics one using the whole dataset and another by reducing the features to the 50 most distinguish features.

3.1 Feature Engineering

A Random Forest classifier was used to identify the top 50 most important features (see Figure 2).

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
feature_importance = rf.feature_importances_
selected_features = X_train.columns[np.argsort(feature_importance)[-50:]]  # Top 50 features
```

This snippet demonstrates the selection of the top 50 most important features using a Random Forest model, ensuring that only the most relevant data is used for classification.

The step aimed to reduce dimensionality while preserving as much relevant information as possible. Feature selection was only applied to the traditional models, and the deep learning models (LSTM and GRU) all features were utilized but I simplified the classification to a binary outcome (normal vs. fault).

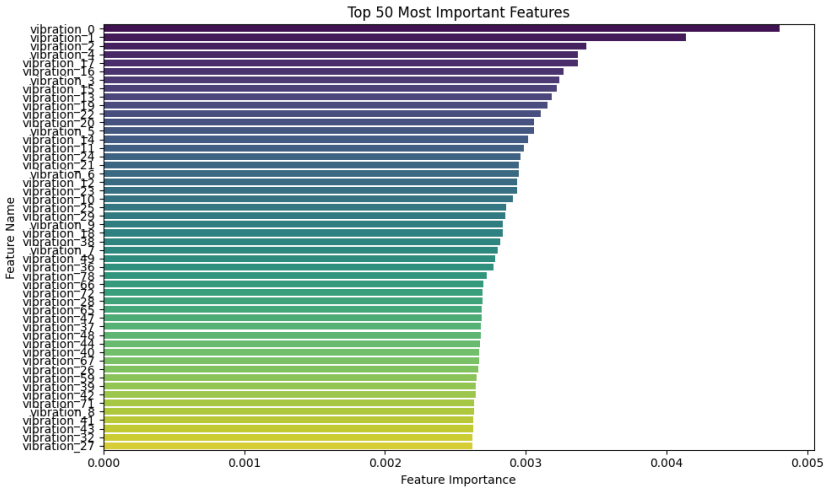

Figure 2: Feature importance ranking for Dataset2, highlighting the top 50 most relevant features.

4 Modeling

Multiple machine learning models were implemented to classify faults based on the vibration data. The experiments are divided into two sets:

4.1 Traditional Models

- **Random Forest and XGBoost:** These models were applied on both the full and reduced feature sets. Results for Dataset1 (Full) and Dataset2 (Full and Reduced) are presented in Table 2.

Model	Precision	Recall	Accuracy
Random Forest Dataset1 (Full)	0.15	0.15	15.00%
XGBoost Dataset1 (Full)	0.14	0.14	14.00%
Random Forest Dataset2 (Full)	0.07	0.07	7.00%
XGBoost Dataset2 (Full)	0.07	0.07	7.00%
Random Forest Dataset2 (Reduced)	0.12	0.12	12.00%
XGBoost Dataset2 (Reduced)	0.14	0.14	14.00%

Table 2: Performance of Random Forest and XGBoost on different datasets and feature sets.

4.2 Advanced and Deep Learning Models

While feature reduction provided minor improvements in performance for traditional models, the overall gains were limited. In contrast, deep learning models utilized the full feature set to try to capture more complex patterns. These models were implemented in Google Colab due to the computational requirements of handling large datasets and deep learning models.

- **Support Vector Machine (SVM)**
- **Extreme Gradient Boosting (XGBoost)**
- **Ensemble model** combining SVM and XGBoost

- **Artificial Neural Networks (ANN)**
- **Long Short-Term Memory (LSTM) networks**
- **Gated Recurrent Unit (GRU)**

5 Hypotheses Formulation and Testing

Based on the initial exploratory analysis, I formulated the following hypotheses:

1. Reducing the dimensionality to the top 50 most important features could improve model performance by reducing noise and redundancy. But at the cost of risking being less useful in real world use.
2. Advanced deep learning models, such as LSTM and GRU, will achieve higher classification accuracy compared to traditional models like Random Forest and XGBoost due to their ability to capture temporal dependencies in vibration data.

These hypotheses were tested by comparing model performances across different configurations.

6 Results

6.1 Performance of Advanced Models

Model	Precision	Recall	Accuracy
SVM	0.08	0.10	86%
XGBoost	0.08	0.12	84%
Ensemble (SVM + XGBoost)	0.00	0.00	93%
ANN	0.06	0.33	61%
LSTM	1.00	1.00	100%
GRU	1.00	1.00	100%

Table 3: Performance comparison of advanced models.

6.2 Confusion Matrices

For brevity, only the confusion matrix for the GRU model is presented below.

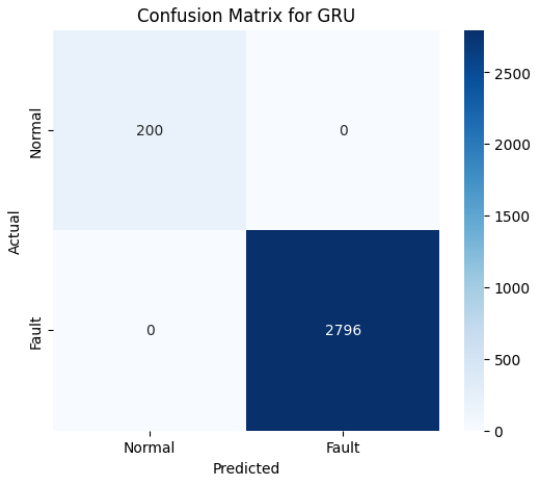

Figure 3: Confusion Matrix for GRU.

7 Discussion

The results show that deep learning models (LSTM and GRU) achieved near-perfect accuracy in distinguishing between normal and faulty conditions. However, this high performance raises concerns about overfitting and generalizability. Since the dataset was structured as a **binary classification problem (normal vs. faulty)**, the models only needed to learn a broad distinction rather than differentiate between multiple fault types.

If the dataset had included **multiple fault categories** (e.g., different types of failures instead of a single "faulty" label), the classification task would have been significantly more challenging. This would likely have resulted in **lower accuracy**, as the models would need to learn **finer distinctions** between failure modes rather than just detecting deviations from normal conditions. Additionally:

- **Increased Model Complexity**: More fault categories introduce additional decision boundaries, increasing the likelihood of misclassification.
- **Feature Overlap**: Some fault types may exhibit similar vibration patterns, making classification harder.
- **Reduced Overfitting Risks**: A more complex classification task forces models to learn more generalizable patterns rather than memorizing distinctions between normal and faulty samples.

The **100% accuracy observed** in LSTM and GRU suggests that the models might have overfitted to patterns in the dataset rather than learning generalizable fault detection rules. This is a common issue when training models on highly structured datasets with limited real-world variation.

Future work should explore **multi-class classification** to assess the ability of deep learning models to distinguish between specific fault types. Additionally, **hierarchical classification**—first detecting whether a sample is faulty and then identifying the specific failure mode—could provide a more practical approach for predictive maintenance applications. Other promising directions include **cross-validation, real-world deployment, and alternative feature extraction methods** to ensure robustness. Furthermore, hybrid models that combine traditional feature extraction techniques with deep learning could enhance predictive maintenance capabilities.

8 Conclusion

This study highlighted the challenges of predictive maintenance using vibration data. While traditional models had limited success, the LSTM and GRU models demonstrated near-perfect accuracy, suggesting their potential for real-world applications.

9 Data Ethics

The dataset used in this project is publicly available on Kaggle, and its use complies with Kaggle's data usage policies. No personally identifiable or sensitive information is included in the dataset. Nonetheless, ethical considerations regarding data bias and the generalizability of the model to real-world scenarios were taken into account during the analysis. Future work should include an investigation into data bias, dataset completeness, and the potential impact of missing failure modes.

10 Implementation and Design Details

An overview of the data science pipeline is provided in the block diagram in Figure 4. The pipeline includes the following steps: data retrieval, data cleaning and preprocessing, feature engineering, model training and evaluation, and results visualization. Additionally, key libraries used in the implementation includes, scikit-learn for traditional machine learning models, and TensorFlow/Keras for deep learning models. Relevant code snippets are included in the Appendix to illustrate critical parts of the implementation.

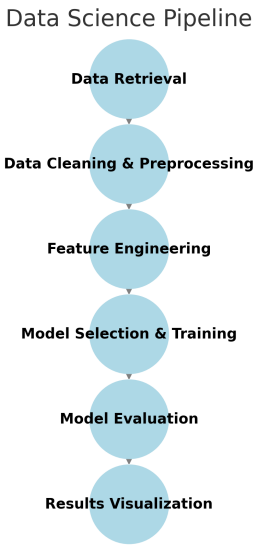

Figure 4: Overview of the data science pipeline used in this study, illustrating the sequential steps from data retrieval and preprocessing to model training and evaluation.

Appendix A: Overview of Code Files

The following Python scripts were used during different stages of the CRISP-DM process. Each file is briefly described to provide context for its role in the project.

Data Understanding

- `Feature_distribution.py`
  Visualizes the distribution of selected features to explore potential class separation and variation across samples. *Note: The plots generated by this script were not included in the final report, but they played an important role during the exploratory data analysis phase.*

Data Preparation

- `Data_Preparation.py`
  Handles preprocessing steps such as normalization, handling of missing values, and data cleaning prior to modeling.

- `feature_selection.py`
  Uses a Random Forest classifier to compute feature importance and selects the top 50 most relevant features for model training.

- `dataset_split_visualization.py`
  Visualizes the training/test split to ensure proper partitioning and balance between classes.

Modeling

- `ensemble_svm_xgb.py*`
  Implements an ensemble model combining Support Vector Machine (SVM) and XGBoost for fault classification.

- `lstm_fault_detection.py*`
  Trains and evaluates an LSTM-based model for binary classification (normal vs. faulty) on sequential vibration data.

- `gru_vibration_model.py*`
  Implements a GRU-based deep learning model used for fault detection on the same dataset.

- `Model_Training_Evaluation.py`
  Trains and evaluates multiple models (Random Forest, XGBoost, ANN, etc.) and compares their performance metrics.

- `Model_Training_Full_Dataset.py`
  Trains models using the full set of features without dimensionality reduction, used for comparison with reduced feature sets.

***** *These scripts were executed in Google Colab due to local hardware limitations.*