Predictive Maintenance using Hard Disk Telemetry

1. Scenario and Dataset

A PC contains multiple components and has many points of failure. With each hardware component acting as a potential point of failure. Nonetheless, a key component of PCs is the hard disk - where data is stored and where read/write operations happen. By observing the hard disk telemetry, we will be able to understand the disk health, similarly to disk health monitoring tools and metrics retrieved from consumer and enterprise hardware.

We will investigate real-world hard disk failure telemetry provided by Backblaze. This dataset contains key statistics of a hard disk, with more than two hundred data points of Self-Monitoring, Analysis, and Reporting Technology (SMART) codes per hard disk every day.

We will focus on a single brand of hard disks in this dataset, instead of the full dump, with the following assumptions:

1. Dell only mounts hard disks from a handful of brands (e.g., Dell proprietary disks). There is no need to mix varying brands into a model.

2. Disks from different brands exhibit different performance thresholds. For example, a temperature of 80°C might be normal for Brand A but anomalous for Brand B. By segregating the model to be brand specific, it would be able to “contextualize” to the brand.

2. Exploratory Data Analysis (EDA) and Findings

The objective of EDA in this project is to understand the distribution and behavior of normal disk operations versus failing or anomalous ones. We also want to then understand the data we have generated after performing data preprocessing, which will be used to train our models.

Key findings of the EDA:

Category	**Normal disk**	**Anomalous disk**
I/O (Read/Write)	Stable	Erroneous raw read error rate; Spikes in reallocated sector count
Power-on hours	Gradual Decrease	Gradual Reduction of power till failure
Temperature	Moderate (< 60°C)	High

Normal disks generally have stable behaviours, with SMART telemetry within expected ranges. However, anomalous disks have spikes present in the different categories. This makes the anomalous detection a difficult task as given the telemetry in different categories and their correlation with one another, it is difficult to determine if a specific data point is telling of a failure.

Through EDA, we were able to identify key SMART codes, and generate new features based on these codes for model training. These new features mostly leverage on windows (time periods) to generate and lookback values of a specific hard disk based on its serial number.

3. Modeling Approach

In this project we looked at using autoencoders to perform anomaly detection, a form of unsupervised learning. Although this dataset provided a label, which is the "failure" column, we decided to proceed with this approach as the proportion of "failed" to "normal" data is extremely low (~0.15%). This suggests that data is highly skewed even when we try to train the model and it is subjected to overfitting. As such, a supervised learning model would not be appropriate. Instead, an unsupervised model like autoencoders, which learns the data patterns of "normal" disks would be useful to identify data points which do not fall in the expected distribution.

An additional point would be autoencoders are good for data with high dimensionality as it is able to compress and perform dimensionality reduction as part of its encoding.

As explained in section 2, feature engineering was performed to generate lookback metrics for each data point, which are used for the model training. The final input dimension for the model is 105, with a mix of SMART code values available from the dataset and the feature engineered.

The autoencoder will be a 3-layered neural network, with the following (tuned) dimension progression for the encoder - 105 > 128 > 64 > 32 > 8 - and the reverse for the decoder. There will be dropouts between all layers and early stopping during training if the loss is increasing instead of decreasing more than the set threshold. This is to prevent model overfitting. We will also be using LeakyRelu as the activation function to prevent diminishing gradients during backward propagation when training the model's weights.

We will evaluate the model based on the reconstruction error (MSE) and Precision, Recall, F1-score (based on the "failure" column provided by the original dataset). To determine the predicted values for failure, we calculated a "threshold" value whereby it is the n^{th} percentile of the test MSE. We will then apply the threshold to generate the autoencoder's prediction. Additionally, we have also generated a visualization between the MSE distribution of anomalous, normal data and threshold to understand the distribution between the two datasets.

4. Optimization Techniques

1. Leaky ReLU

Introduced the Leaky ReLU activation function, which uses a small negative slope for negative inputs. This helps prevent the "dying ReLU" problem and improves model activation, especially for sparse gradients.

2. Dropout Rate

Applied dropout regularization during training to randomly deactivate a fraction of neurons. This mitigates overfitting by forcing the network to generalize better and prevents co-adaptation of features.

3. Scheduler: CosineAnnealingLR

Utilized a cosine annealing learning rate scheduler to cyclically adjust the learning rate following a cosine curve. This promotes better training stability and helps the model adapt efficiently over epochs.

4. Adaptive Learning with AdamW

Adopted the AdamW optimizer, which features adaptive learning rates and decoupled weight decay. This optimizer accelerates convergence while providing effective regularization.

5. Robust Loss Methods: nn.SmoothL1Loss

Employed the Smooth L1 Loss function, which offers a balance between L1 and L2 losses. It delivers higher precision on small errors and is less sensitive to outliers compared to Mean Squared Error (MSE).

6. Batch Training

Implemented batch training to update model weights based on mini-batches of data. This approach enhances training speed and efficiency, allowing more epochs within the same computation time.

Model performance between baseline and optimizations before and after

	Baseline Vanilla Autoencoder	Optimized Autoencoder
Anomaly Recall	0.16 at a Threshold of 95%	0.51 at a Threshold of 95%
Epoch (on T4 GPU)	450	200

5. Integration on Dell PCs

Assuming that the required environment is set up in Dell PCs (e.g. Python 3.10++, TensorFlow, PyTorch), these are the proposed workflow to deploy the autoencoder model.

This service will then read the SMART metrics using in-built tools, such as `smartctl` to retrieve real-time logs from the PC. The telemetry will then be preprocessed to generate the new features explained in section 2. The log will run against the autoencoder and the MSE will be generated. Given an agreed threshold, the service will compare the MSE against this value and send alerts, or logs when anomaly is detected. Additionally, the alert can also categorize the potential fault through existing libraries such as SHAP.

This service can run through the following methods:

1. Embed the model as a background service daemon. In this case, we would need to convert the service into a low-level language like C++.

2. Containerized as RESTful API service

3. Containerized as a service integrated with systemd