Anomalous Sound Detection in Industrial Equipment

Alexis BALESTRA, Gabriel LAFONT–MILLET

May 30, 2025

Abstract

This report details a machine learning approach to anomalous sound detection (ASD) in industrial equipment, replicating aspects of the well-known DCASE 2020 challenge. The objective is to identify unknown anomalous sounds emitted from machines, specifically focusing on the ”slide rail” machine type, given only normal sound samples for training. We utilized a subset of the d MIMII datasets, comprising 10-second single-channel audio recordings. Our methodology focuses on developing an unsupervised model capable of learning the characteristics of normal operation from the provided training data and subsequently assigning anomaly scores to unseen sounds. In this work, we specifically investigate the performance of two distinct methodologies for unsupervised anomalous sound detection: a distance-based approach using k-nearest neighbors (k-NN) and a reconstruction-error-based method employing autoencoders.

1 Introduction

In this challenge, we are not supposed to use any anomalous sound during the training part. This means that, contrary to the usual ways, our classifier should be able to classify a sound into the categories ”anomaly” or ”normal”, without ever being trained on a single ”anomaly” audio file.

Our first instinct was to manage to transform the audio files to a space with a notion of distance, allowing us to classify any audio file too far from any ”normal” audio as an ”anomaly”. This is how we found the Mel frequency cepstral coefficient method, which transforms audio files into a vector. This gave us a solid base to start our experiments.

For the metric, we chose to use, as advised, the area-under-curve Receiver-operating-characteristic (AUC-ROC) metric because it is one of the best metrics to use since our model will output the probability for the binary classification and not a simple boolean. It is independent of any threshold to determine whether the sound is anomalous or not, and thus more adapted for our use case.

2 Data preparation

Since there is no class imbalance in the training dataset (for the ids) and the testing datasets (ids and anomaly/normal), there is no need to do balancing.

We divided the testing dataset in a 60/40 repartition for test/validation datasets, making sure those datasets were also balanced.

We also kept in mind the possibility to use dimensionality reduction to simplify our data if we wanted to speed up our algorithms, with for example the T-distributed Stochastic Neighbor Embedding algorithm that can be applied once the sound is transformed into a vector.

The dataset we used is in the form of .wav files. We used panda to create dataframes with the file paths, the labels (whether anomalous or not), the ids of the machine. In the training dataset, the labels are useless since all the audio samples are normal, the labels are only used for validation.

Preprocessing strategies were assessed using k-NN models on the validation data, using the Area Under the Receiver Operating Characteristic Curve (AUC ROC) metric. K-NN models were selected for their training speed and ease of comparison over autoencoders. We anticipate that effective preprocessing for k-NN will translate to improved performance with autoencoders.

We also used feature scaling, specifically using StandardScaler to normalize features before training our models.

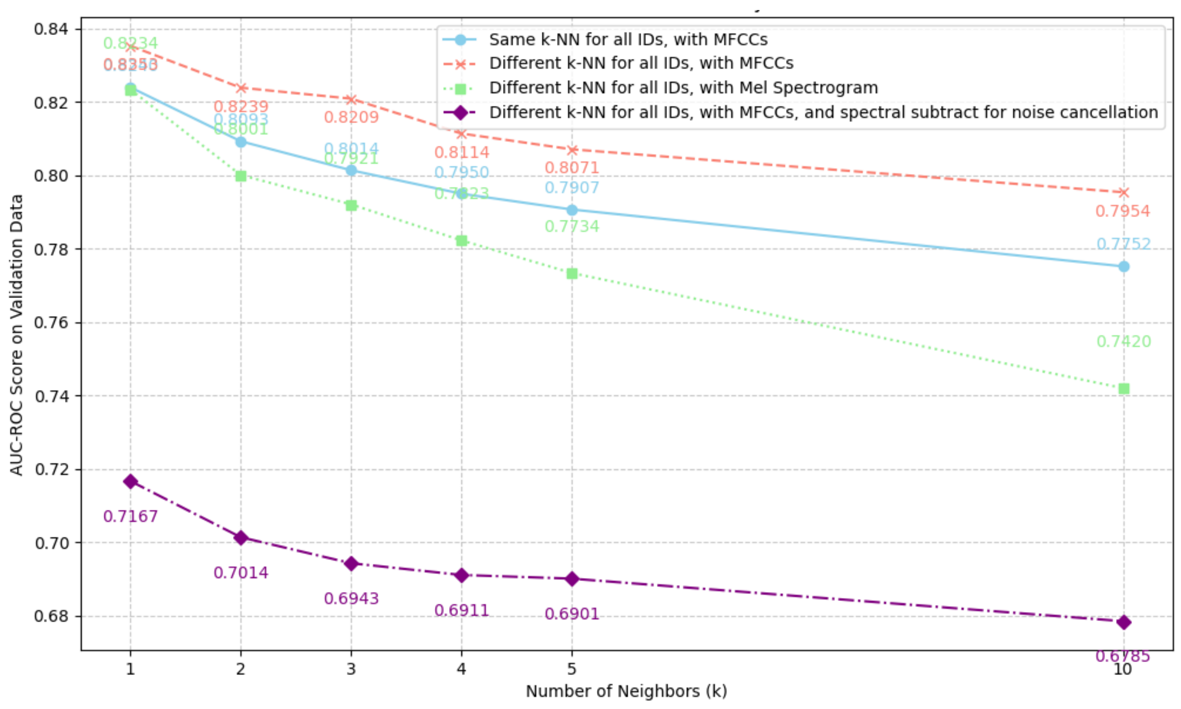

Figure 1: Comparison of different k-NN

2.1 Mel frequency cepstral coefficients (MFCCs) and Mel spectrogram

We need to turn the audio samples into vectors so that they can be used in our machine learning models. We tested two pre-processing techniques, Mel frequency cepstral coefficients (MFCCs) and Mel spectrogram. MFCCs are a set of features that compactly represent the **spectral envelope** of an audio signal on the perceptually motivated **Mel scale**, often used to characterize timbre in applications like speech and music analysis. A Mel spectrogram is a visual representation of an audio signal's frequency content over time, where the frequencies are mapped onto the **Mel scale**.

In order to test which of the two is more suited for our use case, we decided to test the performance of both feature extraction techniques by comparing their performance on a k-Nearest Neighbors algorithm (figure 1). MFCCs show better results than Mel Spectrograms on this comparison.

The parameters used for Mel Spectrograms are:

- **Sampling Rate:** The audio was resampled to 22050 Hz, balancing fidelity with computational efficiency.

- **Number of Mel Bands:** 128 Mel filter banks were used to capture frequency information on a perceptually relevant scale.

- **Maximum Frequency:** Frequencies were capped at 8000 Hz to focus on the most relevant audible range.

The parameters used for MFCCs are:

- **Number of Coefficients:** 13 coefficients were extracted, providing a compact representation of the spectral envelope.

- **Sampling Rate:** The audio was resampled to 22050 Hz, aligning with the rate used for Mel spectrograms to ensure consistency.

One shortcoming of our comparison is that we didn't compare the two techniques when using different hyperparameters. However, we think that MFCCs is better for our use case because of its robustness to noise, as there seems to be a lot of noise in the audio file we listened to. We are going to use MFCCs in the rest of the report.

2.2 Spectral Substract for noise cancellation

We tested if a noise cancellation technique would improve the performances of our models. We decided to test spectral subtraction for noise cancellation. This is a technique that estimates the noise spectrum

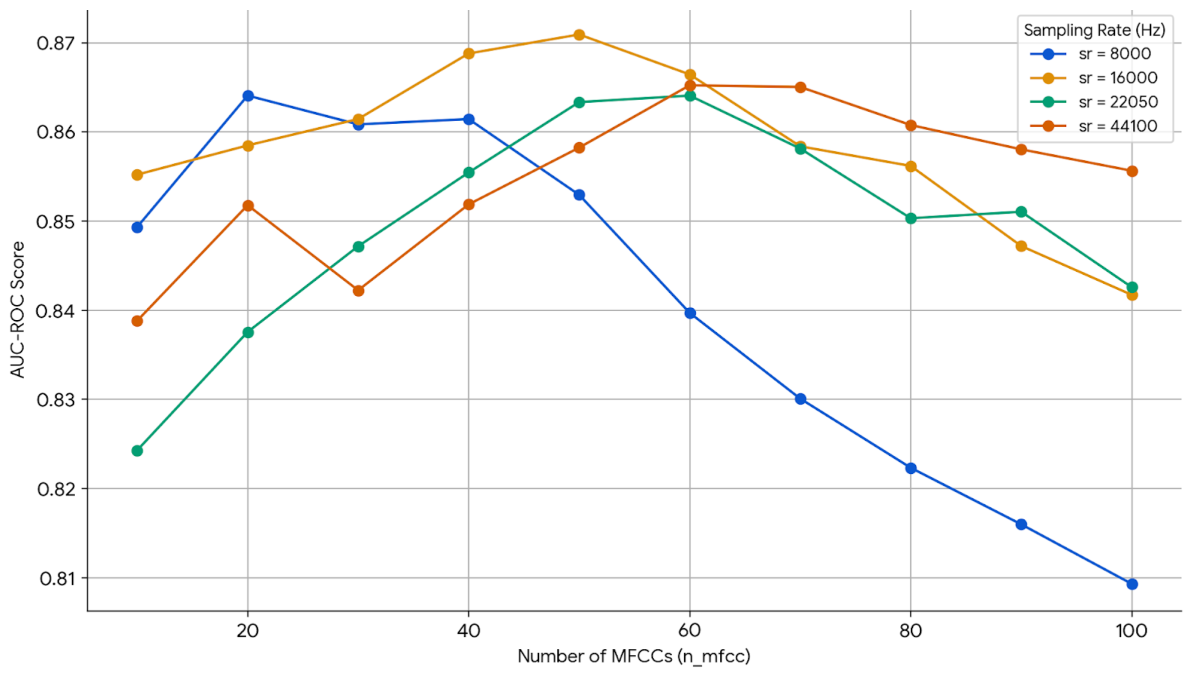

Figure 2: AUC-ROC score depending on the number of MFCCs

from silent or low-activity periods of an audio signal and then subtracts this estimated noise from the noisy signal's spectrum to enhance the desired sound. We used the following parameters:

- **Frame Length**: Defines the size of the window used for Fast Fourier Transform (FFT) analysis. A value of 2048 samples was used, influencing spectral resolution.

- **Hop Length**: Determines the stride between successive frames. A 512 sample hop length was chosen, impacting the overlap and temporal resolution of the analysis.

- **Over-subtraction Factor (α)**: A scaling factor applied to the estimated noise spectrum, allowing for more aggressive noise reduction. A value of 1.0 was utilized.

- **Spectral Floor (β)**: A minimum value applied to the noise-reduced spectrum, preventing musical noise artifacts. A spectral floor of 0.0 was set.

The results (figure 1) showed that the spectral substract method gives significantly worse results than without it. It is probable that this method either removes important data or introduces audio artifacts. As MFCCs are highly resistant to noise, we decided not to use other noise cancellation techniques.

2.3 Hyperparameter tuning for Mel frequency cepstral coefficients (MFCCs)

The parameters used for MFCCs are the number of coefficients and the sampling rate. We decided to test different combinations of those two parameters in order to find the one with the best performance. For that we used k-NNs, using k = 1, and iterated with different sampling rates, 8000 Hz, 16000 Hz, 22050 Hz and 44100 Hz, as well as different number of MFCCs, going from 10 to 100 with increments of 10. The results are in Figure 2.

The results show that the best parameter combination is a number of MFCCs of 50 with a sampling rate of 16000 Hz. Normally a number of MFCCs of 13 to 20 is more common, but in our case a number of MFCCs of 50 is better, this is probably due to the sound of the machines needing a more granular analysis. The sampling rate of 16000 Hz is something standard that stays within the audible range.

3 K-nearest neighbors (k-NN)

3.1 Number of neighbors

In our tests in Figure 1, we compared different models of k-NN, and the result we got was that the lower the number of neighbors, the better the performance. The optimal number of neighbors seems to be 1. The anomaly score for an unseen sample is determined solely by its distance to its single closest normal neighbor. Even a single normal reference point is sufficient to discern whether a new sample is an anomaly, as true anomalies are sufficiently far from any normal operating sound. Often having a number of neighbors of 1 can make the model sensitive to noise or outliers, but in our case there seem to be a clear distinction between normal and anomalous sound samples.

3.2 Using several k-NN

The dataset contains the sound samples of several different machines, each associated with an ID. We decided to test whether it was better to use one k-NN regardless of the ID, or to use a different model for each IDs. The results seen in Figure 1 show that using one k-NN per ID is more effective. While having one model for each ID might give more data for training, overall it diminishes the performance of the k-NN, and better results can be obtained with k-NN that are trained on a single machine. It is possible that if we had data for a lot more machines, it would be useful to try to build models that can generalise better to different machines instead of needing to be trained for each machine.

3.3 Final results

We took the best model for our validation data and evaluated it on our testing data and got the following results:

- **AUC ROC score**: 0.871

We get a very good AUC ROC score, which is the most important metric in our case, since all the other metrics can be changed by adjusting the threshold used to differentiate between normal and anomalous data. This high AUC ROC score means that our current results already give a very capable model, with only a simple k-NN.

Using the 99 percentile of distances in the training dataset as threshold, we end up with these metrics :

- **F1 score**: 0.9102
- **Precision**: 0.9091
- **Recall**: 0.9114
- **Accuracy**: 0.8692

4 Autoencoders

We decided to test if we could push those results further by using a more complex architecture. With autoencoders, we train a deep learning model to reconstruct a sound from normal data. In case of an anomaly, the reconstructed sound is not accurately reproduced, leading to a significantly higher reconstruction error compared to normal sounds.

We used the same data preparation as the one used for our best k-NN model, we also trained one model per ID.

To prevent overfitting, we use the regularization technique of early stopping. This monitors the model's performance on the validation data loss for each epoch, and if it stops improving for a certain number of epochs, the training is halted and the weights with the best results are restored.

4.1 Number of layers, and size of the bottleneck layer

We tested several configurations for our autoencoders, with different values of n, and different size for the bottleneck layer. In our figure, each model has:

- **1 input layer**: This layer receives the raw data features for the autoencoder to process. It has a size of 50 since this is the number of MFFCs that we are using. This also means that the size of the bottleneck layer can't be at 50 or above.

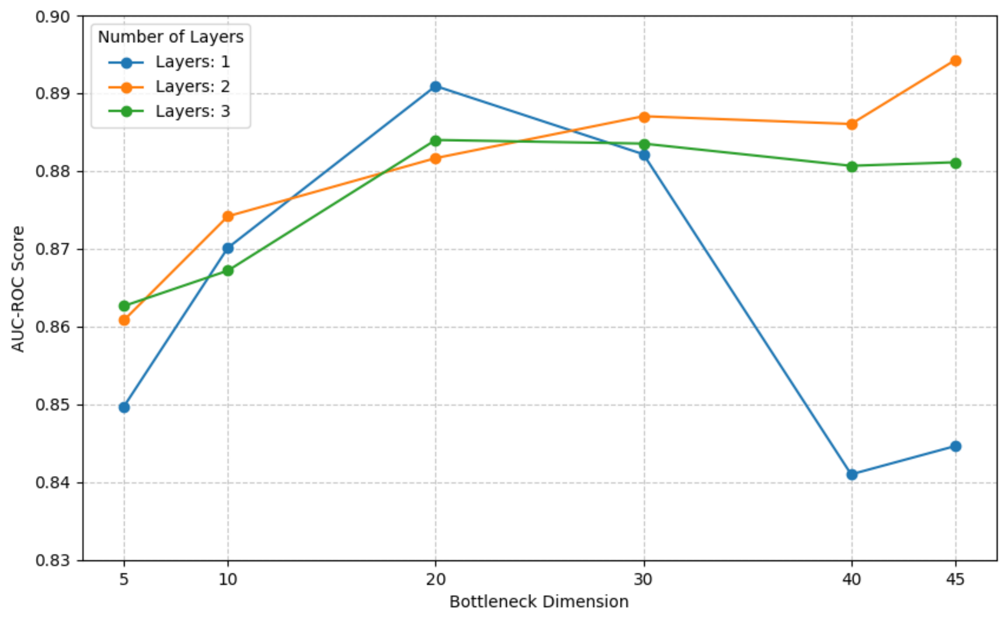

Figure 3: AUC ROC score depending on n and the bottleneck dimension for autoencoders

- n encoding layers: These layers progressively reduce the data's dimensionality, extracting key features and patterns.

- 1 bottleneck layer: This central layer represents the most compressed and abstract encoding of the input data.

- n decoding layers: These layers symmetrically reconstruct the data, expanding from the bottleneck back to the original dimensionality.

- 1 output layer: This layer produces the final reconstructed data, in order to closely match the original input.

When we change the number of layers and the size of the bottleneck layers, our code automatically adjusts the size of all the encoding and decoding layers so that the difference between the layers is gradual. We have the results in Figure 3, which shows the best results come from a n of 2 and a bottleneck of 45, although it seems like an outlier. In general, the best bottleneck dimension seems to be around 20 and 30.

4.2 Results

The best autoencoder we found has the following shape:

Table 1: Autoencoder Model Summary

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	50	0
encoder_dense_1 (Dense)	48	2.448
encoder_dense_2 (Dense)	47	2.303
bottleneck_dense (Dense)	45	2.160
decoder_dense_1 (Dense)	47	2.162
decoder_dense_2 (Dense)	48	2.304
output_layer (Dense)	50	2.450

It has an AUC ROC score of 0.88 on testing data, which is slightly better than the result we got with the k-NN architecture. Overall this is a very good model.

5 Conclusion

We managed to obtain good results for two different architectures, the simpler k-NN architecture and the autoencoder architecture. While we got slightly better results on the autoencoder architectures, the k-NN are faster and easier to train, and for inference.

There are still a lot of parameters that we didn't have time to try to optimize, such as the batch size for the autoencoders, or trying to add more layers to the autoencoders. We could also have tried other architectures, such as robust random cut forest or isolation forest.