Master's thesis

2022

Simen Berg-Hansen

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Simen Berg-Hansen

An exploration of acoustic gas leak and anomaly detections suitability for autonomous inspections in industrial plants

June 2022

NTNU
Norwegian University of
Science and Technology

An exploration of acoustic gas leak and anomaly detections suitability for autonomous inspections in industrial plants

Simen Berg-Hansen

Engineering and ICT
Submission date: June 2022
Supervisor: Gunleiv Skofteland
Co-supervisor: Christian Holden

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Abstract

Today, many faults in industrial plants are found by skilled workers doing manual inspections and using hearing to detect damaged machines or leaks. Equinor has begun the transition to autonomous robot inspections instead and wishes to examine acoustic options. This thesis explores two directions: to specialize the robot to detect gas leaks, since it has the highest damage potential, or to recognize sound anomalies, which are sounds that deviate from the normal. A compressed air leak dataset created by Johnson *et al.* [1] is used, and a handful of machine learning models are chosen for each approach. The tree-based family is selected for gas leak detection and One-Class Classifiers for anomaly detection. Through several experiments, both approaches showed promising results, with the gas leak detector indicating it might be possible to recognize leaks regardless of background noise or leak type. The anomaly detector classified 90% of anomalies correctly with the least noisy data and 70% on the hardest. It is also exhibited that looking at a signal's average power between 20-24 kHz is effective in classifying leaks in the auditory hearing range in this dataset. In the end, it is concluded that the gas leak approach is the most promising and should be the main focus of any succeeding work.

Sammendrag

I dag blir mange feil i industrianlegg funnet av kvalifiserte arbeidere som utfører manuelle inspeksjoner og bruker hørselen for å oppdage skadede maskiner eller lekkasjer. Equinor har begynt overgangen til inspeksjoner utført av autonome roboter i stedet og ønsker å utforske akustiske muligheter. I den sammenheng undersøker denne oppgaven to mulige retninger: å spesialisere roboten til å oppdage gasslekkasjer, som har det høyeste skadepotensialet, eller å gjenkjenne lydanomalier, som er lyder som avviker fra det normale. Et datasett med komprimert luft laget av Johnson *et al.* [1] brukes, og en håndfull maskinlæringsmodeller er valgt for hver tilnærming. Modeller fra beslutningstre familien brukes for gasslekkasjedeteksjon og *One-Class Classifiers* for anomalideteksjon. Gjennom flere eksperimenter viste begge tilnærmingene lovende resultater, og gasslekkasjedetektoren peker mot at det kan være mulig å gjenkjenne lekkasjer uavhengig av bakgrunnsstøy eller lekkasjetype. Anomalidetektoren klassifiserte 90 % av uregelmessighetene riktig i møte med data med lite støy og 70 % ved høy. Det er også vist at å se på et signals gjennomsnittlige effekt mellom 20-24 kHz er konstruktivt for å klassifisere lekkasjer i det brukte datasettet. Til slutt konkluderes det med at gasslekkasjetilnærmingen er den mest lovende og bør være hovedfokus for påfølgende arbeid.

Preface

This master thesis is written as part of a five-year M.Sc. program named Engineering and ICT at the Norwegian University of Science and Technology. The thesis is conducted within the specialization in Robotics and Automation, at the department of Mechanical and Industrial Engineering (MTP) during 20 weeks in the spring semester of 2022. The thesis is a prolongation of the work done in a specialization project from the autumn semester of 2021. I would like to thank my supervisors Gunleiv Skofteland and Christian Holden for advising me through this process.

Contents

Chapter 1

Introduction

1.1 Problem Description

Early detection of problems in machines or equipment can be essential to avoid further destruction. For companies operating in the oil and gas sector, a leak can have significant repercussions environmentally, economically, and safety-wise. The worst leak outcome is an explosion caused by a spark or flame reacting with the gas. Current leak detection approaches include mounted gas leak detectors and manual inspections by skilled workers' that use hearing to discover and localize leaks. The gas leak detectors are not always suited for outdoor use as wind heavily impacts them. Having other essential tasks to tend to, struggling to access small or high places physically, and the human hearing range, which is around 20-20kHz, are factors that limit the workers. Equinor has proposed a solution to this problem by equipping an autonomous inspection robot, such as the one displayed in Figure 1.1, with microphones that can conduct routine inspections on on-and offshore oil and gas platforms. The robot should process the recorded audio and notify a human if any alarming sounds are detected. This use of autonomous robots is part of Equinor's vision of having fully unmanned platforms. However, the technology is not limited to the oil and gas sector, and most other industrial plants and manufacturing facilities could benefit from this technology. In this thesis, we consider two different angles to the problem; one is to specialize the robot in recognizing the sound of a gas leak and alerting humans when it believes the sound is present. The other is to instead focus on alerting humans when the robot observes some sounds in a location that differs from the expected, often called an *anomaly*. Not too dissimilar from how mechanics can detect a broken engine by comparing the sound to how they know it should sound.

In recent years, researchers have achieved excellent results in speech and audio classification tasks using machine learning, but the work is not as advanced for industrial audio analysis. However, there is a growing interest in the field, and the same methods might also prove successful here.

The overarching goal of this thesis is to explore which of the two proposed directions above, which will be referred to as leak detection and anomaly detection, are most suitable for the robot. This goal is divided further into the following sub-goals:

- Find a publicly available relevant dataset with gas leaks and industrial noise.
- Explore the data to learn about the characteristics of gas leaks.
- Use the knowledge about the data to extract relevant information from the audio, called *features*, and explore which features are most impactful.
- Create and train suitable machine learning models for gas leak and anomaly detection.
- Utilize the dataset to test the models on experiments as relevant as possible to how the robot would be used in an industrial plant with the data available.

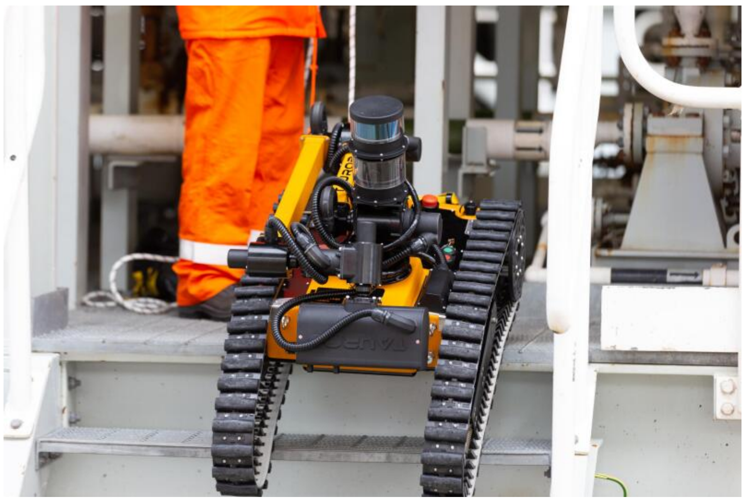


Figure 1.1: An autonomous inspection robot created by Equinor which has been deployed to an offshore platform. Source: [2]

In Section 1.2, there is a brief explanation of how the robot is intended to be used, as well as how this, and other factors, create limitations for the thesis.

1.2 Robot Behaviour and Limitations

This thesis assumes that the robot will approach several pre-decided locations a few times a day and stop for a predetermined time to record the audio. We assume the sound is stationary during this recording, meaning the sound image does not change. However, the sound in one location can change from day to day. Audio will be recorded with a non-contact microphone capturing airborne sound. The robot always knows where it is down to the millimeter, thanks to a GPS, but the steering is not precise enough to do the same, so we cannot presume recordings from precisely the same position each time. The robot can also store specific information and connect that to a GPS location which can be used later. In addition, the scope and achievements in this thesis are limited by the available dataset's quality and relevance.

1.3 Related Work

Using mobile robots for acoustic leak inspections is not common in the literature, yet, Schenck *et al.* [3] proposed combining ultrasonic microphones and LIDAR on a vehicle to localize pressurized leaks. Kroll and Gunther [4] created an ultrasonic scanning system mounted on a mobile service robot for ground detection and a remotely controlled micro aerial vehicle where it did not reach.

Using contact sensors, such as acoustic emission sensors and accelerometers, is most prominent in related literature since it allows for more frequent recordings, and it can be less noise.

When it comes to anomaly detection, Nunes [5] has completed a systematic review of the papers published about detecting sound anomalies with machine learning and found out that Autoencoders (AE) and Convolutional Neural Networks (CNN) are the techniques most cited currently. Oh and Yun [6] and Duman and Bayram [7] use AE that inputs images representing the change in frequency over time, called spectograms. The AE returns how similar new observations are from examples of typical sounds it has seen before. A similarity threshold is set to decide if the new data is dissimilar enough to be viewed as anomalous. Kampelopoulos *et al.* [8] tests several one-class classification algorithms, such as One-class Support Vector Machine (OC-SVM), Isolation Forest (IF), and Local Outlier Factor (LOF), which uses examples of normal data and aims to create a boundary around safe data and new data that falls outside of the edge is regarded as an anomaly. Müller *et al.* [9] begin with a CNN model that is trained on millions of regular images and uses transfer learning to train the model further on spectograms before also using One-Class Classifiers and autoencoders to detect anomalies.

For gas leak detection, Quy and Kim [10] classify new data as normal or gas leak based on the most similar data points with a K-Nearest Neighbour algorithm. Xiao *et al.* [11] separate leaks from non-leaks with an SVM classifier, creating a hyperplane that separates the classes. Cruz *et al.* [12] test several algorithms such as K-nearest neighbors, RF, AdaBoost, and XgBoost for gas leak detection and found Random Forest to perform best.

All the previously mentioned works use contact sensors, but research on airborne sound recorded by non-contact microphones is limited, probably because of few available datasets. Johnson *et al.* [1] discuss this issue and note that there are no large audio datasets with air leakage sound included, at least publicly available. They contribute to the field by creating a new dataset consisting of several leak types and background noises. They use a Convolution Neural Network (CNN) on spectogram images from their data to determine a baseline. Ning *et al.* [13] use a CNN as well on data unavailable to the public. Nevertheless, they execute an enhancement procedure to the spectograms to emphasize the leak in the images before feeding them to the network. The same authors published [14] the year later, where they used a Random Forest algorithm on several acoustic features and achieved similar results.

Henze *et al.* [15] has created a pipeline combining both leak and anomaly detection. Autoencoders are used to detect anomalies, initializing a classifier to diagnose the cause if triggered.

1.4 Report Structure

In Chapter 2 the essential background theory is presented. Chapter 3 describes the methods used and ends with an explanation of several experiments conducted. The results of these are presented in Chapter 4 before they are discussed in Chapter 5. Any weaknesses are also acknowledged, and the main approaches are compared. Chapter 6 concludes the thesis and presents further works.

Chapter 2

Theory

This chapter aims to equip readers with the preliminaries to follow this thesis. It begins with a general introduction to sound and its digital representation, followed by a brief overview of the Fourier transform and why it is useful in signal processing. Then what *audio features* are, and what the features used in this thesis represent. Finally, Machine Learning (ML) and relevant concepts in that field are presented before two categories of ML methods are introduced.

2.1 Signal Processing

2.1.1 Audio Signals

Rossing and Moore [16] define sound as longitudinal waves that propagate through either gas, liquid, or a solid and cause slight changes in air pressure that ears or microphones can detect. The frequency of that sound is determined by how fast these waves oscillate. According to Lerch [17], an *audio signal* is a function of these sound pressure levels over time. The sound emitted from gas pipeline leaks is from the turbulence caused by the gas decompressing when it enters an environment with lower pressure [1]. To humans, it appears as a hissing noise, and the intensity is dependent on the distance from the source, pressure in the pipes, gas viscosity and more. The leak sound can also reach the ultrasonic frequencies, making it inaudible to humans who can only hear frequencies between 20-20kHz. According to [18], the frequencies generated by gas leaks are generally between 10 and 60 kHz.

However, machines are not limited to the same frequency range as humans, and by using sensors like microphones, these higher frequencies can be observed if the equipment is made for it. Microphones convert mechanical energy in sound waves into electrical energy, but the signal has to undergo sampling for it to be used by computers. Sampling converts a continuous-time signal to a discrete-time signal by gathering a sequence of *samples* from the original signal. A value at a point in the signal is called a sample. The sample rate is the number of samples taken in a second, and the unit is samples per second or hertz (Hz) [19]. According to Nyquist Sampling Theorem [20], the sample rate should be at least twice the signal's highest frequency to reconstruct the original signal from the samples without any loss of information:

$$\text{sample rate} > 2 \cdot freq_{max} \tag{2.1}$$

Figure 2.1 shows a plot of a signal sampled with a rate not following Nyquist's theorem. The distortion of the signal caused by this is called *aliasing*.

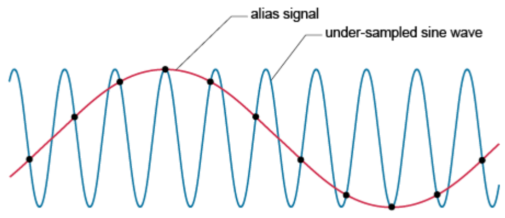


Figure 2.1: Under-sampled sine wave. Source: [21]

Therefore, to use signals with frequencies up to the mentioned 60 kHz that gas leaks could reach, one needs a microphone that can record those high frequencies and sample the signal with a sample rate twice that. Figure 2.2 shows a plot of a waveform of an audio signal for a trumpet.

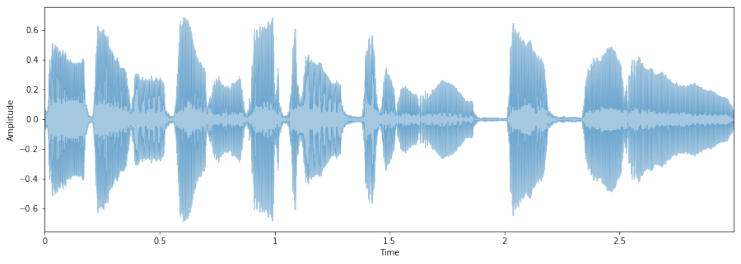


Figure 2.2: Waveform plot of a 3s trumpet sound. For sound, a waveform is a graph of the variations in air pressure over time.

Instead of analyzing the entire audio file length at once, it can be more beneficial to look at shorter segments, or *frames*. A process called *framing* is used for this and can be visualized Figure 2.3. By sliding a window with a specific length, called the frame-length, across the signal, the samples inside the window at each step make up a frame. The hop-length describes how far the window moves for the next frame, either in seconds or samples. We can convert between samples and seconds by dividing or multiplying the sample rate. Commonly, frames overlap in order to extract as much information as possible. Each frame must contain the frame-length amount of samples, implying that the final frame will be the last part of the signal with a full window.

The primary motivation for framing is that real-world audio signals are often non-stationary, meaning there can often be rapid property variations over time. One sudden short burst of energy can dominate the signal during analysis, so breaking it down into frames makes it possible to avoid this. In other cases, it can turn one large datapoint into several smaller providing more data for analysis [22].

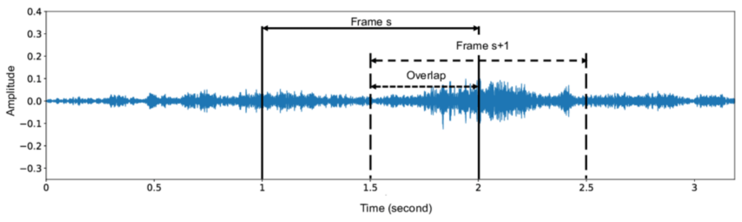


Figure 2.3: Framing applied to a signal using 1 second frame-length, and 0.5 seconds hop-length. Source: [23]

2.1.2 Fourier Transform

An audio signal is composed of several single-frequency signals superimposed. The Fourier transform is a mathematical concept that can be used to separate the signal into the individual frequencies again. This transform can give information about which frequencies are in the signal and their magnitude and phase. Transforming our data from the time domain to the frequency domain gives another way of representing it that can benefit the analysis. Figure 2.4 illustrates the relationship between the time- and frequency domain for a signal.

The *Discrete Fourier Transform* (DFT) is a version of the transform that allows discrete data, which is required for a computer to execute it. In practical implementations of the DFT, it is common to use the *Fast Fourier Transform* (FFT) algorithm that computes the DFT efficiently. Several python libraries exist with

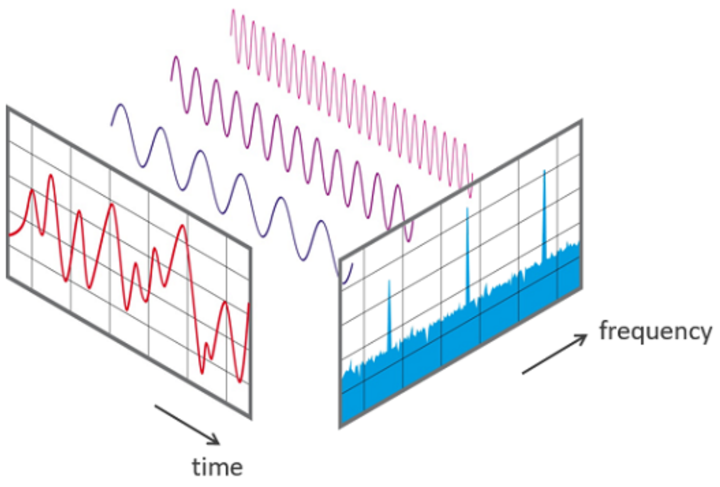


Figure 2.4: Figure illustrates the relationship between a signal's time- and frequency domain. The signal in the time domain (red line) consists of three sine waves (purple, single lines). In the frequency domain (blue lines), we see peaks at the frequencies of the three sine waves from the time domain and minor peaks along the x-axis caused by noise. Source: [24]

slightly different implementations of the FFT, such as Scipy[1]. Using the methods from the Scipy FFT package on an audio signal, a frequency spectrum can be plotted. An example using the same sound from the waveplot above is found in Figure 2.5. A Spectrogram is a visual representation of how the frequencies of a signal change with time. Figure 2.6 shows an example, and it can be considered a frequency spectrum plot as in 2.5 at each time step.

[1] https://docs.scipy.org/doc/scipy/tutorial/fft.html

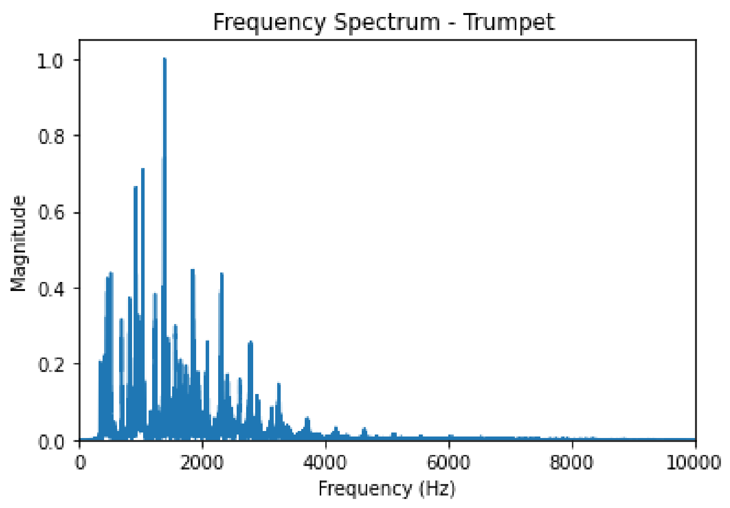


Figure 2.5: Plot of the frequency spectrum of a trumpet sound.

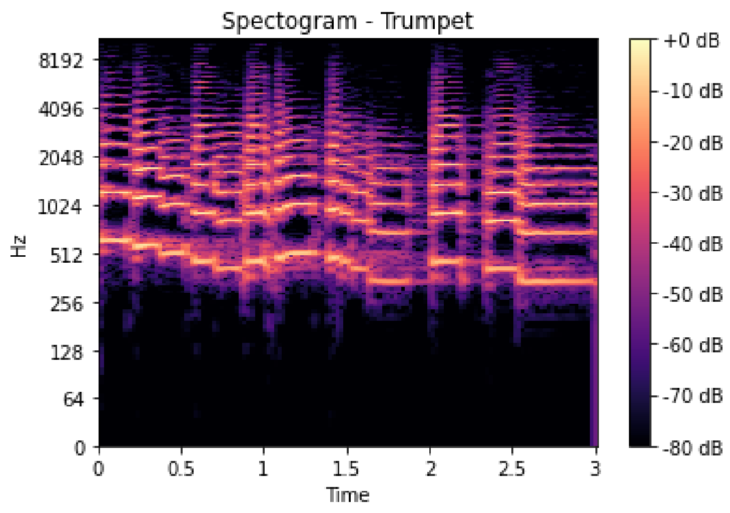


Figure 2.6: A plot of the spectogram for a trumpet sound, illustrating the change in frequency over time. Magnitude (dB) is represented by color; lighter colors indicate high values, and frequencies are log-scaled.

2.1.3 Signal Power and Energy

In physics, power is the amount of energy transferred or converted per unit of time. The same relation holds for audio signals as well.

The energy of a discrete signal is found by taking the sum of the squared absolute value of the signal's amplitude at each sample. For a signal x the energy $E(x)$ is defined as:

$$E(x) = \sum_{n=0}^{N-1} |x(n)|^2 \tag{2.2}$$

The average signal power is defined as the total energy $E(x)$ divided by the length of the signal or the number of samples. Average power P(x) is:

$$P(x) = \frac{E(x)}{N} = \frac{1}{N} \sum_{n=0}^{N-1} |x(n)|^2 \tag{2.3}$$

Another common way of representing the average power of a signal is using *Root Mean Square* (RMS). The RMS is computed by simply taking the square root of average power [25].

$$RMS = \sqrt{P(x)} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} |x(n)|^2} \tag{2.4}$$

Power Spectral Density (PSD) represents the distribution of power in a signal's frequency components, and it builds on the Discrete Fourier Transform. For each frequency component, the mean square amplitude is computed and averaged over the number of samples [26].

2.2 Audio Features

Audio features can be considered a condensed representation of the most valuable information from an audio signal. The original signal contains much information and is ill-suited for analysis tasks. Getting the features from the signal is called feature extraction and requires a good understanding of the problem domain to know where the information is [22]. It is common to separate between time-domain and frequency- (spectral) domain features based on the signal representation. The ones that have been used in the project will now be quickly introduced:

2.2.1 Time- and Frequency Domain Features

- Spectral Contrast
  - This feature aims at capturing the relationship between the highest and lowest frequencies in a frame. Each frame is divided into N sub bands (default N = 6), and the top energy is compared to the lowest energy. The contrast in each sub band are returned for each frame.
- Spectral Centroid
  - This feature aims at capturing where the spectral center of "gravity" is for each frame. A spectrum for each frame is extracted and normalized, and the mean of the magnitudes are returned as the centroid.
- Spectral Bandwidth
  - This feature aims at capturing the variance from the spectral centroid and is computed by taking the weighted mean of the distances of the frequency bands from the centroid.
- Spectral Rolloff
  - This feature captures at what frequencies most of the power is located. It returns the frequency bin where at least a predetermined percent of the power in the spectrum is below it. The default is set to 85%.
- Root Mean Square Energy (RMSE)
  - This feature returns the root-mean-square value of each frame.
- Zero Crossing Rate (ZCR)
  - This feature aims at capturing the smoothness of a signal and returns the number of zero-crossings within a frame.

2.2.2 Mel-Frequency Cepstral Coefficients (MFCC)

MFCC is a way of representing the power of a sound that emulates how humans perceive sound. Humans are much better at distinguishing between small differences in the lower frequencies than the high. For example, between 100 and 200 Hz and 1000 and 1100 Hz. Frequencies can be transformed to the *mel scale* to capture this relation by using Equation (2.5). Since it mimics how humans perceive sound, it is often used in music and voice recognition tasks and might also prove useful here.

$$Mel(f) = 2595 * log10(1 + f/700) \tag{2.5}$$

To get the MFCC the following steps are commonly done [27]:

1. Take the Fourier Transform of the signal.
2. Convert the frequencies to the mel scale.
3. Take the logarithm of the powers of the mel frequencies.
4. Take the Fourier Transform of the mel log powers again.
5. Return the amplitudes from the spectrum.

Figure 2.7 shows the MFCC computed on the trumpet sound using librosa's MFCC method.

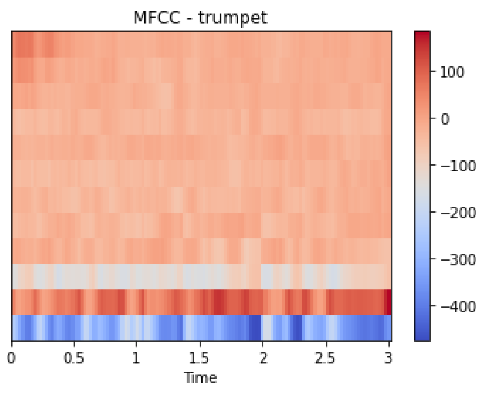


Figure 2.7: A plot of the MFCCs of a trumpet sound, using 12 rows of mel frequency bins and magnitude (dB) represented by color.

2.3 Machine Learning

2.3.1 What is Machine Learning?

This section is mainly inspired by Russell and Norvig [28] Zhang [29]. Machine Learning is a direction in Artificial Intelligence (AI), as seen in Figure 2.8. AI is, according to Rich [30] *the study of how to make computers do things at which, at the moment, people are better*. Machine learning uses sample data, called training data, to build mathematical models that can make predictions or decisions without explicitly being programmed for the task. A test set the models have not seen is used to evaluate their performance.

It is common to divide ML problems into supervised or unsupervised based on the information available during training. In supervised learning, the model is presented with example input-output pairs and learns a function that maps an input to an output. The desired output is often called the *target* value. If the output is a discrete value, the task is called *classification*, and *regression* if it is continuous. Classification is called either *binary*, if there are two classes to classify, or *multi class* if there are three or more. In unsupervised learning, the model has no information about the output data and learns patterns in the input data without feedback. *Clustering* is a common application where data with similar features are grouped. Figure 2.8 also shows that ML can be divided into three categories, including both classification and clustering, as well as reinforcement learning. This thesis does not use reinforcement learning and clustering algorithms; however, several different classification approaches are.

One-Class Classification (OCC) is a classification problem where the goal is to identity if objects belong to that one class in the training set or not, often called either *novelty* or *outlier* Detection. Both approaches aim to separate a group of regular observations from irregular observations, called outliers or anomalies. Suppose the data only contains data points of the regular observations. In that case, we call it novelty detection, but if there are some outliers in the training data, it is called outlier detection [31].

2.3.2 Training and Testing Set

In Section 2.3.1, it was stated that subsets of the entire dataset, called train- and test sets, are needed when working with machine learning models. The train set is used as examples for the model to learn from, while the test set's purpose is to evaluate how well the model performs on unseen data. The data can be anything

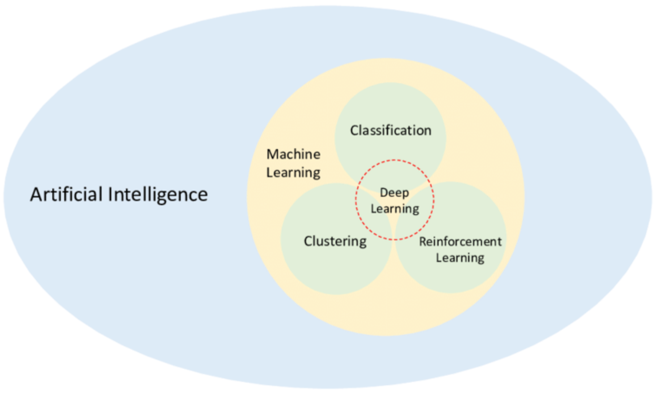


Figure 2.8: Relationship between artificial intelligence (AI), machine learning (ML) and deep learning (DL). Source: [32].

from a vector of numbers to images or audio files. Nevertheless, it is common to represent images with vectors of pixel intensity and audio files as a vector of sampled amplitudes. In this thesis, one data point is a vector with one value for each feature extracted from the signal. When splitting the data into train and test sets, a common approach is randomly selecting 80% for training and 20% for testing. However, this is not always the right solution, and according to Riley [33], *the question you want to answer should affect the way you split your data*.

2.3.3 Evaluation Metrics

Reliably evaluating the models will be necessary. The most common metric for testing the performance of a classifier is the accuracy, which is:

$$accuracy = \frac{\textit{number of correct predictions}}{\textit{total number of predictions}} \tag{2.6}$$

However, this does not take into account the classes and their distribution. Especially if one class dominates the others in the dataset, any model will get a misleadingly high accuracy by predicting everything as the dominant class. Precision and recall are commonly introduced to understand the models' actual performance better. They are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

where TP is true positive, FN is false negative, FP is false positive and TN is true negative. Figure 2.9 shows a confusion matrix which is used to illustrate these relations.

Figure 2.9: Confusion Matrix.

		Prediction class	
		P	**N**
Actual class	**P**	True Positive	False Negative
	N	False Positive	True Negative

In a leak detection context, we can think of *precision* as representing how many times we predicted something to be a leak, which actually was, and *recall* as how many of the actual leaks we detected. An ideal model will have high precision and recall; however, these are contradictory. For example, classifying everything as a leak would give a high recall but low precision. Therefore, there must be a trade-off and deciding what is most important for the use case. For higher-risk situations such as gas leak detection, it is much more essential that all leaks are caught, and some false alarms are therefore allowed - thus, a high recall is required. In other domains, such as email spam filtering, it would be more damaging to classify something as spam that was not than doing nothing, so precision is most important.

The f_1-score is a metric that represents the relationship between the precision and recall for predictions, and is computed with Equation (2.9).

$$f_1 = \frac{2 \cdot Precison \cdot Recall}{Precison + Recall} \tag{2.9}$$

The f_1-score will only be high if both precision and recall are high and low if one or both are low.

2.3.4 Overfitting

Overfitting is an issue for supervised machine learning models where a model can get a high score on train data and a low on test data. The models do not generalize to new data; it can appear like the models memorize all the train data. In the plot to the far right in Figure 2.10 we observe a model that has overfitted the training data, as can be seen by perfectly classifying all the data in the training set. The plot to the far left illustrates a model that underfits, meaning it learns little from the training data. The plot in the middle shows the optimal line separating the classes, ignoring the noise. It can be complicated to find the reason for the overfitting, but often it happens either because of small data sets or too complicated models [34].

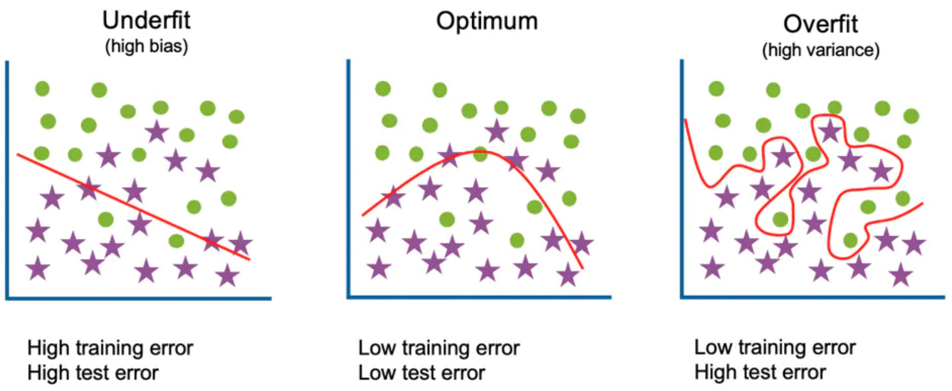


Figure 2.10: The figure illustrates three possible solutions to the binary classification task of separating the green circles from the purple stars with a line. The plot to the right shows overfitting. Source: [35]

2.3.5 Cross-validation

According to Lones [36], who discusses machine learning pitfalls in academic research many ML models are unstable. Small changes to the training data can result in significant variations in performance, which means that evaluating a model only once can give a wrong impression of a model's true performance. Cross-validation (CV) is a technique used to tackle this issue by training the models on different subsets of the training data. Each training iteration on a subset is often called a *fold*. Figure 2.11 illustrates a four-fold CV training scheme. After four folds of 25 samples, all the data is used for testing, and training. Some datasets require slight adaptations to the CV. If one class is small, Stratified CV ensures each class is adequately represented. By using Grouped CV, it is possible to make sure that data from the same group will not appear in two different folds, which can be useful to test the data on unseen groups while still using CV.

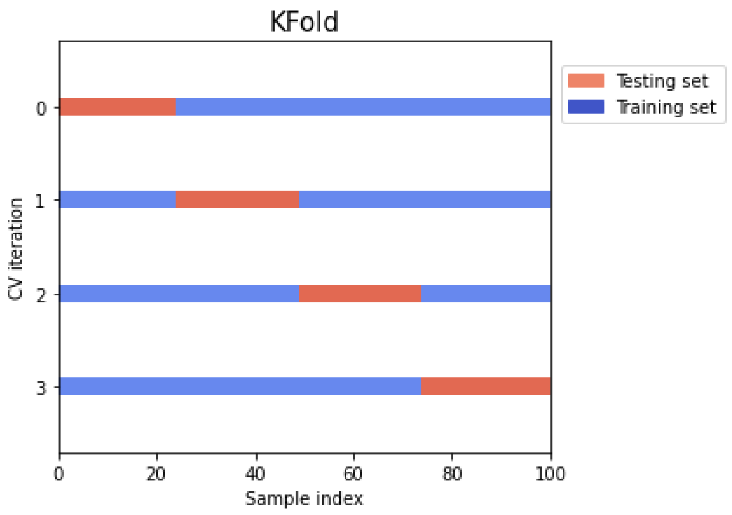


Figure 2.11: Figure showing cross-validation with four-folds. Each row is one fold, the red regions of the line are test data for that fold, and the blue is the train set.

2.4 Machine Learning Models

This section presents the theoretical background behind the machine learning models used in this project. First, the tree-based models' Decision Trees, Random Forest, AdaBoost, and Xgboost are presented in Section 2.4.1. Then the One-class detection methods One-Class SVM, Information Forest, and Local Outlier Factor is in Section 2.4.2.

2.4.1 Tree-based Models

This section is inspired by [37] and Russell and Norvig [28]. Tree-based models are a family of supervised machine learning recognizable by tree-like structures created for either classification or regression tasks. Decision Trees (DT) are the backbone of the family, and the tree structure is created during a training phase. Figure 2.12 illustrates one possible Decision Tree for the *Titanic survival prediction dataset* [2]. That is a typical beginner machine learning problem where the goal is to predict whether a person would survive the Titanic sinking based on age, gender, number of siblings onboard (sibsp), and more.

Every internal node works as a test on a feature. The internal nodes are selected based on their "importance," meaning how well they separate the passengers

[2] `https://www.kaggle.com/c/titanic`

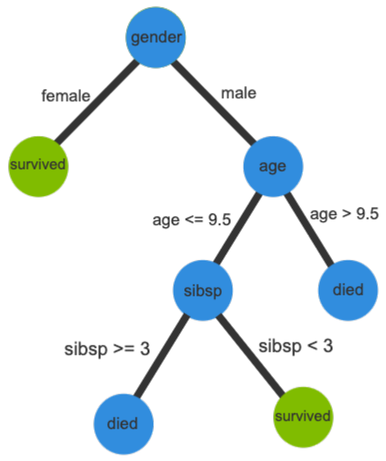


Figure 2.12: Figure showing a potential Decision Tree for the titanic survival classification problem based on [38]. The internal nodes are a feature used to split the data, such as gender, and each edge represents a feature's value, such as female or male. The leaf nodes represent the model's predicted value, either *died* or *survived*. A modified figure from: [39].

in the target classes *survived* and *died* based on only this attribute. The remaining attributes continue down the tree, and a leaf node is created when all the data belong to one class. New data can be classified by traversing the tree from the root down to a leaf, following the edges that match the values, and returning the leaf node value. Decision trees are popular in many domains due to achieving good accuracy with little data set and being easy for humans to interpret. On the other hand, they are known to return varying results and work best when train and test data are similar.

Ensemble learning is a method in which several models are combined to get more reliable and accurate predictions than only one model. Bagging is one technique where models are joined in parallel. The final prediction is based on the most common classification, called majority voting. *Random Forest* (RF) is an example of an ensemble learning method in the tree family. It works by combining multiple decision trees, all with some different partition of the dataset. Figure 2.13 illustrates a Random Forest model.

One advantage of the tree-based model family is that it can convey information about the *feature importance*, which means how essential each feature was for the model when making its decision. In Random Forest, one impurity-based approach looks at the total decrease in node impurity to rank the features. Node impurity is a measure of how homogeneous the labels at a node are.

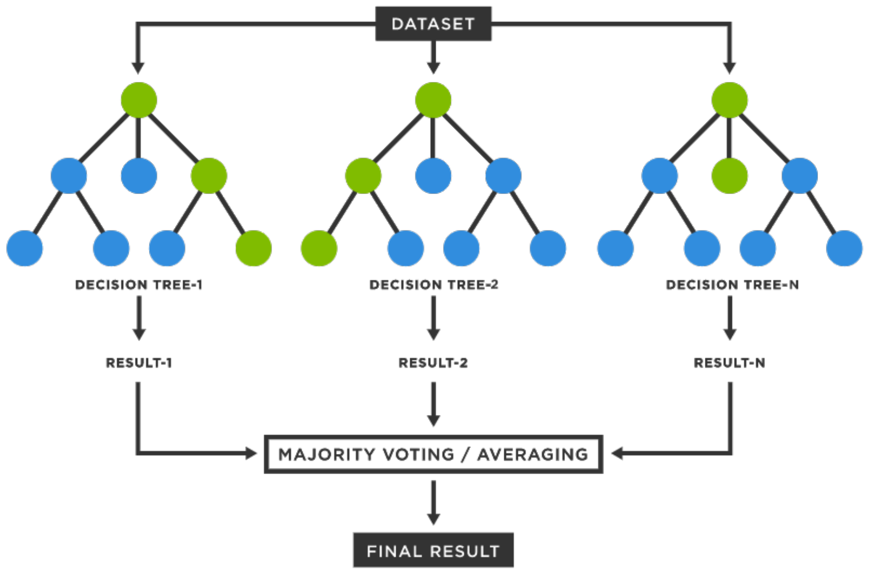


Figure 2.13: Figure illustrating the Random Forest algorithm where N different Decision Trees, such as in Figure 2.12, are used in parallel, and each tree returns one result, and the final prediction is the result with the majority of votes. Source: [39]

Boosting is another ensemble learning technique, where the models are added sequentially instead of in parallel, and the models learn from the errors of their predecessors. In contrast to in bagging, the training data for each subsequent classifier changes, and misclassified examples by earlier classifiers get assigned a higher weight, with the goal that the following models will prioritize them. AdaBoost (Adaptive Boosting) is a popular boosting algorithm [40]. XGBoost is another and stands for eXtreme Gradient Boosting.

2.4.2 Anomaly Detection Methods

One Class Classifiers (OCC) differ from binary- and multi-class classifiers because they are only trained using the “normal” class of data. The classifiers will create a boundary around the regular data, and the thought is that the new anomalous data will end up outside the border and be classified accordingly. OCCs can be used for both novelty and outlier detection.

OC-SVM is an adaptation of a popular binary classification algorithm called *Support Vector Machine* (SVM). The traditional SVM algorithm is a binary classification method that aims to separate two classes by finding the optimal hyperplane dividing the classes. The optimal hyperplane is the one that gives the most significant margin, meaning the distance from the hyperplane to the closest data point. These points are called support vectors and are the reason for the algorithm’s name. Figure 2.14 illustrates the optimal hyperplane for that data. New points are predicted to belong to a given class depending on which side of the hyperplane it falls. The SVM algorithm requires the data to be linearly separable, so it must be possible to divide the two classes with a line or a hyperplane. If the data is non-linearly separable, it can be transformed to a higher dimension where it is, using a kernel function, as seen in Figure 2.15. Different functions handle the transformation differently, but the most common are the linear, polynomial, and radial basis. The selection of a suitable kernel has a significant impact on the result of the models [41]. The SVM algorithm is adapted to handling only one class by trying to separate the training samples from the origin using a hyperplane instead of two classes [42].

Isolation Forest is an adaptation of Decision Trees. The algorithm assumes regular points are much harder to isolate than anomalous ones. An isolation tree is created by selecting a random value of a feature between the min and max of that feature partitioning the data until one point is isolated. This process is illustrated in figure Figure 2.16 and is continued until each point is separated. The figure

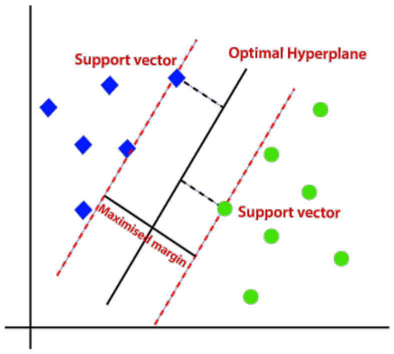


Figure 2.14: Figure showing optimal hyperplane dividing two classes. Source: [43]

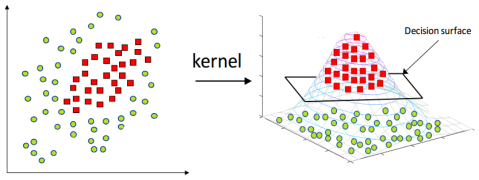
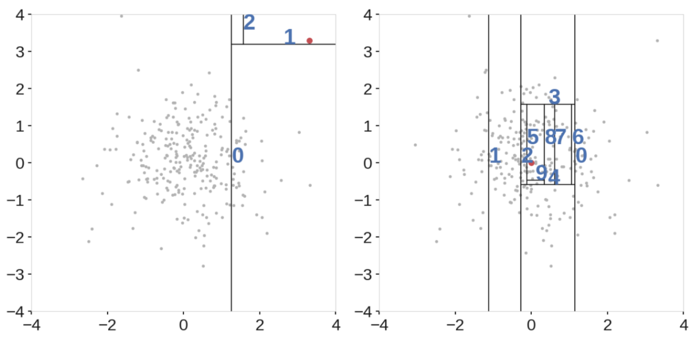


Figure 2.15: Figure showing linear separability after transforming the data. Source: [44]

shows that one point can be isolated by splitting twice, while the other ten, indicating that the first point potentially lays far away from the rest of the points and is an outlier.



(a) Anomaly point

(b) Nominal point

Figure 2.16: Figure showing the isolation of two points. Each line represent one split of a feature. Source: [45]

All the trees create a forest, as seen in Figure 2.17 where each partitioning, like in Figure 2.16, is represented as a tree. During the evaluation, a data point traverses through the forest, and based on how far down each three the point gets, it is assigned an *anomaly score*. The algorithm requires a parameter for the expected *contamination*, meaning the number of anomalies in the data. Based on the contamination value, a threshold is created. Datapoints with an anomaly score above this are classified as an anomaly [45].

Finally, the *Local Outlier Factor* (LOF) classifies anomalies by comparing the local density of each point to the density of the N closest points, called neigh-

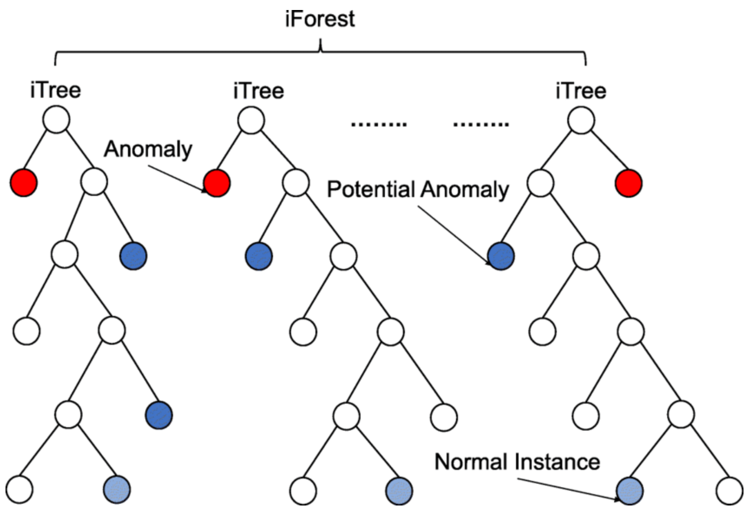


Figure 2.17: Figure showing a isolation forest. Source: [46]

bors. LOF uses the distance from the neighbors to estimate the density, and the algorithm assumes that a point is an outlier if it has a low density compared to its neighboring points [47].

Both OC-SVM and IF require a critical parameter for the amount of expected contamination in the training set. If this is sat low, it can create a too large boundary around the normal data making it less likely to pick up anomalies, and setting it high can cause a too small boundary resulting in many false alarms. This parameter should be selected by testing different values. LOF has a built-in method for calculating the contamination [8].

2.4.3 Hyperparameter Tuning

Machine learning models have a few parameters that can be tweaked depending on the problem for better performance. In a Random Forest model, examples of these parameters are how many decision trees the forest should include (n_estimators) and how deep one tree can grow (max_depth). These parameters are decided before training and cannot change during; therefore, they are called *hyperparameteres*. *Hyperparameter tuning* is a process to find the optimal hyperparameters for the model to solve the task. However, some models, such as Random Forest, are known to perform well with the default parameters [48], while others need tuning to do good on a task.

According to Probst *et al.* [48] *overfitting* is also a potential issue during tuning. The issue arises if the parameters are too perfect for the training data, making it generalize poorly to unseen data. Figure 2.18 illustrates the f1 score on training and test set for a Random Forest model for an increasing number of max_depth. Max_depth is the number of nodes in the longest path between the root node and a leaf node in a tree. It is clear that as the depth increase, the f1 score on the training data approaches 1, while the test score decline after reaching its peak at around 5. This indicates that for deeper trees, the model is allowed to learn everything about the training data, which results in bad performance on the test set. Probst *et al.* [48] note overfitting during tuning can be partially avoided by using a test-set to notice the model is overfitted or cross-validation.

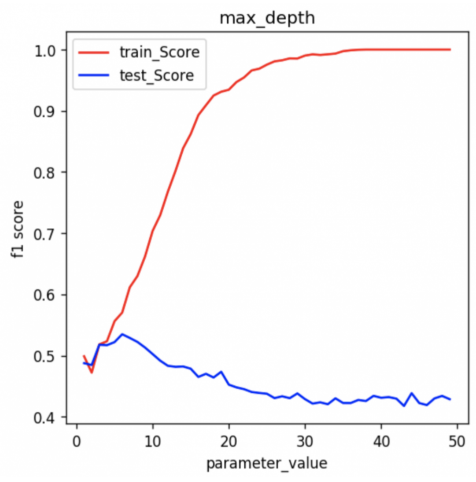


Figure 2.18: Figure showing the f1 score of a Random Forest model when increasing the max depth of the trees. The score on the train data increase, and the test data decrease, which indicates overfitting. Source: [49]

Two common approaches for executing the tuning are *grid search* and *random search*. Grid search is the most straightforward strategy, where a list of potential values is used, and each value is tested and evaluated. Random search randomly selects values between two limits to test and is, according to [50] more efficient than grid search in high-dimensional spaces.

Chapter 3

Methods

This chapter begins with a section introducing the selected dataset, followed by an exploratory data analysis (EDA) to understand the data better. Then, the preprocessing steps used on the data are explained, followed by which and how the features are extracted. The machine learning models that are used are then presented and, finally, the experiments conducted to test them.

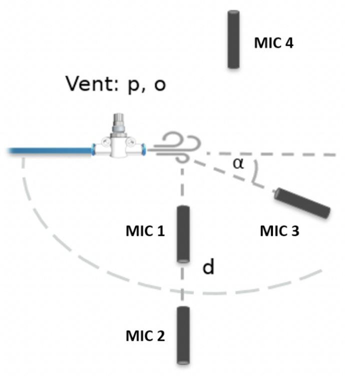
3.1 Dataset

For the project the *IDMT-ISA-COMPRESSED-AIR* [1] dataset from Johnson *et al.* [1] was selected. To our understanding, there are no comparable datasets publicly available using non-contact microphones of a considerable size that include gas leaks. The issue is mentioned in the paper associated with the dataset and is the motivation for its creation. We can use the dataset for leak and anomaly detection by operating with leaks as the anomalies.

The dataset contains two main leak types: *ventleak* and *tubeleak*. The ventleak is created in a lab by releasing compressed air from a choke vent controlled by a knob that opens and closes the vent. Each rotation of the knob opens the vent more, and the researchers found that when the knob is rotated 5.5-9 times, enough air leaks from the vent to be classified as a leak. Below 5.5 is no leak. All 0.5 increments of knob rotations from 0 to 9 are included in the dataset, meaning some leaks will be stronger than others. Introducing a damaged tube into the system gives the tubeleaks. The pressure of the air system is set to 6 bar, but to create *ventlow*, a lower version of ventleak, the pressure is set to 5 bar.

[1] https://www.idmt.fraunhofer.de/en/publications/datasets/isa-compressed-air.html

Four Earthworks M30 omnidirectional measurement microphones are oriented around the leak source at a configuration seen in Figure 3.1 and have a frequency range of 3 Hz to 30 kHz. However, a 48 kHz sample rate is used for discretizing the signals. To emulate noisy environments in the lab, the researchers played several noise recordings from a speaker. Those are hydraulic machine noise (high and low) named *hydr* and *hydr_low*, and general factory workshop noise (high and low) called *work* and *work_low*. The recordings without any noise played are labeled *lab*, and all the data was recorded in three different recordings.



(a)

Mic	Distance (d)	Angle (α)
1	20 cm	90°
2	2 m	90°
3	20 cm	30°
4	Full Room	-

(b)

Figure 3.1: Microphone positions in *IDMT-ISA-COMPRESSED-AIR* dataset. Source: [1]

Naturally, the dataset is imperfect since the creators intend to use it differently than intended in this thesis. They want to place microphones at critical locations in a compressed air network and detect when it leaks.

An ideal alternative would be to create a similar dataset using noise from one of Equinor's oil platforms, for instance. However, it would be time-consuming and hard logistically to get the same quality and amount of data. However, there is still a lot that can be learned from the data that can be transferable to the robot-use case if it is used smartly. Below are some of the issues with the data presented and potential workarounds to account for them.

- The dataset only contains two types of leaks released from a vent in a small pipe, and there is uncertainty about the resemblance of those leaks' characteristics and those from an actual leak.
- The leaks always come from the same position, which works well for monitoring a vent in a known location, but for the inspection robot, a leak can come from anywhere. It might be possible to utilize that there are used four

microphones to simulate different leak positions.

- The highest frequency that the microphones can record is 30 kHz, and a sampling frequency of 48 kHz is used, which limits it to 24 kHz. Therefore we are limited to slightly more than the human hearing range at 20 kHz, which does not take full advantage of the equipment.
- The noise is played from a speaker, which results in one source from the audio that might not be realistic, and some frequencies can be lost coming from a speaker. In addition, the noises used in the dataset are not very relatable to an oil or gas platform. The creators mention that other noises can be superimposed on the *lab* noise to get more noise.
- The dataset consists of an equal number of leaks and no-leaks, which does not represent the real world where a leak occurs less frequently than not. Dropping some of the files with leaks can mitigate this.

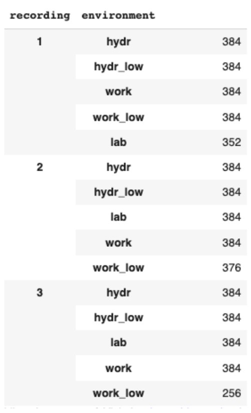
Finally, by using external datasets, there are no guarantees that the data is error-free, and one can only trust that the creators have constructed it as said.

3.2 Exploratory Data Analysis

The second step is to better understand the data by doing a simple Exploratory Data Analysis (EDA). EDA is a method that data scientists often use to analyze the data before using it, often by visualizations, to have a better foundation for making assumptions. It can also help identify erroneous or anomalous data or interesting patterns [35]. The analysis will focus on understanding the different impacts of the leaks, environments, and microphones on the sound in the files, which hopefully can be helpful in the detection later.

General Information

Firstly, the dataset contains a total of 5592 files. The dataset is supposed to contain 384 files for each recording session and environment; however, Figure 3.2 shows that some files are missing from *lab* in recording 1 and *work-low* in recording 2 and 3, caused by corrupted data according to Johnson *et al.* [1]. The average duration of an audio file in the dataset is 30.1 seconds, with the longest being 36.6 and the shortest being 26.2 seconds.



recording	environment	
1	hydr	384
	hydr_low	384
	work	384
	work_low	384
	lab	352
2	hydr	384
	hydr_low	384
	lab	384
	work	384
	work_low	376
3	hydr	384
	hydr_low	384
	lab	384
	work	384
	work_low	256

Figure 3.2: Table showing the number of files with each environment noise for the three recording sessions.

Leak Types

The dataset contains two types of leaks; ventleak (high and low) and tubeleak. To understand their differences, we look at the frequency contents in two files with the same environment noise, recording session, and microphone but with different leak types.

First, we plot the frequency spectrum of the two main leaks (vent- and tubeleak) in comparison to no leak with work and hydr as environmental noise. In Figures 3.3-3.6 this is plotted for the four combinations. In all plots, the magnitude (dB) difference between the leaks and no-leaks seems to increase as the frequency increases, even though the ventleak and tubeleak behave very differently. It can look like the tubeleak appears more like a broadband white noise, while the ventleak is more sporadic and higher in some specific frequencies. An auditory inspection also confirms this, where the ventleak is easy to pick out as a high-frequency noise, and the tubeleak is more indistinct. The figures also show that the biggest gap between leak and no-leak emerges at around 20 kHz in all cases. However, the gap is the biggest for the ventleak, where the leak causes a big increase in dB in these high frequencies. In the tubeleak case, the gap is smaller and is created by the no-leak dB decreasing, and the leak increases slightly.

Secondly, we compare the frequency spectrum of the two leaks to each other, now, not using dB. Frequencies below 5kHz have been removed due to them being much greater than the highest ones, which we expect to be most interesting based on the plots such as Figure 3.3. In Figure 3.7 the frequency spectrum for a ventleak with the standard and low version is plotted together with work as environmental noise. We observe that they appear very similar, with three distinct

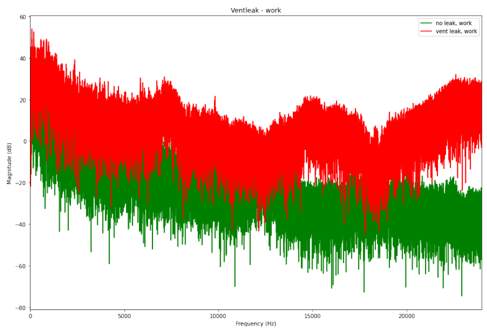


Figure 3.3: Frequency spectrum of *work* noise, with and without *ventleak* present (microphone 1)

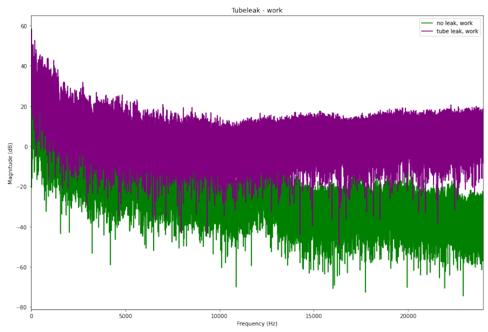


Figure 3.4: Frequency spectrum of *work* noise, with and without *tubeleak* present (microphone 1)

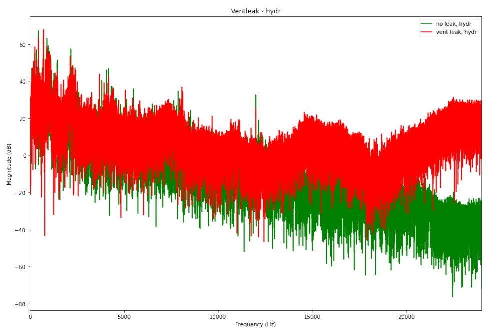


Figure 3.5: Frequency spectrum of *hydr* noise, with and without *ventleak* present (microphone 1)

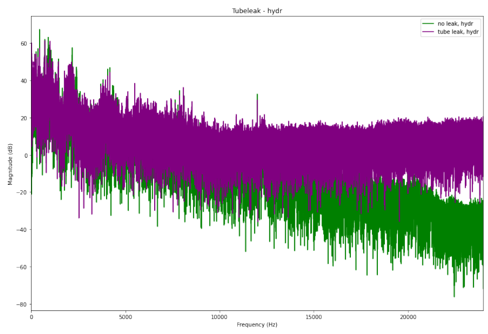


Figure 3.6: Frequency spectrum of *hydr* noise, with and without *tubeleak* present (microphone 1)

spikes around 7, 15, and 23 kHz, only separated by the ventlow having a slightly lower magnitude than the normal one. Figure 3.8 is the same plot but with a tubeleak instead. We observe that the highest amplitude, around 25kHz for both leak types, is double for the ventleaks compared to tubeleaks in this case. However, the most significant difference is that the three peaks are gone, and there appears to be a more gradual increase from 5-25kHz.

The information extracted from these plots tells us that it is possible to distinguish between leaks and no-leaks by looking at frequencies above 20kHz in this dataset and that detecting tubeleak might be a more challenging task than ventleaks since the gap appears to be smaller.

Microphone Position

The dataset contains recordings done by four microphones, all from different positions. Microphone 1 is placed perpendicular to the leak direction 20 cm away, and

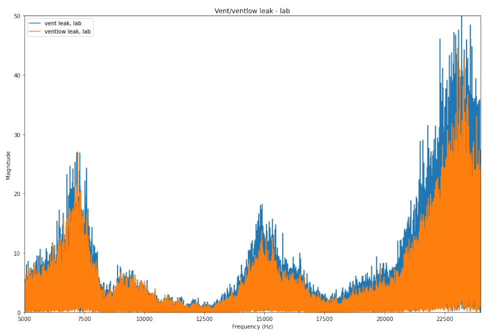


Figure 3.7: Frequency spectrum of lab noise, with *ventleak* and *ventlow* present (microphone 1).

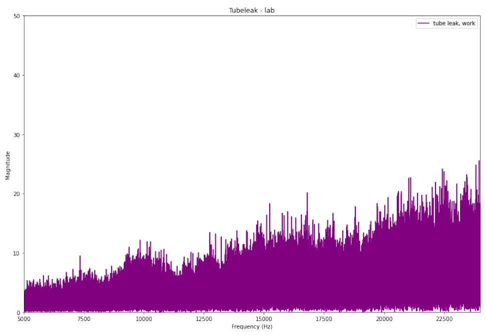


Figure 3.8: Frequency spectrum of lab noise, with *tubeleak* present (microphone 1).

microphone 3 is placed the same distance away but at a 30-degree angle. Microphone 2 is placed behind microphone 1 and is 2 meters away from the leak, while microphone 4 is positioned to record the full room. To better understand the impact these placements have, the frequency spectrums of four signals recorded by each microphone at the same time are plotted. In Figure 3.9 we look at a situation with work noise and ventleak. We only look at frequencies above 20 kHz since it appears to be the most relevant segment. Microphone 1 has the highest magnitude for all frequencies, but the difference increase with the frequencies. Microphone 3 is most similar, which was expected since they are placed at an equal distance, but surprisingly microphone 1 is at a more favorable angle. Microphones 2 and 4 give a similar spectrum but are significantly lower in magnitude than the others. In Figure 3.10 the recordings are from the same environment but of a tubeleak, and we now ignore frequencies below 5 kHz. The difference between the microphones is significantly smaller, particularly below 10 kHz. At the frequencies above this, the same pattern as seen in 3.9 returns. The fact that microphones further away struggle more at the highest frequencies is likely because the sound is faster absorbed, the higher the frequency [51], resulting in fewer waves reaching 2m than 2cm, for instance.

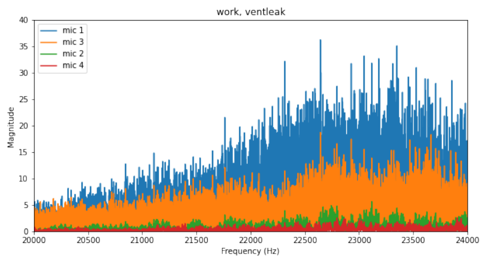


Figure 3.9: Frequency spectrum above 20 kHz of *work* noise, with *ventleak* present for all four microphones.

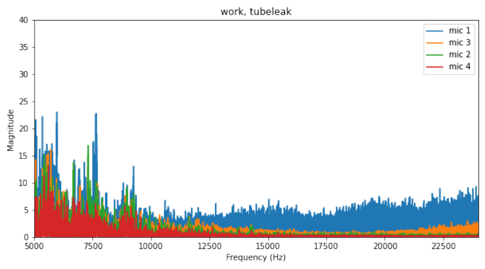


Figure 3.10: Frequency spectrum above 5 kHz of *work* noise, with *tubeleak* present for all four microphones.

3.3 Pre-processing

3.3.1 Loading data

The dataset from Section 3.1, was downloaded and stored in a Google Drive folder[2]. Code was written to handle the data more efficiently by extracting the information from the file paths and creating one table for all the files. This table can then be used to select only data with one particular leak, environment, or microphone.

The path to the files all follow this format:

$$/ventleak/hydr/1/1\_niO\_6.5n\_3l\_.wav$$

The code can be found in Appendix A.1. The *save_paths* method iterates over the paths of every file in the root folder and extracts the information. Doing this is possible since we know all the paths have the same form. Each piece of information is stored in a column, and all the columns are stored as an entry in a table that is saved to an CSV file. Table 3.1 presents a sample of the table.

To get the audio files in a format that can be worked with in python, the librosas.load(), Code listing 3.1, method is used. It returns a NumPy array of the samples and the sample rate used. The parameters are the path to the input file and the wanted sample rate (sr). Default sr is 22050, which will resample the signal with that sample rate, which would result in a sampled signal with $22050/2 = 11025Hz$ as the highest frequency due to Nyquist Sampling Theorem.

[2] `https://drive.google.com/drive/folders/1oc_4C5-dob3sHXk6yVxnxZ4JiaIx5HPx?usp=sharing`

Table 3.1: Tabular representation of information of a few files from the *IDMT-ISA-COMPRESSED-AIR* dataset.

path	leaktype	environment	recording	mic	knobrotations	leak
...wav	valve	lab	3	4l	1.0	0
...wav	vent	work	1	1l	7.5	1
...wav	vent	hydr-low	3	4l	9.0	1
...wav	vent	hydr	1	3m	3.5	0
...wav	vent	work	3	4l	0.0	0

Setting sr=*None* will preserve the native sampling rate of the file, which in this case is 48000.

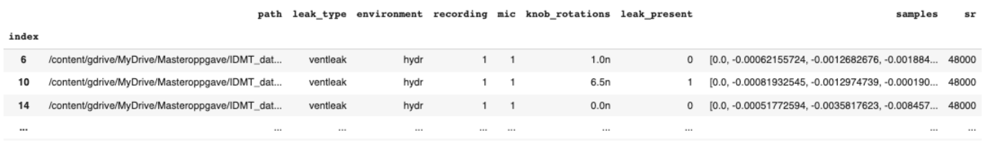
```
signal, sr = librosa.load(path, sr=None)
```

Code listing 3.1: Librosa's load method

Code has been written to select subsets of all the files which will be useful when conducting the experiments. The method takes in a table, like Table 3.1, of all the file information and a dictionary of which values wanted for given columns. An example of this dictionary can be seen in Code listing 3.2.

```
wanted_files = {
    'leak_type' : ["ventleak"],
    'environment' : ["work"],
    'mic' : [1]
}
```

Code listing 3.2: Dictionary of which subset of files to use



	path	leak_type	environment	recording	mic	knob_rotations	leak_present	samples	sr
index									
6	/content/gdrive/MyDrive/Masteroppgave/IDMT_dat...	ventleak	hydr	1	1	1.0n	0	[0.0, -0.00062155724, -0.0012682676, -0.001884...	48000
10	/content/gdrive/MyDrive/Masteroppgave/IDMT_dat...	ventleak	hydr	1	1	6.5n	1	[0.0, -0.00081932545, -0.0012974739, -0.000190...	48000
14	/content/gdrive/MyDrive/Masteroppgave/IDMT_dat...	ventleak	hydr	1	1	0.0n	0	[0.0, -0.00051772594, -0.0035817623, -0.008457...	48000
...	...	...	...	...	...	...	...	...	...

Figure 3.11: Table containing file information for each file, and a list of samples after the data is loaded.

3.3.2 Normalization and Denoising

After loading the files, it was considered to normalize them by dividing all by the global maximum amplitude in all the data. In the end, this was not done.

However, we standardize the features between 0 and 1 after they are extracted. Doing this is most important in clustering methods and models like SVM, where the euclidean distance is used. If one feature has values much higher than another, it will dominate the distance rather strongly. Even though this is not necessary for tree-based methods, it is done nonetheless since it will not impact the results negatively. However, it is essential that any normalization or standardization is based on the train set and that the test set uses the exact same scaling. Suppose the train and test features are scaled together. In that case, the test data will impact the train data resulting in the test data no longer being able to test the model's generalizability neutrally.

Doing some denoising or frequency-enhancing on the data was considered but rejected since the dataset's creators achieved good results without it. Also, Ning *et al.* [13] learned that it is challenging to remove noise in signals where gas leaks are included and wanted, but not impossible.

3.4 Feature Extraction

The next step is to extract information from the signals that can be useful for the machine learning models in the form of features. Kampelopoulos *et al.* [8], Quy and Kim [10], Xiao *et al.* [11], Ning *et al.* [14] and Rahimi *et al.* [52] were the main inspiration for all time-and frequency domain features considered in this thesis, since their work is similar in terms of goal and detection methods used. All features mentioned were researched, and based on the results in the papers and knowledge about the gas leak problem domain, most features were discarded. The second type of feature represents the power in segments of frequencies, either relative or not. It was based on the assumption that the total contribution to the energy in the higher frequencies would be more significant during a leak based on observations from the EDA in Section 3.2.

3.4.1 File Segmentation

The average length of one audio file in the dataset is 30 seconds. However, it does not mean we must analyze the entire file simultaneously. Splitting it into shorter segments of around 1, 5, or 10 seconds has been considered, and the selection is discussed in Section 3.4.5.

Regardless of the segment's length, two different approaches for handling the frames when extracting features from the signals were considered. One was to divide the signals into frames and take the average over all the frames for each feature so that we ended up with one data point for each file. The other was to get the frames in the same way but save them as unique data points instead of averaging. The advantage of added data points is why option two was selected.

3.4.2 Time- and Frequency Features

By looking at similar papers, very few report having good results using time-domain features, for leak detection at least. Therefore, the main focus is on the frequency features. The chosen features are spectral- centroid, bandwidth, flatness, contrast, zero-crossing rate, root mean square energy for the whole frame, and MFCC.

A method that takes in a signal, frame-length, and hop-length was written to make the features. Librosa's *feature* module was used to compute the wanted spectral features, and each returns an array with the computed value for each frame. Code listing 3.3 illustrates the method call for one of the features. The method returns an array of the feature for N frames, where the number of frames is decided by "n_fft" and "hop_length". Those parameters are the number of samples in one frame and how many samples the beginning of the frame is moved at each step. "Signal" is the audio time series, and "sr" is the sample rate.

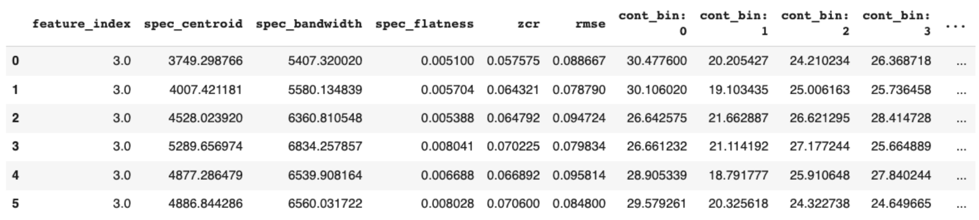
```
spec_centroid =
    librosa.feature.spectral_centroid(y=signal, sr=sr, n_fft, hop_length)
```

Code listing 3.3: Librosa's spectral centroid method

Spectral contrast and MFCC, however, return an array of the value of each frequency bin for every frame, so we get several values that are used as unique features for each frame. Each frame is stored as a row in a table, with the feature values as columns.

A column with the index of the file it originates from is appended to each row, ultimately resulting in tables like the one in Figure 3.12. We repeat this operation for each file, and all the tables created are combined to create the entire feature dataset.

In Figure 3.13, three selected features are plotted to observe if the features are able to separate instances of leaks and no leaks. The values are plotted for each frame for files from the same environment, leak, and microphone. In the spec-bandwidth plot (left), we observe a good separation between the two classes, with some outliers of frames with leaks with the no-leaks. The variance inside



	feature_index	spec_centroid	spec_bandwidth	spec_flatness	zcr	rmse	cont_bin: 0	cont_bin: 1	cont_bin: 2	cont_bin: 3	...
0	3.0	3749.298766	5407.320020	0.005100	0.057575	0.088667	30.477600	20.205427	24.210234	26.368718	...
1	3.0	4007.421181	5580.134839	0.005704	0.064321	0.078790	30.106020	19.103435	25.006163	25.736458	...
2	3.0	4528.023920	6360.810548	0.005388	0.064792	0.094724	26.642575	21.662887	26.621295	28.414728	...
3	3.0	5289.656974	6834.257857	0.008041	0.070225	0.079834	26.661232	21.114192	27.177244	25.664889	...
4	3.0	4877.286479	6539.908164	0.006688	0.066892	0.095814	28.905339	18.791777	25.910648	27.840244	...
5	3.0	4886.844286	6560.031722	0.008028	0.070600	0.084800	29.579261	20.325618	24.322738	24.649665	...

Figure 3.12: Table showing how features for frames are stored, using 5s long frames and 2.5s overlap.

the classes appears minor compared to the zero-crossing rate plot (center), where there is a greater spread for the frames with a leak. However, the variance is lower for the frames without a leak, and it is still possible to visually separate the classes. For the rmse (right) plot, there is no clear distinction between the classes because the energy in the lowest frequencies dominates the total energy, so the distinctive energy in the frequencies above 20 kHz gets lost.

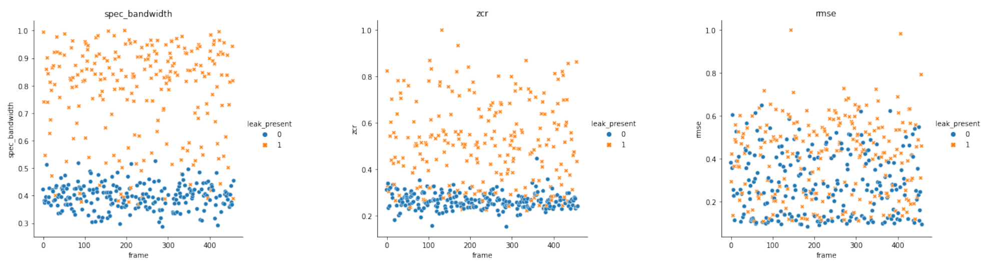


Figure 3.13: *Spectral bandwidth* (left), *zero crossing rate* (center) and *root mean square energy* (right) feature plots in *work* environment during a *tubeleak* recorded with microphone 1 for frames with 5 seconds length, for 50 files. The frames are shuffled. Orange points are from files with a leak, and blue is without.

In Figure 3.14 the Spectral Contrast in bin 5 for ventleak and tubeleak are plotted, with same conditions. Observe that in the case of a ventleak (left), this feature separates the classes well, while tubeleaks (right) are not as strong. This illustrates that a feature can be useful in classifying one type of leak and not another.

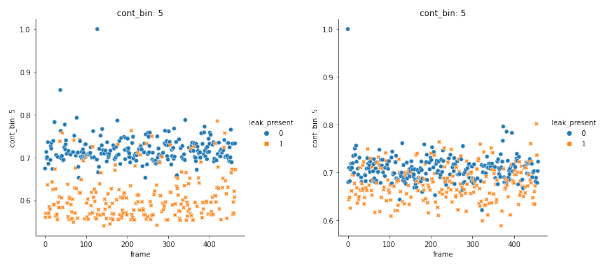


Figure 3.14: Plots of *Spectral Contrast* in bin 5 for data with the *work* environment but with different leak types. Otherwise same conditions as in Figure 3.13.

3.4.3 (Relative) Power in Frequency Segments

Outside of the standard features explored above, another feature was tested that captures the power, or the relative power, in specific frequency segments of a signal. This feature can potentially overcome the issues of taking the RMSE of the entire frame, as was seen in Figure 3.13. It is based on a hypothesis that the total power in a signal's highest frequencies will be greater during a leak than not. This feature has not been found in papers with similar goals to ours but is more common with EEG signal processing in neurological- and sleep research [53]. It is also possible to look at the relative power of a segment, meaning the total contribution that segments have to the total power.

The code to extract this feature is found in Appendix A.2, and was inspired by Vallat [54], a postdoc at UC Berkeley. First, Scipy's Welch method is used to estimate the Power Spectral Density (PSD). The PSD of an arbitrary signal is plotted in Figure 3.15 displaying only up to 5Hz for visibility. Scipy's Simpson method is then used to approximate the integral between the beginning and end of all the segments, see Figure 3.16. A list is used to describe where each segment starts and stops.

If the argument "relative" is set to *True* the relative power of each segment is returned, which is found by using the total power of the signal as a divisor on each segment. This results in a vector that can be plotted. In Figure 3.17 the relative power for frequency segments with 4000 kHz intervals has been plotted for two files recorded under similar conditions, with and without a leak. However, a difference between the two is more apparent in Figure 3.18 where the logarithms of the y-values have been plotted.

The next step is to get this feature in a format that makes it possible to merge with the same frame of features created above. The *util.frame()* method from librosa was used to divide the signal into frames in the same way as what is done

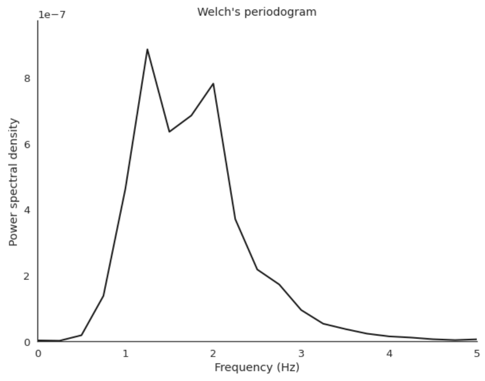


Figure 3.15: Welch's PSD estimation.

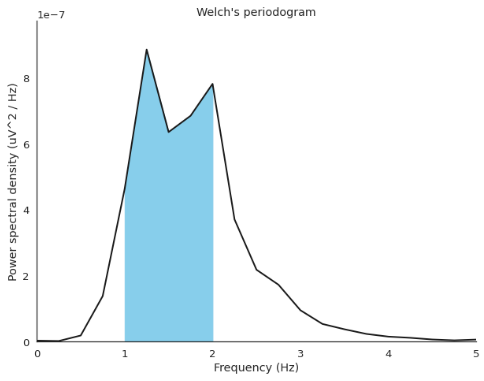


Figure 3.16: Integrating segments.

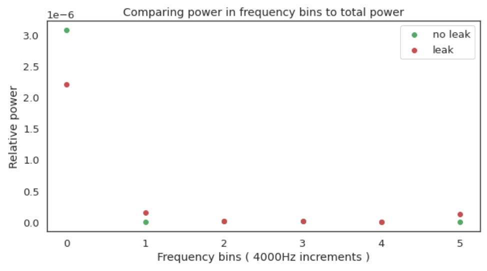


Figure 3.17: Plot of relative power with 4 kHz increments. *Work environment*, *ventleak* and microphone 1 is used.

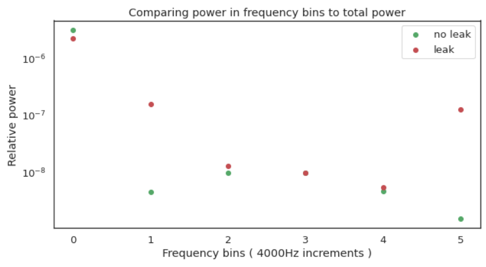


Figure 3.18: Plot of the logarithm of the relative power with 4 kHz increments. Same conditions as 3.17.

for the spectral feature computation in Section 3.4.2. The method created for extracting these features takes in the same arguments as time-frequency features to ensure that the frames are the same. A *segment list* is also a required argument that should say at what frequencies the segments start and stop. Each frame is passed to a method that returns a list of the relative power in each segment.

The segment list has to be handcrafted, and there is no perfect list which causes several questions to arise. Is few or many segments best? Should they be equal in length or maybe decrease as frequencies increase? The answers to these questions will vary depending on which problem the features will be used. However, it is possible to get a good alternative by using knowledge about gas leaks from the EDA and studying figures of relative power with different segment lists. This choice will be made in section Section 3.4.5.

In Figure 3.19 the log relative power of four combinations of leaks and environmental noise is plotted. It is a clear trend in all four subplots that the *leak* and *no leak* samples diverge at around the frequency bin 20, which means 20-21kHz. We can also see that the final separation between the classes is more significant during a ventleak than a tubeleak. Based on these results, it is evident that looking at the relative power of frequencies above 20kHz might prove most beneficial, which coincides with the discoveries made in Section 3.2.

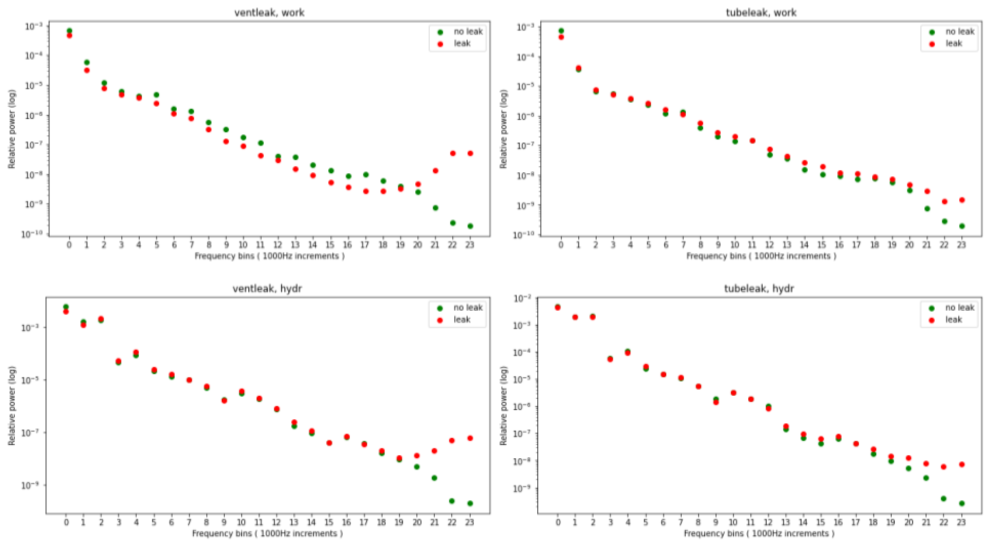


Figure 3.19: The log relative power with 1 kHz segments for *ventleaks* in left column and *tubeleaks* in the right. *Work* environmental noise is used in the top row, and *hydr* in the bottom. All recorded with microphone 1.

3.4.4 Feature Correlation

Figure 3.20 shows a *correlation matrix* for all the features presented in this chapter using microphone 1 data. The matrix is always square and symmetrical, where each cell *ij* is the correlation between the columns *i* and *j* of the features. Ideally, the features have a low correlation with each other since this implies that they might represent different information from the data.

From the figure, we observe a high correlation between the relative power features 10-20 and 20-24 kHz and the spectral- centroid, bandwidth and flatness, and zcr particularly. This correlation can mean that the mentioned spectral features also capture the information related to power in the top frequencies. Besides that, all the cont_bin's beside 0 have a relatively high correlation, but none with the features discussed above.

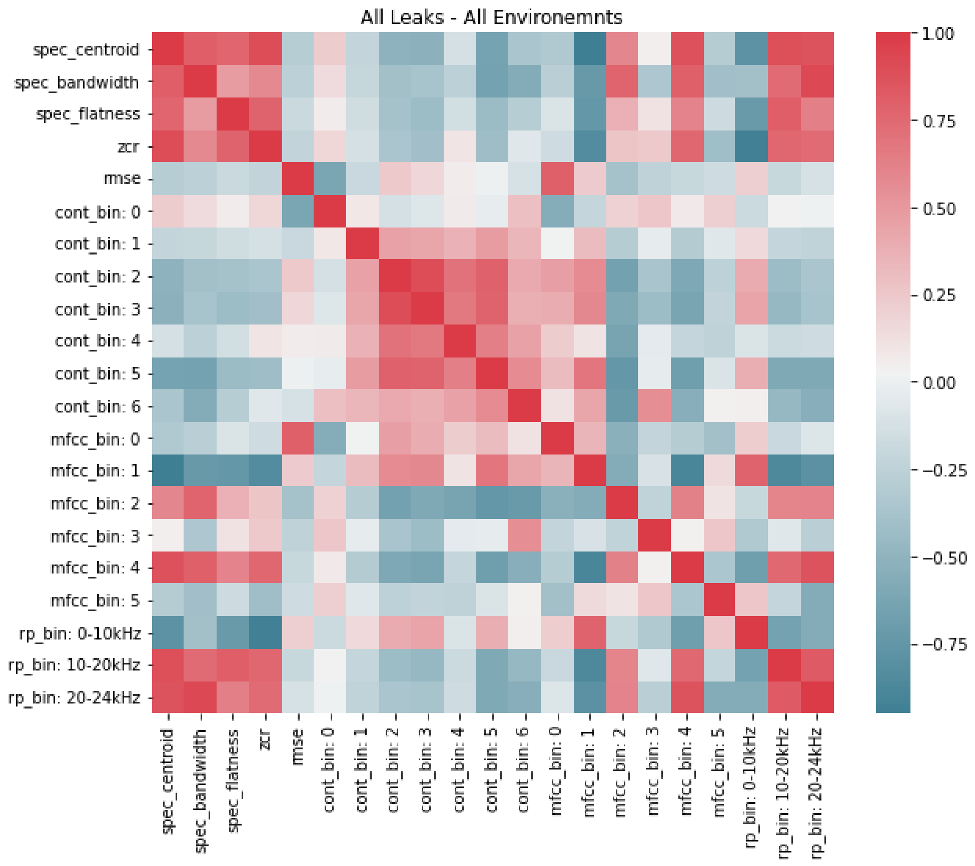


Figure 3.20: Correlation matrix for features extracted from all files recorded with microphone 1. The matrix depicts the correlation between all possible feature pairs. Darker red represents higher positive correlation, while colder blue is more negative correlation.

3.4.5 Feature Parameter Selection

Figures and domain knowledge is used to make an educated decision on the parameters *segment length* in Section 3.4.1 and *segment list* for power features in 3.4.3. We considered basing this selection on model performance. However, this was rejected after observing that different models performed differently with different values and acknowledging that this selection should be data-based, not model-based.

Selecting Segment List for Power Feature

As mentioned in Section 3.4.3, there is no correct selection of segments, and it will depend on the data. It might be reasonable to assume that fewer segments might be preferable to generalize it. If we were to use 1kHz segments as in Figure 3.19, one segment might be good for one leak but bad for another. The subfigure at the top left indicates that bin 17 is a segment that has higher values for leaks than not. However, this segment does not have the same separability in the other plotted examples. Training a model on the first leak and testing it with the others could result in a poorly performing model. This would not be an issue if each gas leak we encountered were present in the dataset. Since this is not the case, it is assumed that a few segments are preferable to make a more general model. Figure 3.19 indicates that the separation between leaks and no-leaks begins at around 20kHz in all four combinations of vent/tubeleak and work/hydr noises. 20kHz appears like a natural beginning of one segment, which goes to the highest frequency, 24kHz. According to [18], leaks emerge above 10kHz generally. Including a segment below this (0-10kHz) would be another good segment based on this information. Between 10-20 kHz can either be one segment or further divided. The figure indicates that this section is mostly unimportant for this data and is kept as one.

Selecting File Segment Size for Frames

As was for segment list selection above, there is no standard size that would be ideal here. The goal is to find a value that compromises between being long enough for the data not to be too volatile but short enough to extract as many data points as possible.

In Figure 3.21 the spectral bandwidth value for frames with increasing framelength is plotted, but the following observations also hold for other features. The plot to the left, with 1 second, shows numerous orange points (leaks) grouped with the blue (no leaks). The contamination is not as strong in the middle and right plots, where 5 and 10 seconds are used, respectively. The difference between the final two is not that great either, and since using 5 seconds gives more data, this is ultimately chosen for the frame length.

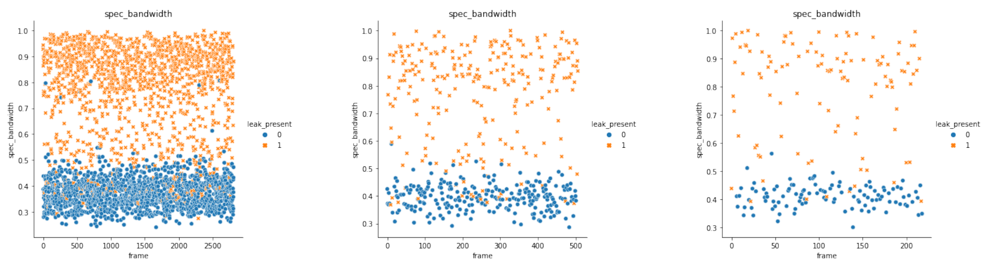


Figure 3.21: *Spectral bandwidth* plots for frames extracted from 50 files with varying frame lengths. 1s is used in the left plot, 5s in the middle, and 10s in the rightmost plot. The environment is *work*, and *tubeleak* is the leak. Recorded with microphone 1, and the frames are shuffled. Orange points are from files with a leak, and blue is without.

3.5 Machine Learning

This thesis explores which approach between gas leak- and anomaly detection is most suitable for the robot, given the limitations in Section 1.2. Machine learning is the tool that allows the robot to learn how to do either of these. Several models from the tree-based family are selected for the binary classification problem of gas leak detection, and three One class classifiers are chosen for the anomaly detection. In the following sections, we present the reason for selecting these particular models, how they would be used on the robot, and finally, how the models are implemented in code.

3.5.1 Tree Based Binary Classification

This family was selected to represent the gas classification approach because they are known to perform well even with restricted tuning and had been proven in a similar context by Ning *et al.* [14]. In addition, these models are often more interpretable than at least the deep learning models. Interpretability is essential since this thesis aims not to maximize the accuracy but to learn more about leak detection for further work. All features are used, but the *relative* option in the frequency bin power is set to *False*. The reason is discussed in 4.1.

To implement the models Sklearn's [55] *tree* and *ensemble* modules were used. The models are initialized with the selected hyperparameters, and the model instance is trained by calling *fit(X_train,y_train)* on it. All the tree-based models are supervised, so they require training samples and the target value for those samples. In this case, the target is the column "leak_present." Before training, this is separated from the rest of the features and named y. The remaining data is then named X. After the model is trained, it can be used to predict the "leak_present" vector for the test data by calling *predict(X_test, y_test)* on it. Code listing 3.4 shows all these steps. The only argument in the *DecisionTreeClassifier* call is "random_state=0," which is added so that the model always behaves the same to achieve reproducibility. Since no other arguments are passed into the method, the model will use the default hyperparameters. In Appendix B.1 each models hyperparameters are presented.

```
from sklearn import tree
dt_model = tree.DecisionTreeClassifier(random_state=0)
dt_model = dt_model.fit(X_train,y_train))
predictions = dt_model.predict(X_test, y_test)
```

Code listing 3.4: Code for initializing, training, and testing a model using Sklearn.

3.5.2 One-Class Novelty Detection

The models chosen to represent the second approach of anomaly detection are One-Class SVM, Isolation Forest, and Local Outlier Factor. They were selected since they displayed some promising signs for the task in [8], ass well as being highlighted by Sklearn in a *Novelty and Outlier Detection* introduction [3]. These models also have in common with the tree-based models that it is easier to interpret the results than deep learning methods such as the CNN Johnson *et al.* [1] uses. All the features are also used here, but now relative power is used as well.

Sklearn modules were also used to implement the OCC models, and the *fit()* and *predict()* methods from above are similarly used. However, since the models are unsupervised with only one class, the target value will not be passed in the methods. Instead, only the "normal" data should be used during training, but note that the "leak_present" column will be used for this. A vector with predictions is returned when the predict method is called on the model. That vector consists of

[3] `https://scikit-learn.org/stable/modules/outlier_detection.html`

the predictions for each data point; if the predicted value is 1 it is classified as normal, and if it is -1 it is classified as an anomaly. In Appendix B.2 each models hyperparameters are presented

3.6 Experiments

Several experiments were conducted to measure the performance of the models with the goal of evaluating them in general, as well as in situations as close to the intended use as the dataset allows. First, the experiments for gas leak detection are presented, followed by the experiments related to anomaly detection.

For each experiment, the relevant data were extracted, and the models were trained using a Cross-validation technique called Leave-One-Out (LOO). In practice, this means the training is done in three folds where each combination of two recordings is used for training, and the last is used for testing. The final score is the average of the three folds. This is inspired by the experiments conducted in Johnson *et al.* [1], which are the ones behind the dataset that is used. They say the motivation for using the LOO approach is to avoid data leakage between training and testing datasets and to be able to observe potential overfitting in a specific recording session where the scores vary drastically between folds.

3.6.1 Experiments for Gas Leak Detection

First, the models will be trained and tested comparably to experiment *E1* conducted in [1] to compare this approach to the CNN spectogram approach. E1 is, as we understand it, an experiment where all the data is from microphone 1 with all leaks and environmental noises, and their CNN model is evaluated using Leave-One-Out (LOO) for recording sessions and accuracy score as the metric. This experiment will be referred to as E1 in the following chapters.

Secondly, several smaller experiments were crafted to utilize the data to test situations that the robot might encounter. Following the name notation of E1; experiments E2-E4 will be presented below:

- E2: How would a model perform if the robot encountered a location with an unheard environmental noise? This is tested by training on one type of noise and testing on others.
- E3: How would a model perform if it encounters a leak it has not heard before? This is tested by training on one leak and testing on another,

with all noises included.

E4: How would the model perform if the leak occurred at a different location than expected? This is tested by training on data gathered from several microphones and testing on a different one.

The experiments are all executed by the same code that is presented in Appendix A.3. For each experiment, additional lines are added to get only the relevant data. Code listing 3.6 illustrates the lines of code that are different for each experiment. Those particular lines give experiment E2, where using different environments are tested. By changing the variable ’environment’ to ’leak_type,’ and ’work,’ and ’hydr’ to, for instance, ’tubeleak’ and ’ventleak,’ we get E3.

```
X_train_E2 = X['environment'].isin(['work'])]
X_test_E2 = X['environment'].isin(['hydr'])]
```

Code listing 3.5: Simplified code showing how train and test data is separated for each experiment from all the data, called X. X_train_E2 is the training set for E2 with only the *work* environmental noise, and X_test_E2 is the test set with only *hydr* noise.

3.6.2 Experiments for Anomaly Detection

Testing the anomaly detection models requires different experiments than the binary classification of gas leak detection. Our goal is to test if the models can distinguish normal data from irregular data. In this thesis, this is done by training on one type of noise without leaks and then testing with the same noise with all the gas leaks included. First we test with the main noises *work* and *hydr*, before training with for example *work* and testing with *work_low* in order to see if the model handles slight variations in the noise.

The same code executes the experiment as for the binary classification displayed in Appendix A.3. The difference is that for the lines in 3.6, both X_train and X_test should have the same environment. In addition, X_train should only contain data where ’leak_present’ == 0, meaning that there is no leak.

```
X_train = X[X['environment'].isin(['work']) & X['leak_present'] == 0)]]
X_test = X['environment'].isin(['work'])]
```

Code listing 3.6: Simplified code showing how train and test data is extracted for anomaly detection experiments. X_train selects the data points where 'environment' is '*work*' and no leak is present. X_test is of same environment as train data, but also includes data with leaks.

Chapter 4

Results

This chapter presents the results for all the experiments introduced in Section 3.6. The experiments for the tree-based binary classification approach E1-E4 are presented in 4.1 and start with a summary of the tests, followed by the results and potentially exciting observations. The results for the anomaly detection approach are presented in an equal manner in 4.2.

4.1 Gas leak detection results

This section presents the results of the experiments E1-E4. All the models are first trained using the default parameters, and hyperparameter tuning is only done if it appears to be necessary due to bad results or potential overfitting.

Experiment 1 Results

Experiment 1 (E1) tested how well the model classified leaks when all environmental noises and leak types were used and recorded with microphone 1. The accuracy score is used as the evaluation metric since [1] uses it in their experiment, allowing a fair comparison between their CNN and the tree-based models. In Section 2.3.3, it was pointed out that the accuracy score might be a weak evaluation metric in unbalanced datasets. However, the metric is useful here since there are equal amounts of each target class.

The score is presented in Table 4.1 alongside the results of the CNN model on E1. The scores are divided into condition and noise types to see potential weaknesses in specific areas. By inspecting the table we observe that the accuracy is in the high 90s for most conditions for all the models, which is comparable to the

CNN model. The ensemble methods RF, Ada, and Xgb, get slightly higher accuracy than the DT model, and the Xgb model seems to have the highest accuracy over all the conditions. The Xgb model outperform the CNN model in all noise types except hydr.

Table 4.1: Table showing the results of Experiment 1 (E1) for four models, and the CNN model from [1]. E1 is an experiment where all data recorded from microphone 1 is used, and LOO for recording session is used an averaged.

Condition	Noise Type	DT	RF	Ada	Xgb	CNN
Vent Leak	Lab	98.57 ± 1.52	99.86 ± 0.18	100.0 ± 0.0	99.9 ± 0.14	96.63 ± 3.07
Vent Low	Lab	95.75 ± 3.02	97.44 ± 2.46	97.64 ± 2.53	96.94 ± 2.40	94.45 ± 5.88
Tube Leak	Lab	97.04 ± 1.68	97.73 ± 2.37	98.12 ± 1.81	97.82 ± 2.23	99.11 ± 0.77
Average	Lab	97.12 ± 1.15	98.34 ± 1.08	98.58 ± 1.01	98.22 ± 1.23	96.73 ± 3.90
Vent Leak	Work Low	95.25 ± 0.60	98.66 ± 1.33	98.22 ± 1.77	98.81 ± 1.18	92.65 ± 3.73
Vent Low	Work Low	90.7 ± 3.26	94.06 ± 0.29	93.86 ± 0.20	94.26 ± 0.53	91.17 ± 0.77
Tube Leak	Work Low	93.92 ± 1.70	97.01 ± 1.50	95.81 ± 1.20	96.41 ± 1.44	97.43 ± 1.47
Average	Work Low	93.29 ± 1.90	96.58 ± 1.90	95.96 ± 1.78	96.49 ± 1.85	93.89 ± 3.43
Vent Leak	Work	95.69 ± 1.34	99.12 ± 0.47	98.53 ± 1.09	99.02 ± 0.73	98.46 ± 0.88
Vent Low	Work	93.38 ± 1.98	95.84 ± 2.52	95.45 ± 2.49	95.25 ± 2.12	93.35 ± 4.96
Tube Leak	Work	92.7 ± 1.39	95.85 ± 2.72	94.86 ± 3.25	95.85 ± 2.51	97.02 ± 2.73
Average	Work	93.92 ± 1.27	96.94 ± 1.54	96.28 ± 1.61	96.71 ± 1.65	96.28 ± 3.66
Vent Leak	Hydr Low	96.72 ± 1.28	98.71 ± 0.77	96.13 ± 0.72	98.21 ± 0.25	98.55 ± 1.29
Vent Low	Hydr Low	91.83 ± 0.98	94.72 ± 1.10	93.72 ± 0.50	94.32 ± 0.65	87.62 ± 6.58
Tube Leak	Hydr Low	91.73 ± 1.44	96.06 ± 0.76	93.2 ± 0.46	95.57 ± 0.63	96.03 ± 3.73
Average	Hydr Low	93.43 ± 2.32	96.49 ± 1.65	94.35 ± 1.27	96.03 ± 1.62	94.07 ± 6.27
Vent Leak	Hydr	93.31 ± 0.56	95.69 ± 1.78	93.6 ± 0.77	95.6 ± 1.84	94.24 ± 2.13
Vent Low	Hydr	89.46 ± 1.33	94.73 ± 1.40	93.06 ± 2.27	94.74 ± 1.37	92.75 ± 2.77
Tube Leak	Hydr	88.33 ± 0.90	92.84 ± 1.32	89.41 ± 0.48	92.54 ± 2.23	98.39 ± 0.81
Average	Hydr	90.37 ± 2.13	94.42 ± 1.18	92.02 ± 1.86	94.29 ± 1.28	95.24 ± 3.20

Figure 4.1 shows the average feature importance over each fold for the Random Forest model during this experiment. We observe that 'p_bin: 20-24kHz' is the dominant feature, with the following five except 'mfcc_bin: 5' were noted to be heavily correlated to this feature, as seen in Figure 3.20.

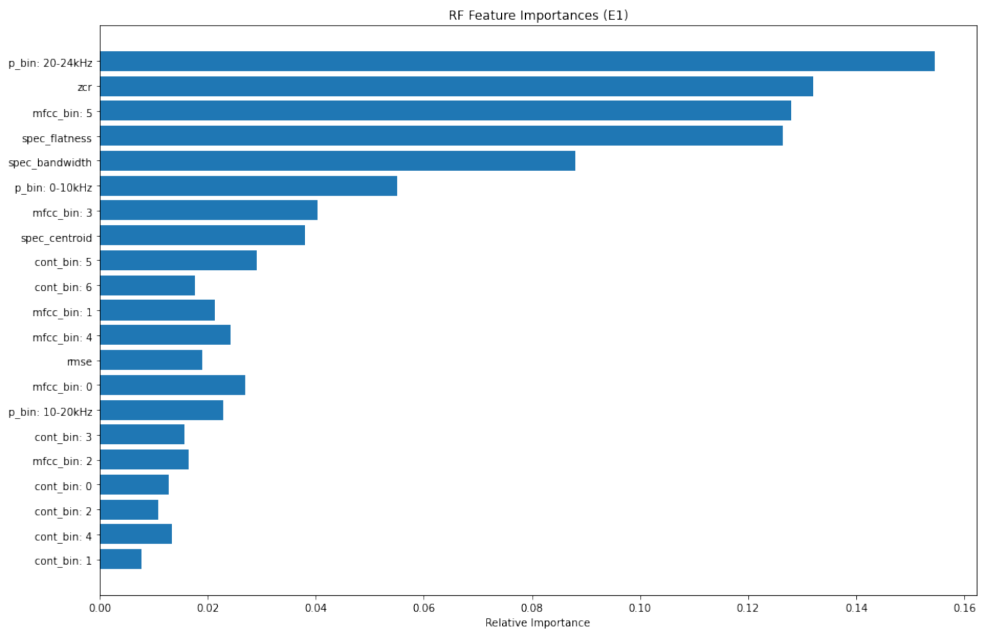


Figure 4.1: Average feature importance for three iterations of the LOO method on the Random Forest model with training data as explained in E1.

To test how using fewer features impacts the result, the RF model is tested with *all*, *top five* and the *top-ranking* features from the feature importance. RF is used since that is the model from which the feature importance is obtained. The results are presented in Table 4.2. We observe that using *all* or *five* features yields almost identical results, with the highest drop in average accuracy being just under 2% in the hydr case. For the model trained with only one feature, the average accuracy is around 10% lower in all cases.

Table 4.2: Table showing the results of Experiment 1 (E1) for the Random Forest model, with all, top five and top one features used.

Condition	Noise Type	RF - All features	RF - Top 5	RF - Top 1
Vent Leak	Lab	99.86 ± 0.18	99.86 ± 0.18	79.48 ± 4.88
Vent Low	Lab	97.44 ± 2.46	97.64 ± 2.53	81.21 ± 1.95
Tube Leak	Lab	97.73 ± 2.37	97.83 ± 2.44	75.08 ± 3.68
Average	Lab	98.34 ± 1.08	98.44 ± 1.00	78.59 ± 2.57
Vent Leak	Work Low	98.66 ± 1.33	98.81 ± 0.88	90.94 ± 0.12
Vent Low	Work Low	94.06 ± 0.29	92.98 ± 1.55	86.99 ± 4.62
Tube Leak	Work Low	97.01 ± 1.5	94.81 ± 1.32	85.74 ± 3.47
Average	Work Low	96.58 ± 1.90	95.53 ± 2.43	87.89 ± 2.21
Vent Leak	Work	99.12 ± 0.47	98.23 ± 0.48	92.48 ± 0.53
Vent Low	Work	95.84 ± 2.52	95.74 ± 2.8	88.95 ± 1.86
Tube Leak	Work	95.85 ± 2.72	94.18 ± 1.81	82.84 ± 2.01
Average	Work	96.94 ± 1.54	96.05 ± 1.67	88.09 ± 3.98
Vent Leak	Hydr Low	98.71 ± 0.77	98.51 ± 0.87	89.98 ± 1.52
Vent Low	Hydr Low	94.72 ± 1.1	94.42 ± 0.61	89.24 ± 1.28
Tube Leak	Hydr Low	96.06 ± 0.76	93.6 ± 0.89	75.0 ± 0.84
Average	Hydr Low	96.49 ± 1.65	95.51 ± 2.14	84.74 ± 6.89
Vent Leak	Hydr	95.69 ± 1.78	94.98 ± 0.69	87.61 ± 0.88
Vent Low	Hydr	94.73 ± 1.4	94.04 ± 1.91	86.09 ± 1.6
Tube Leak	Hydr	92.84 ± 1.32	89.41 ± 0.63	69.41 ± 1.73
Average	Hydr	94.42 ± 1.18	92.81 ± 2.43	81.04 ± 8.24

Experiment 2 Results

Experiment 2 (E2) tests how the model performs when encountering background noise that was not present during training and is performed by training on one noise and testing on another. The first test is done by training on hydr and testing on work, and the second test is the opposite. This experiment is the reason why *relative* frequency bin power is not used. Since hydr has much more energy in total than work, but the same in the frequencies above 20 kHz, they will have completely different values. The models used are still the default from E1. The evaluation metric used is also the same as in E1 to allow comparison with E3 from [1]. Unfortunately, the researchers do not deliver any in-depth explanation of their implementation of this experiment, so there is no guarantee that they are identically executed. However, the experiments are deemed equal enough to make a broad comparison. Again, only microphone 1 is used.

The accuracy score for all the models trained on hydr and tested on work is presented in Table 4.3. We observe that the Xgb model achieves the best average accuracy, around 85%. Interestingly, the standard deviation is almost 10 for vent and tubeleak, meaning there is a high variation in accuracy between the folds.

In Table 4.4 the accuracy score for all the models when trained on work and tested on hydr is presented. The table reveals that all the models reach an almost identical accuracy for all leak conditions.

Table 4.3: Accuracy when training on *hydr* and testing on *work*

Condition	DT	RF	Ada	Xgb
Vent Leak	72.59 ± 16.7	77.06 ± 2.06	85.03 ± 11.8	85.8 ± 9.18
Vent Low	68.41 ± 15.6	75.43 ± 1.99	84.07 ± 10.7	88.18 ± 4.78
Tube Leak	65.53 ± 17.9	74.74 ± 1.93	78.24 ± 8.74	80.54 ± 8.98
Average	68.84 ± 2.89	75.74 ± 0.97	82.44 ± 3.00	84.84 ± 3.19

Table 4.4: Accuracy when training on *work* and testing on *hydr*

Condition	DT	RF	Ada	Xgb
Vent Leak	76.0 ± 6.59	72.09 ± 0.53	72.09 ± 0.53	72.29 ± 0.4
Vent Low	75.97 ± 6.82	72.86 ± 1.09	73.17 ± 1.52	73.27 ± 1.66
Tube Leak	75.19 ± 5.29	71.76 ± 0.0	71.96 ± 0.13	71.96 ± 0.13
Average	75.72 ± 0.37	72.24 ± 0.46	72.40 ± 0.54	72.50 ± 0.55

To test the influence of the strongest feature a DT with max-depth 1 is tested. A visual representation of this decision tree can be viewed in Figure 4.2. The tree shows that by simply using ≈ -10 as a threshold for *p_bin: 20-24kHz*, we can classify around 2000 samples correctly and get only 100 wrong.

Table 4.5 shows the average accuracy for the DT model. When testing on hydr the score is equal to the other models in Table 4.4, but for work, the model clearly outperforms the other models in Table 4.3.

Table 4.5: Table showing the average accuracy over all the leak conditions for each of the two combinations of train and test data, trained on a Decision Tree model with *max-depth* = 1.

Train data	Test data	DT (max_depth=1)
Work	Hydr	72.37 ± 0.53
Hydr	Work	95.01 ± 1.91

To find out why the accuracy is so much worse when testing on hydr, we plot two features expected to be strong based on Figure 4.1 for the training and testing data. In Figure 4.3, we discover a cluster of outliers in the hydr-test data of frames

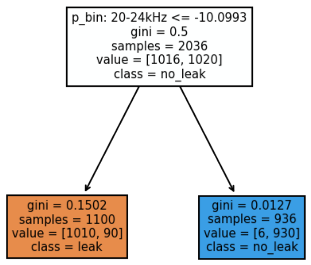


Figure 4.2: Figure representing the structure of a Decision Tree with the *max-depth* = 1, trained on one fold of the data containing only *work* environment noise. The white rectangle is the root node where data with *p_bin: 20-24kHz* values above or below -10 are passed to either the orange or blue leaf node. If data reaches the orange box to the left, it is classified as *leak* and *no_leak* if it reaches the blue node to the right. *Samples* is how many data points arrived at that node, and the *value*, says which class all the samples stem from, with the first index being leak, and the second index being no-leak.

without leak but with a high enough *p_bin: 20-24kHz* value to indicate that they are.

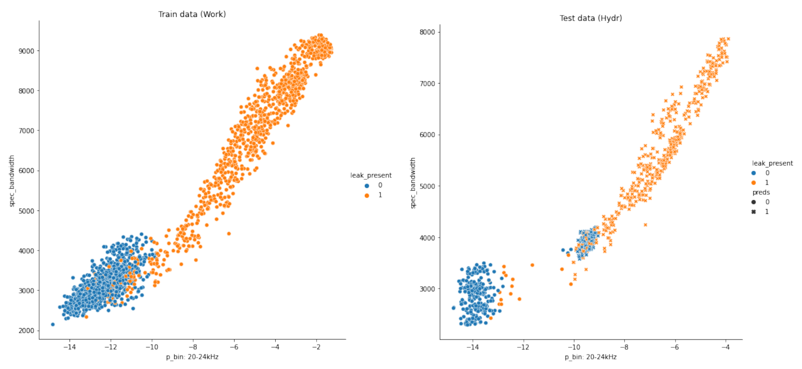


Figure 4.3: Plots of *spectral bandwidth* and *p_bin: 20-24kHz* features for the train (left) and test data (right) with predictions for E2. Training data is only the *work* background noise without leaks, and the test data is the *hydr* noise with leaks included. There is a cluster of outliers in test data where the *p_bin: 20-24kHz* value is between -10 and -8.

In Figure 4.4 the frequency spectrum above 20 kHz for three files are plotted, where the orange is of one of the outliers from Figure 4.3. The two others are from the work environment, with leak (blue) and without leak (green), and is used to compare.

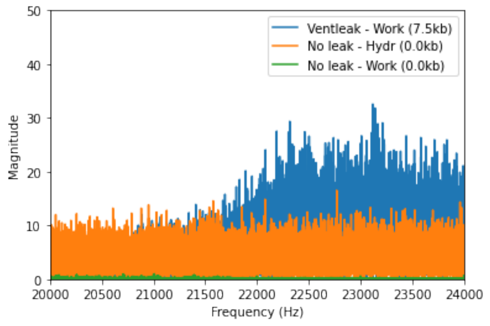


Figure 4.4: Frequency spectrum of three signals showing one of the outliers (orange) compared to one leak (blue) and non-leak (green) with work noise.

Experiment 3 Results

Experiment 3 (E3) evaluates how well the model performs when encountering a leak that was not present during training. We emulate this by using data containing one leak, such as a vent leak, during training and the other for testing, and vice versa. F1-score is used as a metric since there is no similar experiment to compare to in [1].

The results for training on ventleak and testing on tubeleak are presented in Table 4.6. We observe that the ensemble methods score relatively well, with an average above 90%. The only exception is the Xgb model with 86%, which is drastically lowered by the lab noise type where the accuracy was 62% with a 36% standard deviation.

Table 4.6: f1-score of models trained on **ventleak** and tested on **tubeleak**, with all environments and microphone 1 used.

Noise Type	DT	RF	Ada	Xgb
Lab	34.83* ± 20.2	99.0 ± 0.55	97.61 ± 1.62	61.9 ± 35.8
Work Low	73.71 ± 23.3	94.98 ± 1.8	94.32 ± 1.98	95.2 ± 1.9
Work	67.33 ± 33.1	93.89 ± 3.95	93.3 ± 1.47	94.16 ± 3.81
Hydr Low	81.39 ± 12.1	92.49 ± 0.92	92.5 ± 0.67	92.3 ± 0.52
Hydr	74.41 ± 12.1	88.21 ± 1.53	85.71 ± 2.85	87.28 ± 2.62
Average	66.33 ± 16.3	93.72 ± 3.50	92.69 ± 3.89	86.17 ± 12.4

In Table 4.7 the f1-score for training on tubeleak and testing on ventleak is shown, and this gives even better results with all models reaching the high nineties.

In Table 4.8 we see the results of a test done with same conditions as in in Table 4.7, only with microphone 4 instead of 1. The ensemble models handle the

Table 4.7: f1-score of models trained on **tubeleak** and tested on **ventleak**.

Noise Type	DT	RF	Ada	Xgb
Lab	99.02 ± 0.83	100.0 ± 0.0	99.9 ± 0.13	99.9 ± 0.13
Work Low	94.37 ± 0.05	98.94 ± 1.05	97.44 ± 2.25	98.8 ± 1.19
Work	94.98 ± 2.53	99.12 ± 0.63	97.6 ± 1.79	98.65 ± 0.97
Hydr Low	92.76 ± 2.08	96.84 ± 0.14	95.1 ± 1.78	97.04 ± 0.14
Hydr	92.5 ± 1.9	95.32 ± 1.59	93.81 ± 0.29	95.19 ± 1.76
Average	94.73 ± 2.34	98.04 ± 1.71	96.77 ± 2.11	97.91 ± 1.63

more challenging position better than the DT model, only losing around 5% in average accuracy. The reason for calling it more challenging is due to Figure 3.9 in the EDA.

Table 4.8: f1-score of models trained on **tubeleak** and tested on **ventleak** using microphone 4.

Noise Type	DT	RF	Ada	Xgb
Lab	91.33 ± 6.8	94.48 ± 3.19	89.84 ± 10.2	93.32 ± 5.3
Work Low	88.27 ± 0.36	95.96 ± 3.12	91.55 ± 2.99	94.96 ± 2.93
Work	90.51 ± 3.52	94.95 ± 2.56	92.6 ± 2.35	94.63 ± 1.79
Hydr Low	52.82 ± 26.1	95.41 ± 1.11	90.77 ± 1.44	95.54 ± 0.58
Hydr	52.89 ± 15.5	87.86 ± 2.56	83.4 ± 2.78	83.66 ± 3.5
Average	75.17 ± 18.2	93.73 ± 2.97	89.63 ± 3.24	92.42 ± 4.44

Experiment 4 Results

Experiment 4 (E4) aims to test how using recordings from different microphone positions impacts the results. Only recordings from microphones 1-3, work and hydr environments, and tube- and ventleak was used to reduce the amount of data. Each combination of two microphones for training and one for testing is used, for instance, training with microphones 1 and 2 and testing on microphone 3. The f1-score is also here used since there are no results from [1] to compare. The results are presented in Table 4.9, Table 4.10 and Table 4.11. Each table will be referred to with the number of the microphone used for testing.

Microphone 3 gives the best results, with the Xgb model reaching an average of around 94%. Testing on microphone 1 yields second best, with the top-performing model Ada getting an average of 87%. The final test where microphone 2 is used gives an average of 78% by the Xgb model, but the tube leaks reduce the average while it does well on vent leaks. This same trend is also evident when testing on microphone 3. When testing on microphone 1, the Ada model performs very well on both tubeleak and ventleak in the work environment but worse in the hydr environment.

Table 4.9: Table showing f1-score for models trained with data from microphone 2 and 3, and tested on microphone 1.

Condition	Noise Type	DT	RF	Ada	Xgb
Vent Leak	Work	86.56 ± 4.57	95.84 ± 3.74	97.06 ± 1.68	94.67 ± 6.08
Tube Leak	Work	82.44 ± 3.0	92.58 ± 0.62	92.55 ± 1.36	91.01 ± 4.56
Vent Leak	Hydr	80.98 ± 5.29	79.05 ± 1.64	79.78 ± 1.37	80.94 ± 3.02
Tube Leak	Hydr	77.76 ± 0.91	79.15 ± 0.7	79.94 ± 1.69	80.03 ± 0.16
Average		81.94 ± 3.16	86.66 ± 7.64	87.33 ± 7.63	86.66 ± 6.32

Table 4.10: Table showing f1-score for models trained with data from microphone 1 and 3, and tested on microphone 2.

Condition	Noise Type	DT	RF	Ada	Xgb
Vent Leak	Work	83.28 ± 9.37	84.75 ± 7.29	89.4 ± 5.98	86.23 ± 7.14
Tube Leak	Work	67.07 ± 7.03	71.56 ± 3.38	68.55 ± 14.4	71.41 ± 5.47
Vent Leak	Hydr	77.75 ± 1.45	80.39 ± 1.0	84.96 ± 3.28	84.36 ± 1.86
Tube Leak	Hydr	66.95 ± 2.96	74.31 ± 0.69	63.69 ± 7.81	71.98 ± 5.35
Average		73.76 ± 7.03	77.75 ± 5.15	76.65 ± 10.7	78.50 ± 6.83

Table 4.11: Table showing f1-score for models trained with data from microphone 1 and 2, and tested on microphone 3.

Condition	Noise Type	DT	RF	Ada	Xgb
Vent Leak	Work	94.43 ± 2.4	95.76 ± 4.11	97.69 ± 0.98	97.48 ± 2.52
Tube Leak	Work	86.34 ± 4.49	87.72 ± 3.62	88.7 ± 4.01	89.94 ± 2.65
Vent Leak	Hydr	88.67 ± 0.76	94.84 ± 1.41	93.14 ± 3.37	95.61 ± 1.31
Tube Leak	Hydr	81.3 ± 0.57	91.5 ± 1.17	87.87 ± 0.62	90.43 ± 0.83
Average		87.68 ± 4.71	92.46 ± 3.16	91.85 ± 3.92	93.36 ± 3.25

4.2 Anomaly Detection Experiment Results

Only one experiment is conducted for the novelty detection methods. Data is recorded from microphone 4, and one test is done for each environment. The model is trained with data without leaks and tested with data from the same environment but with around 50 % leaks included. We use the f1-score evaluation and LOO.

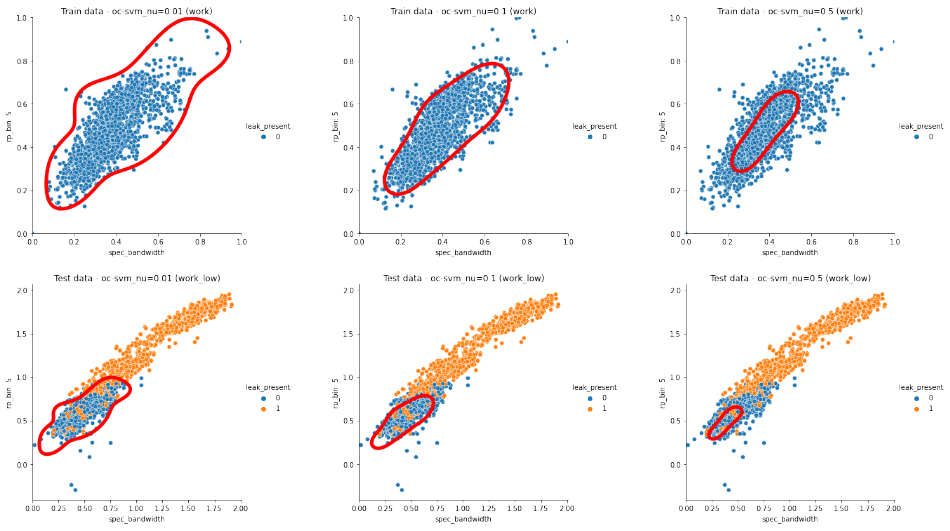


Figure 4.5: Top row shows plots of the *decision boundary* (red line) created around the training data by OC-SVM models with varying nu (0.01, 0.1 and 0.5) with only two features. The training data is without leaks and only with the *work* environmental noise recorded from microphone 4. Bottom row shows the same boundary overlaid the test data, which is from the *work_low* environment, now with all leaks included.

First, the impact of the *nu* parameter for the OC-SVM model is explored. Figure 4.5 shows the *decision boundary* returned by three OC-SVM models with different *nu* that is trained on the work environmental noise. We see that increasing the *nu* parameter creates a stricter boundary, resulting in a higher probability for new data to fall outside and be classified as anomalous. Figure 4.6 is the same plot with a model trained on the hydr noise. We observe that there are two clusters in the training data, and only one in the test data.

Several different values for the *contamination* hyperparameter in IF and LOF and *nu* in the OC-SVM was tested. The best accuracy was with the default value of 0.5 for IF and LOF and 0.1 for OC-SVM. The results are presented in table Table 4.12. We observe that the average f1-score for the lab environment is best for all the models with between 85-90%, and worst for hydr with the best model LOF getting 73%. For both hydr and work, all the models perform well on ventleak and ventlow, with accuracy between 80-90%. However, the tubeleak drags the average down, with the best performing on the hydr data being IF with only 53%.

In Figure 4.7 the decision boundary is plotted for each model when trained and tested using the lab environment noise. We see that the boundary created by OC-SVM and IF is similar and restricted to the central cluster of points, while LOF

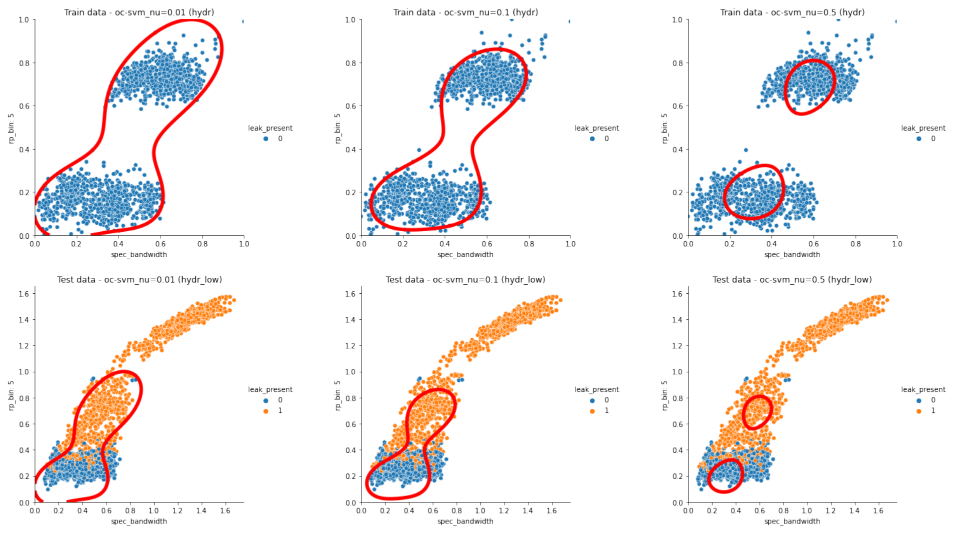


Figure 4.6: Plots with same conditions as in Figure 4.5, except *hydr* is used in top row and *hydr_low* is used in bottom row.

Table 4.12: Table of f1-score for three One-class classifiers that is using same environmental noise during training and testing.

Train Env.	Test Env.	Condition	OC-SVM	LOF	IF
Work	Work	Vent Leak	82.43 ± 7.30	89.30 ± 5.08	82.35 ± 6.97
Work	Work	Vent Low	83.58 ± 5.17	87.26 ± 4.89	82.83 ± 4.57
Work	Work	Tube Leak	69.93 ± 8.81	65.31 ± 13.5	66.33 ± 12.6
Work	Work	Average	78.65 ± 6.18	80.63 ± 10.8	77.17 ± 7.66
Hydr	Hydr	Vent Leak	81.34 ± 3.13	88.64 ± 0.19	81.38 ± 1.49
Hydr	Hydr	Vent Low	81.83 ± 0.99	88.58 ± 2.55	80.31 ± 0.64
Hydr	Hydr	Tube Leak	31.42 5.29	43.37 ± 2.37	53.12 ± 2.76
Hydr	Hydr	Average	64.87 ± 23.6	73.53 ± 21.3	71.60 ± 13.0
Lab	Lab	Vent Leak	85.14 ± 12.3	89.01 ± 12.0	91.16 ± 4.88
Lab	Lab	Vent Low	88.02 ± 6.77	86.00 ± 10.0	88.82 ± 5.09
Lab	Lab	Tube Leak	82.90 ± 12.4	86.97 ± 9.73	88.02 ± 4.71
Lab	Lab	Average	85.35 ± 2.09	87.33 ± 1.25	89.33 ± 1.33

encapsulates more of the deviant points closer to the origin.

The models are also tested by training on the loud version of the leak and testing on the lower version to see how it tackles minor variations in the noise. The results are presented in Table 4.13. The average score for work_low is slightly higher than before, mostly because of the tubeleak increasing to above 80% for all models. For hydr and hydr_low, the score is lower in all areas.

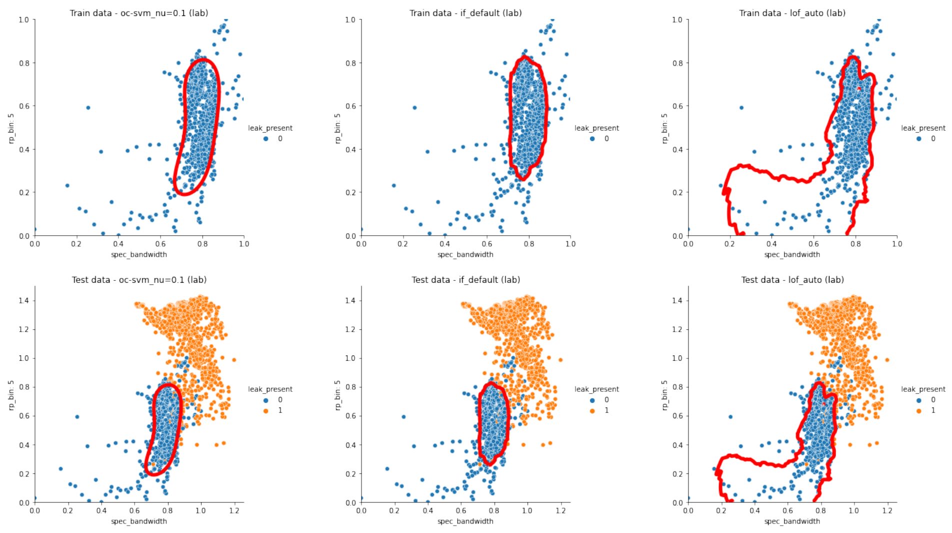


Figure 4.7: Top row shows the boundary created by the OC-SVM, LOF and IF models when trained on the *lab* environmental noise without leak (blue). Bottom row is the test data which is the same noise, now including data with leak (orange).

Table 4.13: Table of f1-score for three One-class classifiers training on the main version of the environmental noise and testing on the lower version.

Train Env.	Test Env.	Condition	OC-SVM	LOF	IF
Work	Work Low	Vent Leak	77.94 ± 6.38	83.74 ± 7.89	77.56 ± 5.86
Work	Work Low	Vent Low	82.23 ± 11.2	83.45 ± 11.8	81.13 ± 10.4
Work	Work Low	Tube Leak	82.07 ± 4.55	83.52 ± 7.54	81.24 ± 4.71
Work	Work Low	Average	80.75 ± 1.98	83.57 ± 0.12	79.98 ± 1.70
Hydr	Hydr Low	Vent Leak	82.87 ± 1.23	76.74 ± 0.72	69.79 ± 2.01
Hydr	Hydr Low	Vent Low	74.45 ± 4.37	71.06 ± 1.90	66.63 ± 1.42
Hydr	Hydr Low	Tube Leak	41.32 ± 2.26	42.60 ± 4.05	48.30 ± 1.58
Hydr	Hydr Low	Average	66.21 ± 17.9	63.47 ± 14.9	61.57 ± 9.47

Chapter 5

Discussion

In this Chapter, the results from Chapter 4 will first be discussed in Section 5.1. The experiments relevance and any weaknesses can also be found here. In Section 5.2 the two approaches considered in this thesis are compared to evaluate their suitability for being used by the robot.

5.1 Results Discussion

E1 Results:

This experiment was included to compare the features and models used here to the deep learning-based approach with CNN and spectograms seen in [1], even though the experiment itself might not be the most relevant to the intended use case. As was seen in Section 4.1 all the models perform comparably to the CNN and, in some cases, even outperform it.

Hyperparameter tuning was not done either. That could have extracted a few percent more out of the models. The reason for not tuning was that the score was so good that we could only get a slight increase with the added potential of overfitting due to no unseen data available.

The most interesting observation from the experiment was that the results remained good even though the number of features was reduced, even down to just one single feature. On the one hand, this should be considered a weak point for the models since they cant pick up on more complex relations. However, it might instead point to a weakness in the dataset. We saw in the Leak Type section of the EDA 3.2 that by looking at the energy in the 20-24kHz range, we could distinguish between leaks and no leaks. If there were included noise in this range as well in the dataset, the models might be forced to use more information from more features. The main takeaway from this experiment is that the tree-based models perform comparably to the CNN model, even with only one feature.

E2 Results:

This experiment is interesting because it indicates that the model, to some degree, could generalize beyond the background noise and that the model did not need to be trained on all possible environmental noise it could encounter.

We observed large standard deviations in Table 4.3, which can point to overfitting in one of the folds. Hyperparameter tuning could potentially resolve this, but it was not prioritized due to the Xgb model performing well.

Tables 4.4 and 4.5 aroused suspicion since all the models performed almost identical, especially considering how Xgb outperformed DT in Table 4.3. In Figure 4.3, we plotted the train and test data and observed a cluster of data points without leaks, with the power and spectral bandwidth values to be it.

After inspecting the files that these points are from, we observed that all have in common that they are files without leaks with the letter “m” after the microphone number instead of “l.” The meaning of these is not described in the dataset explanation, but they most likely stand for “max” and “low” volume settings on the microphones. These files all have energy in the frequencies above 20 kHz to indicate that they are leaks, which can be seen in Figure 4.4. Whether this was an error by the creators or actual observations is uncertain, but the fact that the model does not classify them as leaks is not worrying since this pattern was not present in the training set, and the model will only learn from the examples offered to it. In hindsight, this aspect of the data should have but was not discovered in the EDA.

Either way, even though the best average accuracy achieved in the experiment is close to 85%, it is still far away from the score achieved by the CNN model in [1]. They got an average accuracy of close to 98% for all vent- and tubeleaks and 90% for ventlow. The CNN models results were only slightly reduced in some cases by training on a different noise than testing, so it appears that this deep learning approach picks up on some information in the signals that are not represented in the features selected here.

E3 Results:

As with E2, this is an interesting experiment because it can potentially give some indication about what can be achieved in terms of detection of leaks not included in the dataset. The experimental results are impressive, particularly when tubeleak is used for training and ventleak for testing.

However, the experiment uses only one microphone 20 cm away from the leak, which also always comes from the same place each time. Therefore, this does not prove that by including tubeleaks in the dataset, we can always detect ventleaks as well. The experiment was also tested using microphone 4, which has a different orientation, and the results were good.

Some models had extreme standard deviation in accuracy, probably due to overfitting caused by lack of tuning, but again this was not prioritized due to the good results regardless.

The reason it performs best when trained on tubeleak and tested on ventleak is presumably a consequence of what we observed in Section 3.2. The energy in the top frequencies is much higher in the ventleak. Any split made in any trees which picks up on this, which a lot of the features do, since we know they correlate to *p_bin: 20-24kHz*, the ventleak will always be higher than the highest leak in tubeleak, so it should detect them. Note that the results are good the other way as well, so there must be something different the models pick up on as well, which is promising.

E4 Results:

The goal of including this experiment was to test if the models could detect leaks even though they were recorded from another position. The results are good when tested on microphones 1 and 3, which was expected since the other is then used for testing, and we observed they were pretty similar in Section 3.2. Microphone 2 was performing worse than the other was also expected. The same plot also showed that microphone 2 picked up on less of the frequencies above 20 kHz, which we have come to realize is so important

Anomaly Detection Results:

This experiment is reasonably similar to how it would look on the robot. However, more noise is expected than what is encountered here.

The accuracy is good in the work and lab environment. It is clear from Figure 4.6 that the issue with the hydr cluster from E2 also has an impact here, and the model created a border around data in the training that is normal in the test.

That the models struggle more with tubeleaks than ventleaks is not surprising since we know tubeleaks are less extreme and more difficult to hear as a leak, also for humans. Note that neither of the models is best in all cases, so we cannot conclude that one is better than the rest. Potentially, the models could be combined in an ensemble way where the predictions of all the models are taken into account.

In the experiment, the f1-score was used, which holds information about the relationship between precision and recall. In anomaly detection, recall is most important. We allow false alarms but not to classify something potentially dangerous as an anomaly. This should perhaps be more emphasized in the models, and the *nu* value of the OC-SVM should, for instance, be increased so that it was stricter and got fewer false negatives. On the other side, as seen in Figure 4.5, using nu=0.5 would lead to considerable false alarms that would drastically reduce its usefulness. Therefore, including a condition that, for instance, five consecutive predictions have to be anomalous for the robot to alert humans has the potential of reducing the impact of false alarms.

5.2 Comparison of Approaches

Several strengths and weaknesses have become apparent through the experiments for the *gas leak detection*, and *anomaly detection* approaches. In this section, they will be compared against each other to compare which might be most suitable for an autonomous inspection robot. For simplicity, gas leak detection will be referred to as option one and anomaly detection as option two.

The first option's main advantage is that the system's sole purpose is to detect what we are most worried about, which is leaks since that has the highest damage potential. However, if the robot is to replace human inspectors, it should be able to detect what humans do, which also includes faulty machinery, for instance. The second approach can also be used to detect leaks, as seen in Section 4.2, but this is not a priority above other defects. In addition, there is no way of knowing the anomaly's source after detecting it, leaving the human inspectors with no idea what the problem might be. This might be a more significant issue if we were to look for leaks in the ultrasonic range where humans are physically unable to hear it. Another strength of option one is that it would be possible to begin a localization procedure after the leak is located since we know that the source probably is a high-frequency leak somewhere. Looking at where the energy is most significant could potentially lead to the source.

In Section 4.1, option one was able to detect leaks in a different environment than it was trained on. It must be said that this is much easier when both are recorded from precisely the exact location with the same leak; however, it might be possible with sufficient data. If we assume that it is, this is a great advantage option one has over option two since this means that a new model would not need to be trained at each unique location the robot is inspecting. Using option two would require training new models for each new location.

The main advantage of option two is that it does not need examples of the anomalies in the data set to work. It only needs the normal data; over time, it will probably get better and better the more data it receives. Option two does need labeled data in the form of gas leaks. We saw in the EDA 3.2 that there is a big difference between tube- and ventleaks. There probably exist numerous variations of leaks with vastly different characteristics. Even though option one averaged around a 95% accuracy when training on another leak in E1, the conditions are ideal, so this is probably the most straightforward situation possible. Models learn from the data provided to them, so it makes sense to include a representative amount of potential leaks to create a general gas leak detector. Collecting the leak data can be challenging since some are hard to reproduce or dangerous. Repeating an experiment for leaks caused by material degradation would be complicated, and perhaps the sound from the dangerous gasses sound different from compressed air?

In [1] a CNN model is used for the leak detection instead of the tree-based models used here. In experiments E1 and E2, we saw that the CNN model was better at generalizing beyond the environment, but they performed evenly in the general E1 experiment. The discovery in E2 points to the main advantage of using a deep learning model like CNN. The feature extraction is done by the model, which saves time and is not as dependent on the domain knowledge of the creators of the features. It can also discover patterns in the data that cannot be found using manual feature extraction. The advantage of the tree-based methods is that they are more interpretable, meaning we, to a higher degree, can reason why the models have made their predictions. It can be said that this is valuable in an inspection setting since if a person knows the meaning of the features, they can observe the decision path down a Decision Tree, for instance, which is not possible in most deep learning models.

Chapter 6

Conclusions and Further Work

This chapter summarizes the thesis before we conclude whether we reached our goals and which approaches have the highest potential to be used in an autonomous inspection setting in industrial plants. Followed by a section where any recommendations for further work are presented.

6.1 Conclusions

In this thesis, we have examined different acoustic approaches for detecting damage in industrial plants with a robot. First, the theoretical foundation was laid, and several machine learning methods representing each approach were put forth to solve the problem. Several experiments to test the models were crafted and discussed.

The overarching goal of the thesis was to explore if it might be best to specialize the robot to detect gas leaks or to detect if something sounds wrong in general. Several sub-goals was also presented, and they were all completed to a varying degree.

The first sub-goal was to find a publicly available and relevant dataset, which was done with the *compressed air leak* dataset. It had weaknesses, primarily when used for creating the gas leak detection experiments, but proved suitable for anomaly detection. Nevertheless, several issues with it have been pointed out, and suggestions to improve it in the future are presented in Section 6.2.

The second sub-goal was to explore the data and learn about the characteristics of gas leaks, which was done in 3.2. Information about the difference between tubeleak and ventleak and the potential of looking at frequencies above 20 kHz was discovered. However, analysis of the environmental noise was not thoroughly performed, causing unforeseen observations in the experiments.

The third was to use the knowledge to extract relevant features and explore which are most vital. In Section 3.4 we used proven features from related works and crafted another that used what we had learned about the high frequencies of a leak. In some experiments, the feature importance was presented, and we saw what could be achieved with only a few features.

The final sub-goal was implementing reasonable machine learning models and testing them with the chosen dataset. Several well-proven methods from other domains were tested, and the experiments gave valuable insights to compare the two directions.

Ultimately, as mentioned initially, the goal was to explore the two approaches. However, in light of the discussion of the approaches in Section 5.2, it appears that gas leak detection focusing on the highest frequencies is the most suitable approach. This is mostly based on the fact that it avoids the necessity of training at each new location which would be strenuous. In addition, gas leaks are the most worrying defect on oil and gas platforms due to the potential damage extent. However, ideally, both approaches should be implemented, complementing each other and creating a more significant impact. Suppose a gas leak detector was implemented on an inspection robot on a platform. In that case, the processed data could be used to train potential anomaly detectors, automating the process.

6.2 Further Work

This thesis is only the beginning of much fascinating work in an impactful field. Much research is still required before the robot can alert its first human. The following section presents some encouragement for further work and begins with how to create an ideal dataset for this task.

Improved Dataset

The *IDMT-ISA-COMPRESSED-AIR* dataset created by Johnson *et al.* [1] was used in this thesis. It was created in order to fill a void in gas leak datasets, as well as encourage the same from others. It had several useful characteristics, but ultimately it was made with a different application in mind. If we were to create a more suitable dataset to continue the work in leak or anomaly detection in autonomous robot inspectors, the following aspects should be considered:

- Keep the microphone in one place, with some minor millimeters deviation in all directions, to account for the imprecision in the robot movement.
- Create leaks by releasing compressed air from a bigger tube and make it mobile. Move the leak source around the microphone instead of keeping it fixed to one place to resemble the fact that a leak does not appear from only one location.
- Record environmental noise from a location on a platform and superimpose the sound of leaks to that instead of playing the noise from a speaker.
- Use a sample rate that allows for ultrasonic frequencies and a microphone that permits it.

Improved Machine Learning

Besides using an improved dataset, testing different machine learning models and techniques should be considered. First and foremost, the impressive results achieved with deep learning in other fields such as *object detection* and *natural language processing*, as well as seeing the CNN model of [1] outperform the tree-based models in E2, makes makes it apperant that exploring this would be a reasonable. Regardless of losing interpretability. In theory, it is not unreasonable to assume that a DL model would be able to detect any gas leak in a spectogram image if it can detect a dog. The limiting factor is the amount of data available, and as explained in Section 5.2 it can be challenging to gather gas leak the data. Deep Auto Encoders could be considered for anomaly detection, which also appears to be the leading approach [5].

Until the vast amounts of data required are acquired, it would be reasonable to continue exploring the tree-based solution for leak detection since they perform well with less data. Other, more reliable methods should be explored for the anomaly detection direction. To further improve the models, several steps could be taken.

For anomaly detection, the models should at least be tuned better, and combining them in an ensemble fashion could potentially be interesting. Seliya *et al.* [56]s literature review of OCC mention several variations to the methods used here and some unused.

Implementing denoising of signals should be considered if more noisy data is to be used. In addition, augmenting the data should be considered, which means slightly altering the signals or features to get a more diverse dataset. Augmentation is common in for instance image classification [57].

Bibliography

[1] D. Johnson, J. Kirner, S. Grollmisch and J. Liebetrau, 'Compressed air leakage detection using acoustic emissions with neural networks,' *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, vol. 261, no. 1, pp. 5662–5673, 12th Oct. 2020.

[2] 'Equinor deploys a robot that can find its way around an offshore platform,' JPT. (14th Apr. 2021), [Online]. Available: `https://jpt.spe.org/equinor-deploys-a-robot-that-can-find-its-way-around-an-offshore-platform` (visited on 20/12/2021).

[3] A. Schenck, W. Daems and J. Steckel, 'AirleakSlam: Detection of pressurized air leaks using passive ultrasonic sensors,' in *2019 IEEE SENSORS*, ISSN: 2168-9229, Oct. 2019, pp. 1–4. DOI: `10.1109/SENSORS43011.2019.8956631`.

[4] A. Kroll and T. Gunther, 'Localization of compressed air leaks in industrial environments using service robots with ultrasonic microphones,' [Online]. Available: `https://www.ndt.net/search/docs.php3?id=20360`.

[5] E. C. Nunes, 'Anomalous sound detection with machine learning: A systematic review,' *arXiv:2102.07820 [cs, eess]*, 15th Feb. 2021. arXiv: `2102.07820`. [Online]. Available: `http://arxiv.org/abs/2102.07820` (visited on 15/09/2021).

[6] D. Y. Oh and I. D. Yun, 'Residual error based anomaly detection using auto-encoder in SMD machine sound,' *Sensors*, vol. 18, no. 5, p. 1308, May 2018, Number: 5 Publisher: Multidisciplinary Digital Publishing Institute. DOI: `10.3390/s18051308`. [Online]. Available: `https://www.mdpi.com/1424-8220/18/5/1308` (visited on 10/12/2021).

[7] T. B. Duman and B. Bayram, 'Acoustic anomaly detection using convolutional autoencoders in industrial processes,' in 1st Jan. 2020, pp. 432–442, ISBN: 978-3-658-07615-3. DOI: `10.1007/978-3-030-20055-8_41`.

[8] D. Kampelopoulos, G.-P. Kousiopoulos, N. Karagiorgos, V. Konstantakos, S. Goudos and S. Nikolaidis, 'Applying one class classification for leak detection in noisy industrial pipelines,' in *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, Jul. 2021, pp. 1–4. DOI: `10.1109/MOCAST52088.2021.9493355`.

[9] R. Müller, F. Ritz, S. Illium and C. Linnhoff-Popien, 'Acoustic anomaly detection for machine sounds based on image transfer learning,' *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, pp. 49–56, 2021. DOI: `10.5220/0010185800490056`. arXiv: `2006.03429`. [Online]. Available: `http://arxiv.org/abs/2006.03429` (visited on 28/09/2021).

[10] T. B. Quy and J.-M. Kim, 'Real-time leak detection for a gas pipeline using a k-NN classifier and hybrid AE features,' *Sensors*, vol. 21, no. 2, p. 367, 7th Jan. 2021, ISSN: 1424-8220. DOI: `10.3390/s21020367`. [Online]. Available: `https://www.mdpi.com/1424-8220/21/2/367` (visited on 10/12/2021).

[11] R. Xiao, Q. Hu and J. Li, 'Leak detection of gas pipelines using acoustic signals based on wavelet transform and support vector machine,' *Measurement*, vol. 146, pp. 479–489, 1st Nov. 2019, ISSN: 0263-2241. DOI: `10.1016/j.measurement.2019.06.050`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0263224119306189` (visited on 10/12/2021).

[12] R. P. da Cruz, F. V. da Silva and A. M. F. Fileti, 'Machine learning and acoustic method applied to leak detection and location in low-pressure gas pipelines,' *Clean Technologies and Environmental Policy*, vol. 22, no. 3, pp. 627–638, Apr. 2020, ISSN: 1618-954X, 1618-9558. DOI: `10.1007/s10098-019-01805-x`. [Online]. Available: `http://link.springer.com/10.1007/s10098-019-01805-x` (visited on 14/12/2021).

[13] F. Ning, Z. Cheng, D. Meng, S. Duan and J. Wei, 'Enhanced spectrum convolutional neural architecture: An intelligent leak detection method for gas pipeline,' *Process Safety and Environmental Protection*, vol. 146, pp. 726–735, 1st Feb. 2021, ISSN: 0957-5820. DOI: `10.1016/j.psep.2020.12.`

011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957582020319388 (visited on 10/12/2021).

[14] F. Ning, Z. Cheng, D. Meng and J. Wei, 'A framework combining acoustic features extraction method and random forest algorithm for gas pipeline leak detection and classification,' *Applied Acoustics*, vol. 182, p. 108 255, 1st Nov. 2021, ISSN: 0003-682X. DOI: 10.1016/j.apacoust.2021.108255. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003682X21003492 (visited on 02/11/2021).

[15] D. Henze, K. Gorishti, B. Bruegge and J.-P. Simen, 'AudioForesight: A process model for audio predictive maintenance in industrial environments,' in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Dec. 2019, pp. 352–357. DOI: 10.1109/ICMLA.2019.00066.

[16] T. D. Rossing and R. F. Moore. 'Science of sound, the | 3rd edition | pearson.' (2002), [Online]. Available: https://www.pearson.com/store/p/science-of-sound-the/P100000828672/9780805385656 (visited on 18/12/2021).

[17] A. Lerch, *An introduction to audio content analysis: applications in signal processing and music informatics*. Hoboken, New Jersey: Wiley, 2012, 272 pp., ISBN: 978-1-283-80405-9.

[18] 'Ultrasonic gas detectors with artificial neural network intelligence,' p. 3, [Online]. Available: https://s7d9.scene7.com/is/content/minesafetyappliances/Gassonic%20Ultrasonic%20Gas%20Leak%20Detection%20White%20Paper%20-%20EN.

[19] *Sampling (signal processing)*, in *Wikipedia*, Page Version ID: 1084648620, 25th Apr. 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sampling_(signal_processing)&oldid=1084648620 (visited on 18/05/2022).

[20] *Nyquist–shannon sampling theorem*, in *Wikipedia*, Page Version ID: 1086141927, 2nd Jun. 2022. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem (visited on 18/05/2022).

[21] 'Exploring communications technology,' Exploring communications technology. (), [Online]. Available: https://www.open.edu/openlearn/science-maths-technology/exploring-communications-technology/science-maths-technology/exploring-communications-technology (visited on 21/05/2022).

[22] T. Giannakopoulos and A. Pikrakis, 'Introduction to audio analysis: A MATLAB approach,' in *Introduction to Audio Analysis*, Oxford: Academic Press, 1st Jan. 2014, p. iii, ISBN: 978-0-08-099388-1. DOI: 10.1016/B978-0-08-099388-1.00010-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080993881000108 (visited on 19/05/2022).

[23] S. Abdoli, P. Cardinal and A. Koerich, *End-to-End Environmental Sound Classification using a 1D Convolutional Neural Network*. 18th Apr. 2019.

[24] Phonical, *English: View of a signal in the time and frequency domain*, 30th Nov. 2017. [Online]. Available: https://commons.wikimedia.org/wiki/File:FFT-Time-Frequency-View.png (visited on 22/05/2022).

[25] J. O. Smith, *Physical Audio Signal Processing*, 2010 edition. 2010. [Online]. Available: http://ccrma.stanford.edu/~jos/pasp/.

[26] J. Dempster, 'CHAPTER SIX - signal analysis and measurement,' in *The Laboratory Computer*, ser. Biological Techniques Series, J. Dempster, Ed., London: Academic Press, 1st Jan. 2001, pp. 136–171. DOI: 10.1016/B978-012209551-1/50039-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780122095511500398 (visited on 10/06/2022).

[27] *Mel-frequency cepstrum*, in *Wikipedia*, Page Version ID: 1088378317, 17th May 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Mel-frequency_cepstrum&oldid=1088378317 (visited on 06/06/2022).

[28] S. Russell and P. Norvig. 'Artificial intelligence: A modern approach, 3rd edition.' (), [Online]. Available: https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html (visited on 11/12/2021).

[29] X.-D. Zhang, 'Machine learning,' in *A Matrix Algebra Approach to Artificial Intelligence*, X.-D. Zhang, Ed., Singapore: Springer, 2020, pp. 223–440, ISBN: 9789811527708. DOI: 10.1007/978-981-15-2770-8_6. [Online]. Available: https://doi.org/10.1007/978-981-15-2770-8_6 (visited on 09/12/2021).

[30] E. Rich, *Artificial Intelligence (International Student Edition)*. McGraw-Hill, 1st Jan. 1984, vol. 11, 436 pp.

[31] '2.7. novelty and outlier detection,' scikit-learn. (), [Online]. Available: Giannakopoulos (visited on 18/12/2021).

[32] P. Date, 'Combinatorial neural network training algorithm for neuromorphic computing,' Ph.D. dissertation, 16th Dec. 2019. DOI: 10.13140/RG.2.2.27337.90726.

[33] P. Riley, 'Three pitfalls to avoid in machine learning,' *Nature*, vol. 572, no. 7767, pp. 27–29, Aug. 2019, Bandiera_abtest: a Cg_type: Comment Number: 7767 Publisher: Nature Publishing Group Subject_term: Mathematics and computing, Software, Research data. DOI: 10.1038/d41586-019-02307-y. [Online]. Available: https://www.nature.com/articles/d41586-019-02307-y (visited on 12/05/2022).

[34] X. Ying, 'An overview of overfitting and its solutions,' *Journal of Physics: Conference Series*, vol. 1168, p. 022 022, Feb. 2019, ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1168/2/022022. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022 (visited on 06/06/2022).

[35] 'What is overfitting?' (6th Mar. 2021), [Online]. Available: https://www.ibm.com/cloud/learn/overfitting (visited on 27/05/2022).

[36] M. A. Lones, 'How to avoid machine learning pitfalls: A guide for academic researchers,' *arXiv:2108.02497 [cs]*, 5th Aug. 2021. arXiv: 2108.02497. [Online]. Available: http://arxiv.org/abs/2108.02497 (visited on 12/05/2022).

[37] L. Rokach, *Ensemble Learning: Pattern Classification Using Ensemble Methods*, 2nd ed., ser. Series in Machine Perception and Artificial Intelligence. WORLD SCIENTIFIC, Mar. 2019, vol. 85, ISBN: 9789811201950 9789811201967. DOI: 10.1142/11325. [Online]. Available: https://www.worldscientific.com/worldscibooks/10.1142/11325 (visited on 20/05/2022).

[38] Gilgoldm, *English: A tree showing survival of passengers on the titanic ("sibsp" is the number of spouses or siblings aboard).* 18th May 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Decision_Tree_-_survival_of_passengers_on_the_Titanic.jpg (visited on 20/05/2022).

[39] 'What is a random forest?' TIBCO Software. (), [Online]. Available: https://www.tibco.com/reference-center/what-is-a-random-forest (visited on 20/05/2022).

[40] C. Zhang and Y. Ma, Eds., *Ensemble Machine Learning*, Boston, MA: Springer US, 2012, ISBN: 978-1-4419-9325-0 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-9326-7 (visited on 20/05/2022).

[41] A. Patle and D. S. Chouhan, 'SVM kernel functions for classification,' in *2013 International Conference on Advances in Technology and Engineering (ICATE)*, Jan. 2013, pp. 1–9. DOI: 10.1109/ICAdTE.2013.6524743.

[42] Y. Xiao, H. Wang, W. Xu and J. Zhou, 'Robust one-class SVM for fault detection,' *Chemometrics and Intelligent Laboratory Systems*, vol. 151, pp. 15–25, Feb. 2016, ISSN: 01697439. DOI: 10.1016/j.chemolab.2015.11.010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169743915003056 (visited on 06/06/2022).

[43] V. Pattanashetti. 'Classical ML & its algorithms,' Medium. (18th Jun. 2020), [Online]. Available: https://medium.com/@_vp/classical-ml-its-algorithms-9a47c9d65ee0 (visited on 20/12/2021).

[44] A. Jain. 'Support vector machine(s.v.m) — classifiers and kernels.' (25th Sep. 2020), [Online]. Available: https://medium.com/@apurvjain37/support-vector-machine-s-v-m-classifiers-and-kernels-9e13176c9396 (visited on 20/12/2021).

[45] S. Hariri, M. C. Kind and R. J. Brunner, 'Extended isolation forest,' *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1479–1489, Apr. 2021, Conference Name: IEEE Transactions on Knowledge and Data Engineering, ISSN: 1558-2191. DOI: 10.1109/TKDE.2019.2947676.

[46] 'Isolation forest: Learned iForest construction for toy dataset,' ResearchGate. (), [Online]. Available: https://www.researchgate.net/figure/Isolation-Forest-learned-iForest-construction-for-toy-dataset_fig1_352017898 (visited on 06/06/2022).

[47] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, 'LOF: Identifying density-based local outliers,' p. 12,

[48] P. Probst, M. N. Wright and A.-L. Boulesteix, 'Hyperparameters and tuning strategies for random forest,' *WIREs Data Mining and Knowledge Discovery*, vol. 9, no. 3, e1301, 2019, ISSN: 1942-4795. DOI: 10.1002/widm.1301. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1301 (visited on 12/06/2022).

[49] S. Saxena. 'Random forest hyperparameter tuning in python | machine learning,' Analytics Vidhya. (12th Mar. 2020), [Online]. Available: https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/ (visited on 27/05/2022).

[50] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization,' *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html (visited on 27/05/2022).

[51] R. E. Berg. 'Sound - sound absorption | britannica.' (9th Jun. 2006), [Online]. Available: https://www.britannica.com/science/sound-physics/Sound-absorption (visited on 10/06/2022).

[52] M. Rahimi, A. Alghassi, M. Ahsan and J. Haider, 'Deep learning model for industrial leakage detection using acoustic emission signal,' *Informatics*, vol. 7, no. 4, p. 49, Dec. 2020, ISSN: 2227-9709. DOI: 10.3390/informatics7040049. [Online]. Available: https://www.mdpi.com/2227-9709/7/4/49 (visited on 09/03/2022).

[53] M. R. Azim, S. A. Haque, M. S. Amin and T. Latif, 'Analysis of EEG and EMG signals for detection of sleep disordered breathing events,' in *International Conference on Electrical Computer Engineering (ICECE 2010)*, Dec. 2010, pp. 646–649. DOI: 10.1109/ICELCE.2010.5700776.

[54] R. Vallat. 'Bandpower of an EEG signal.' (May 2018), [Online]. Available: https://raphaelvallat.com/bandpower.html (visited on 31/03/2022).

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, 'Scikit-learn: Machine learning in python,' *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html (visited on 13/06/2022).

[56] N. Seliya, A. Abdollah Zadeh and T. M. Khoshgoftaar, 'A literature review on one-class classification and its potential applications in big data,' *Journal of Big Data*, vol. 8, no. 1, p. 122, 10th Sep. 2021, ISSN: 2196-1115. DOI: 10.1186/s40537-021-00514-x. [Online]. Available: https://doi.org/10.1186/s40537-021-00514-x (visited on 12/06/2022).

[57] H. Inoue, 'Data augmentation by pairing samples for images classification,' arXiv, arXiv:1801.02929, 11th Apr. 2018, type: article. arXiv: `1801.02929[cs, stat]`. [Online]. Available: `http://arxiv.org/abs/1801.02929` (visited on 12/06/2022).

Appendix A

Code

This appendix contains code discussed in the thesis deemed unsuited to include in the text due to length. The complete code can be found on GitHub: `https://github.com/simenb-h/gas-and-anomaly-detectors`. Open the files to see already executed code or follow the instructions in the *readme* to run the code yourself.

A.1 File Information Retrieval

```
import os
import csv
import glob

from google.colab import drive
drive.mount('/content/gdrive',force_remount=1)
path_to_audio_files = '/content/gdrive/MyDrive/Masteroppgave/IDMT_dataset/'

def save_paths(base_dir, file_end, save_file_name="wav_paths.csv"):
  wav_files = []
  for root, dirs, files in os.walk(base_dir):
      for file in files:
          if file.endswith(str(file_end)):
            path = os.path.join(root, file)
            leak_present = 1 if "_niO_" in file else 0
            file_split = file.split("_")
            recording = file_split[0]
            knob_rotations = file_split[2]
            mic = file_split[3]
            root_split = root.split(base_dir)[1].split("/")
            leak_type = root_split[0]
            environment = root_split[1]
            environment_folder = root_split[2]

            wav_files.append([path,leak_type,environment,recording,mic,
    knob_rotations,leak_present])
            #print([path,leak_type,environment,recording,mic,knob_rotations,
    leak_present])


  if ".csv" not in save_file_name:
    save_file_name += ".csv"

  with open(save_file_name,'w') as result_file:
      wr = csv.writer(result_file, dialect='excel')
      wr.writerow(["path","leak_type","environment","recording","mic","
    knob_rotations","leak_present"])
      for wav_file in wav_files:
        wr.writerow(wav_file)
```

```
save_paths("/content/gdrive/MyDrive/Masteroppgave/IDMT_dataset/", ".wav", "
    wav_paths_master.csv")
```

A.2 Relative power in segments code

```python
import pandas as pd
import numpy as np
from scipy.signal import welch
from scipy.integrate import simps

def relative_power_segments(data, sr, segment_list, window_sec, relative=False):
    """Parameters
    ----------
    data : 1d-array
        Input signal in the time-domain.
    sr : float
        Sampling frequency of the data.
    segment_list : list
        List with frequencies to begin new segment.
    window_sec : float
        Length of each window in seconds.

    Return
    ------
    bp : DataFrame
        Relative segment power for len(segment_list)-1 segments.
    """

    # Compute the modified periodogram (Welch)
    freqs, psd = welch(data, sr, nperseg=nperseg)

    avg_segment_power = []

    #Iterating over the segemnts from segment-list
    for i in range(len(segment_list)-1):
      freq_res = freqs[1] - freqs[0]
      low = segment_list[i]
      high = segment_list[i+1]

      # Find closest indices of segment in frequency vector
      idx_segment = np.logical_and(freqs >= low, freqs <= high)

      # Integral approximation of the spectrum using Simpson's rule.
      bp = simps(psd[idx_segment], dx=freq_res)

      #Dividing the segment-power of each bin with the total power
      if relative:
          bp /= simps(psd, dx=freq_res)

      #Storing the results to a list
      avg_segment_power.append(bp)

    return pd.DataFrame([avg_segment_power])
```

A.3 Experiment execution

The following method returns a dictionary with all the models, that will be used.

```
def ml_models():

    dt = DecisionTreeClassifier(random_state=0)
    rf = RandomForestClassifier(random_state=0)
    xgb_model = xgb.XGBClassifier(objective="binary:logistic",  random_state=0)
    ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
            algorithm="SAMME", random_state=0)

    models = {
        "dt" : dt,
        "dt_1" : dt_1,
        "rf" : rf,
        "ada": ada,
        "xgb" : xgb_model,
    }
    return models
```

The following is the code that is used to executa all of the experiments. All

```
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import accuracy_score

#Setting up StratifiedGroupKFold for LOO
cv = StratifiedGroupKFold(n_splits=3) #Using CV with 3 splits
env_groups = X.recording.values # and recording as group

#Splitting train and test data, only needed for CV
X = df_train_features
y = X.leak_present.values

#Calling the ml_models() method to get models
models = ml_models()

#Dictionary to store all the results
all_results = {}

#Iterating over the models and training and testing them
for model in models :
  model_results = {}
  #Iterating over all the train and test set in 3 splits
  for train_idxs, test_idxs in cv.split(X ,y ,env_groups):

    #Dividing the data into train and test, with labels.
    X_train, y_train, X_test, y_test = get_data_labels(X.iloc[train_idxs], X.iloc[
     test_idxs])

    #Code for selecting subset of features
    X_train = X_train[['spec_bandwidth','spec_flatness', 'mfcc_bin: 5', 'zcr', '
     p_bin: 20-24kHz']] #Notation for selecting subset of features
    X_test =X_test[['spec_bandwidth','spec_flatness', 'mfcc_bin: 5', 'zcr', 'p_bin:
      20-24kHz']]

    #Training models on train data of the current fold
    models[model].fit(X_train,y_train)
    score = models[model].score(X_test , y_test)
    predictions = models[model].predict(X_test)

    #Combining the predictions with the Test data
    pred_col = pd.DataFrame(predictions)
    pred_col.columns = ["predictions"]
    pred_col.reset_index(drop=True, inplace=True)
    X_test_info.reset_index(drop=True, inplace=True)
    test_preds = pd.concat([X_test_info, pred_col["predictions"]], axis=1)

    #Iterating over leaks and environments in the test set to get predictions
     category wise
    for leak in X_test_info_preds.leak_type.unique():
      for env in X_test_info_preds.environment.unique():
        preds = test_preds.loc[(test_preds.leak_type == leak) & (test_preds.
     environment == env)]["predictions"]
        y_test = test_preds.loc[(test_preds.leak_type == leak) & (test_preds.
     environment == env)]["leak_present"]



        if len(predictions) != 0:
          f1 = f1_score(predictions, y_test)
          acc = accuracy_score(predictions, y_test)
          metric = acc #Selecting what metric to use

        else:
          metric = 0

        #Adding the result to a dictionary that stores the models results
        key = str(leak + "-" + env)
        if key in model_results:
          model_results[key].append(metric)
        else:
          model_results[key] = [metric]

  #Iterating over the results for each fold and averaging it
  average_results = {}
  for key, vals in model_results.items():
    if (0.0 in vals) or (0 in vals):
      vals.remove(0)
      average_results[key] = [(np.mean(vals)*100), 100*statistics.pstdev(vals), "*"
     ]
    else:
      average_results[key] = [(np.mean(vals)*100), 100*statistics.pstdev(vals)]

  #Storing the average results for each model
  all_results[model] = average_results

#printing the results for all the models
all_results
```

Appendix B

Hyperparameters

B.1 Tree Based Classification Models

```
Decision Tree:
{'ccp_alpha': 0.0
'class_weight': None
'criterion': 'gini'
'max_depth': None
'max_features': None
'max_leaf_nodes': None
'min_impurity_decrease': 0.0
'min_samples_leaf': 1
'min_samples_split': 2
'min_weight_fraction_leaf': 0.0
'random_state': 0
'splitter': 'best'}
```

```
Random Forest:
{'bootstrap': True
'ccp_alpha': 0.0
'class_weight': None
'criterion': 'gini'
'max_depth': None
'max_features': 'auto'
'max_leaf_nodes': None
'max_samples': None
'min_impurity_decrease': 0.0
'min_samples_leaf': 1
'min_samples_split': 2
'min_weight_fraction_leaf': 0.0
'n_estimators': 100
'n_jobs': None
'oob_score': False
'random_state': 0
'verbose': 0
'warm_start': False}
```

```
AdaBoost:
{'algorithm': 'SAMME'
'base_estimator__ccp_alpha': 0.0
'base_estimator__class_weight': None
'base_estimator__criterion': 'gini'
'base_estimator__max_depth': 1
'base_estimator__max_features': None
'base_estimator__max_leaf_nodes': None
'base_estimator__min_impurity_decrease': 0.0
'base_estimator__min_samples_leaf': 1
'base_estimator__min_samples_split': 2
'base_estimator__min_weight_fraction_leaf': 0.0
'base_estimator__random_state': None
'base_estimator__splitter': 'best'
'base_estimator': DecisionTreeClassifier(max_depth=1)
'learning_rate': 1.0
'n_estimators': 50
'random_state': 0}
```

```
XgBoost
{'base_score': 0.5
'booster': 'gbtree'
'colsample_bylevel': 1
'colsample_bynode': 1
'colsample_bytree': 1
'gamma': 0
'learning_rate': 0.1
'max_delta_step': 0
'max_depth': 3
'min_child_weight': 1
'missing': None
'n_estimators': 100
'n_jobs': 1
'nthread': None
'objective': 'binary:logistic'
'random_state': 0
'reg_alpha': 0
'reg_lambda': 1
'scale_pos_weight': 1
'seed': None
'silent': None
'subsample': 1
'verbosity': 1}
```

B.2 One Class Classifying Models

```
One Class SVM
{'cache_size': 200
'coef0': 0.0
'degree': 3
'gamma': 'scale'
'kernel': 'rbf'
'max_iter': -1
'nu': 0.1
'shrinking': True
'tol': 0.001
'verbose': False}
```

```
Local Outlier Factor:
{'algorithm': 'auto'
'contamination': 'auto'
'leaf_size': 30
'metric': 'minkowski'
'metric_params': None
'n_jobs': None
'n_neighbors': 20
'novelty': True
'p': 2}
```

```
Isolation Forest:
{'bootstrap': False
'contamination': 'auto'
'max_features': 1.0
'max_samples': 'auto'
'n_estimators': 100
'n_jobs': None
'random_state': None
'verbose': 0
'warm_start': False}
```