# Audio Lab Report:
# Anomaly Detection for Acoustic Sensor Data

Johannes Schmidt

August 16, 2022

**Abstract**

This lab experiments with methods for anomaly detection for acoustic sensor data. There Methods are used as a first filtering step for later using machine learning techniques to predict flow rates of liquids in sewer pipes. In this work an approach of preparing and labeling the data is presented and three experiments for anomaly detection are conducted and the experimental results are discussed. The three experiments aim to determine overall noise, catch sudden anomalous events and computing an anomaly score. While determining the noise worked out well, the other two approaches achieve lower accuracy.

## 1 Introduction

In this lab the task was given by a company who manages sewer pipes and fresh water supply for a German city. Their goal is to use audio from microphones installed in their sewer pipes to determine the amount of liquid flowing through the pipes. At the point this report is written it cannot be excluded that this is even possible to a sufficient precision. Besides saving money (compared to more expensive sensors) another motivation is to improve the monitoring to detect flood events, which became of greater interest in NRW since last year.

When working with measurements made in realistic environments, in contrast to measurements made in the lab, one always has to encounter problems such as loss of connection/data, clipping, hardware problems or wrong calibration. Especially for the case of providing data to neural networks to make predictions one has to make sure that the data has the right representation and scaling. Therefore, the objective of this work is to apply so-called anomaly detection (AD) on the dataset to design a system with a reasonable trade-off between robustness and flexibility. Robustness is important so the filtering is reliable and flexibility is important, because one cannot forecast all types of interfering sounds that can occur. So the hope is that once the behaviour of 'normal'/undisturbed samples is captured, everything that differs of a sufficient degree from that can be detected as an anomaly.
This is of course no clean definition for the term *anomaly*, therefore the data labeling and each experiment done assumes something different of an anomaly. To make sure that there is no confusion by this term, it is discussed and specified before its use.

The rest of this document will be structured as follows. In Section 2 includes a short review on the theoretical background with references about techniques that are used later in the experiments. In Section 3 the data set is presented and some specific data samples are shown. Also it is explained which observations have driven the label and data structure design. Section 4 presents the conducted experiments and their results. A Conclusion of the results with discussion on them is shown in Section 5. Section 6 finished this report with some outlook on further investigation that can be done to improve upon the obtained results.

## 2 Methods used

The experiments were done in python jupyter notebooks[1] in a virtual coding environment. Mainly the packages numpy[2], pandas[3], Scipy[4] were used for processing the data, like computing Fourier

transforms and spectrum's. Machine learning was done with sklearn/scikit-learn[5], deep learning with tensorflow[6] and visualizations were made with matplotlib[7] and seaborn[8].
For this report a basic understanding of methods and concepts used by such libraries, especially the theory behind the applied signal processing tools as covered in the lectures MA-INF 2113 - Foundations of Audio Signal Processing and MA-INF 2212 - Pattern Matching and Machine Learning for Audio Signal Processing by Prof. Dr. Frank Kurth, will be assumed and not further explained.

The next subsections go into a bit more detail for some specific methods used in the experiments. Those explanations are motivated to give more insights about why certain decisions were made during the process of experimenting.

**Random Forest**

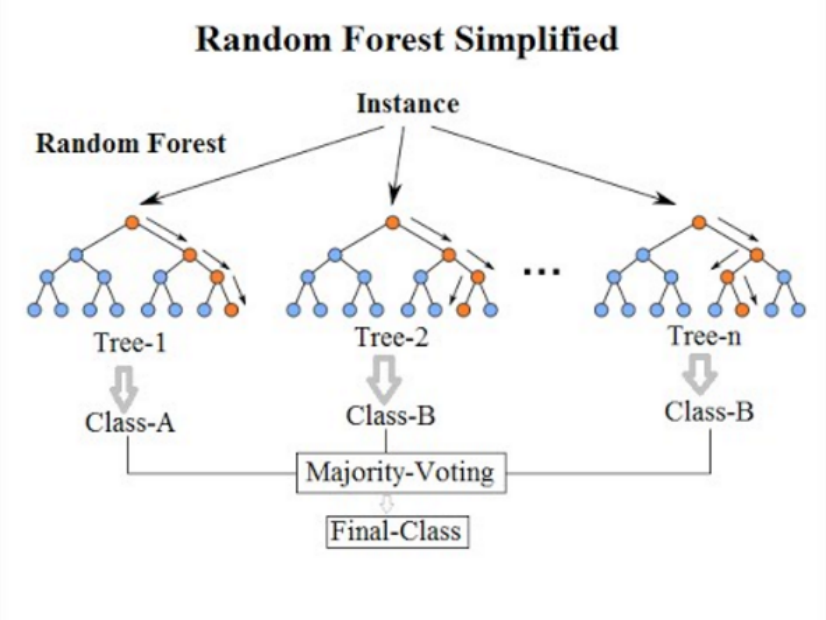


Figure 1: Basic schematic of a random forest (Source here:[9])

A random forest is an ensemble of decision trees[10]. For a visualization see Figure 1. Since decision trees do their reasoning in a human understandable way, they are well suited for the a first investigation of the data. The most important hyperparameters of a random forest are the maximum depth of the individual decision trees and the number of trees used.

### Support Vector Classifier (SVC)

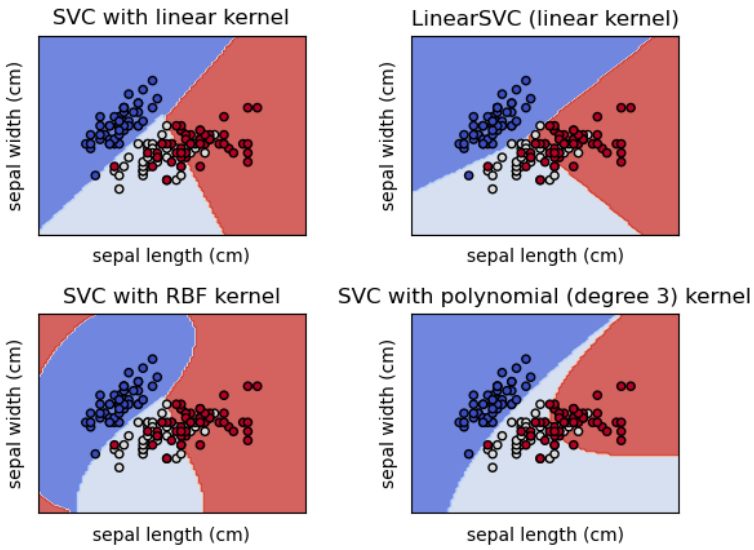


Figure 2: Example image for SVC. Source: sklearn[5]. The shape of the classified regions highly depends on the kernel and therefore has to be chosen per task.

A SVC (not to be confused with support vector clustering) is a Support Vector Machine (SVM) that is used for classification. Roughly speaking a SVM tries to separate data of two classes in a supervised fashion via fitting a so-called maximum-margin hyperplane. The space where this hyperplane lives in depends on the kernel chosen. The visual difference between the boundaries chosen by the SVC are depicted in Figure 2. Other hyperparameters are the by default used regularization and parameters included in the kernels, for example the width of the Gaussian distribution for rbf kernels (often denoted as gamma)[5].

### One-Class SVM / Novelty Detection (ND)

This method is a semi-supervised one called One-Class Support Vector Machine[11]. The high level idea is to abstract a distribution over a dataset and let the network decide if a *new* observation belongs to that distribution. For a visual representation of this scenario, see Figure 3. One has to make sure that in the initially shown dataset there are no outliers, thus semi-supervised. In the scikit-learn documentation the task of assigning data to distributions is divided into two different subtasks: Outlier and Novelty detection. In our case we are interested in novelty detection, because the algorithm should be used as an online method, judging whether a sample is normal or anomalous only for new incoming data. Note, however, that outlier detection methods can also be used for novelty detection with some slight modifications. This is because the difference between outlier and novelty detection is only the way it is trained. For outlier detection the training data contains both anomalous and normal samples, where as for novelty detection only normal samples are contained.

The hyperparameters here are the kernel and gamma again, but this time also the so called *nu* factor (scikit-learn terminology again). It is an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. In practice this parameter highly effects the accuracy and therefore has to be chosen very carefully.

### Autoencoder (AE)

Autoencoders are core archtectures used in many state of the art deep learning approaches tackling many problems, where anomaly detection is one of them. Therefore, there is a lot that can be said

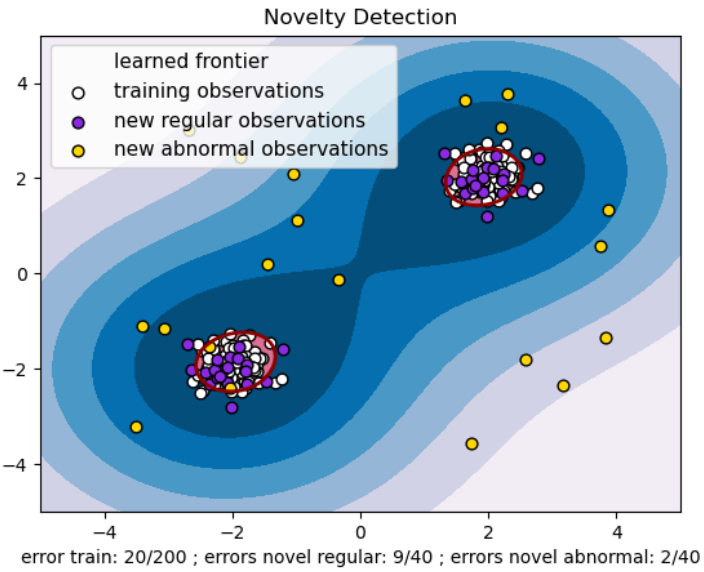


Figure 3: Example image for Novelty Detection. Source: sklearn[5]

about autoencoders. Most of the background will not be relevant in this work, but can be looked up here [12].
The basic idea of an autoencoder is that it tries to efficiently encode and decode data. The efficiency is given by a bottleneck layer in the middle of the network. This representation is trained in an unsupervised way via trying to reproduce the input image. To prevent just learning the identity function different tricks and techniques need to be used such as using a much smaller dimension for the code than the input dimension.

#### Note on using these methods for anomaly detection

Strictly speaking the supervised methods do not implement anomaly detection the way it is usually done in the literature. Here we decide beforehand what kind of anomaly we search for. In the later experiments it will be white noise. The unsupervised methods are the more common approach. There everything that deviates from normal samples is recognised as an anomaly.

## 3 The data and its preparation

#### What was given at the beginning:

The dataset is some data collected on 4 different days. Every minute 10 seconds of audio were captured and additionally (for some of these days) the flow rate every five minutes were measured. The file systems structure is depicted in figure 4.
Per folder there are up to 420 .wav files with a sampling rate of 48 kHz. In total there are 1169 .wav with a total size of about 1 Gb.

#### First observations in the data:

To get a first impression of the data, a random subset of them was inspected. For this subset the spectra was being looked at and the corresponding audio listened to. Some carefully chosen spectra are depicted in Figure 5. The first thing that attracts attention is that some audio files are corrupted by white noise. Either short term interruptions or for the entire ten second period. Besides these pure white noise sounds there are also very short transients in the signal. There also is a varying amount of these peaks. Some files contain only few of them and for some files it is hard to distinguish them from pure white noise. Note that in the case of a few of these short peaks their presence is much easier

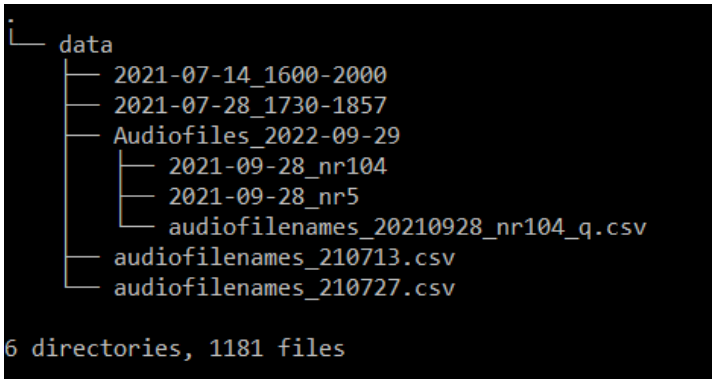


Figure 4: structure of the sent data

realized from the visualized spectrogram than from the actual audio. Note that the peaks and white noise events (no matter if short term or constant) were very clearly detected visually by the fact that much energy of the signal is contained in frequencies above 20 kHz. This suggests that these events are not directly related to the sounds transmitted by the liquid flow, because in normal recordings no energy is contained above 20 kHz. However, it might be related indirectly by the cause of clipping during the recording. That way one could use the occurrence of these distortions as a sign of high flow rate. This would however be very dependent on the microphone setup, both for the configurations as well as location in the sewer pipe and it would also change from pipe to pipe[1].

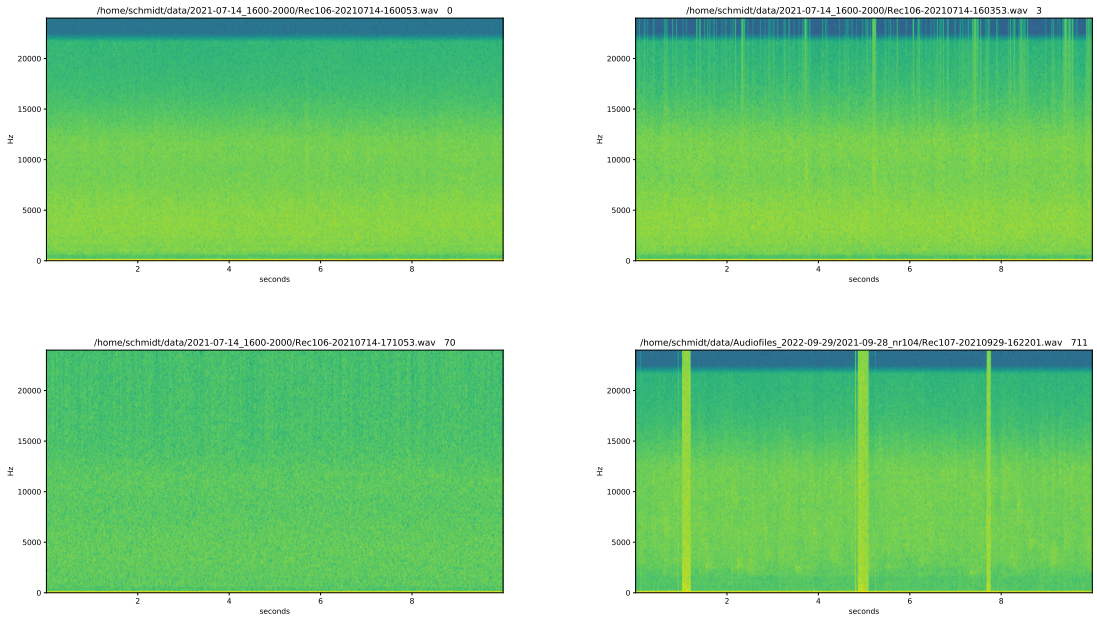


Figure 5: Some images

Besides the white noise distortions there were other sort term events in the data. Some of these events are drip sounds, car wheels, car breaks and even a siren. This observation suggests, that the microphones were placed in sewer pipes next to or under a street. Most of the samples though can be considered to stay constant during the entire 10 seconds.

[1] Guesswork like this appears form time to time in this report, because the task of filtering observations for a purpose of course dependents on the purpose. Since it was not clearly given what changes can be made in the experimental setup later as well as what exact parts in the audio files will be relevant for the feature extraction and ultimately for the flow rate prediction, some assumptions had to be made in order to be able to stay clear of the objective. These assumptions shaped the way an anomaly was defined here.

There were also some more subtle artifacts in the recordings like a constant received signal bellow 20 Hz or for loud short term events a compression was applied with a release time that is in the order of half a second. Such details however had been ignored for the investigations.
Another observation was that the drip sound were almost entirely covered by the recordings of one particular folder. One might argue that this is due to some specific pipe and therefore exhibits a different characteristic sound in general instead of a 'rare' occurring event in that case. This also makes it harder to work precisely here, since is is not known if a model will be trained per sewer pipe to compensate for such characteristic ambient acoustic or a one model fits all solution is desired. For simplicity reasons drip is a short term event here.

**Labeling the data for the purpose of AD:**

From the made observations the following definition for an anomaly seemed reasonable:

**Every sound in the recording that had not been emitted by the liquids themselves is unrelated to its flow and therefore understood as an anomaly.**

According to this definition the parts of the signal that have an equal amount of energy in all frequencies (white noise) are considered as anomalous. Still it should be differentiated between different amounts of overall noise in the data and short term occurrences of white noise or other sounds.
The amount of noise present in the entire ten seconds is captured by a quantity named noise level. This is an perceptual measure that roughly gives an idea of how noisy the recording is. It contains of the integers 0 to 3 with the following meaning:

0. clean signal, no white noise contained, no peaks above 20 kHz
1. Some visible transients with peaks above 20 kHz contained, but no audible distortions
2. much visible and audible noise like distortion
3. visually and acoustically hardly distinguishable from synthetic white noise

The short term events however were additionally captured by a list of variable size for each recording. Every time an anomaly as defined above was noticed a note containing the information of how it sounded like (in one single word) had been added to that list. These lists ended up containing the following labels:

```
ad_labels = ['wn_hard','wn_interrupt', 'cracks' ,'drip','break','siren','clean']
```

Note that the ad label and noise level quantities aren't redundant in the sense that, if the noise level is 3 no wn_interrupt will be added since it represents only the short term events. For a recording that is clean the entire time but has a short term white noise event, it would be labeled as noise level 0 with the additional ad label list ['wn_interrupt']. Also note that this does not capture when these short term events occurred.
The process of labeling was done entirely by hand. For that a jupyter notebook was build that depicts a recording with playback and two slots for entering the label. After the labels were added the next recording was shown. This way the process was still tedious, but finalised quickly.
Since there was no obvious reason why the date should be loaded entirely into memory (which shouldn't cause much problem for 1 Gb though) a simple pandas DataFrame was chosen to capture the data with labels. For a snippet of it see figure 6.
Note that the DataFrame does not contain any audio files. It only contains its name and location in the original file structure that was given in the beginning.

Figure 7 roughly depicts the results of that labeling.
Most of data is clean and for the noise data there is more of the heavily distorted one. The interrupts are almost entirely for the clean data (not visible in the plot) and the other more special labels like break (7) and siren (1) are very rare so not significant enough for this visualization.

| | file_location | file_names | ad_labels_list | noise_level |
|---|---|---|---|---|
| **0** | 2021-07-14_1600-2000/ | Rec106-20210714-160053.wav | [clean] | 0.0 |
| **1** | 2021-07-14_1600-2000/ | Rec106-20210714-160153.wav | [clean] | 0.0 |
| **2** | 2021-07-14_1600-2000/ | Rec106-20210714-160253.wav | [clean] | 0.0 |
| **3** | 2021-07-14_1600-2000/ | Rec106-20210714-160353.wav | [clean] | 0.0 |
| **4** | 2021-07-14_1600-2000/ | Rec106-20210714-160453.wav | [cracks] | 1.0 |
| **...** | ... | ... | ... | ... |
| **1164** | Audiofiles_2022-09-29/2021-09-28_nr5/ | Rec104-20210929-165557.wav | [clean, drip] | 0.0 |
| **1165** | Audiofiles_2022-09-29/2021-09-28_nr5/ | Rec104-20210929-165657.wav | [clean, drip] | 0.0 |
| **1166** | Audiofiles_2022-09-29/2021-09-28_nr5/ | Rec104-20210929-165757.wav | [clean, drip] | 0.0 |
| **1167** | Audiofiles_2022-09-29/2021-09-28_nr5/ | Rec104-20210929-165857.wav | [clean, drip] | 0.0 |
| **1168** | Audiofiles_2022-09-29/2021-09-28_nr5/ | Rec104-20210929-165957.wav | [clean, drip] | 0.0 |

1169 rows × 4 columns

Figure 6: Snippet of the DataFrame containing all information needed for working with the dataset

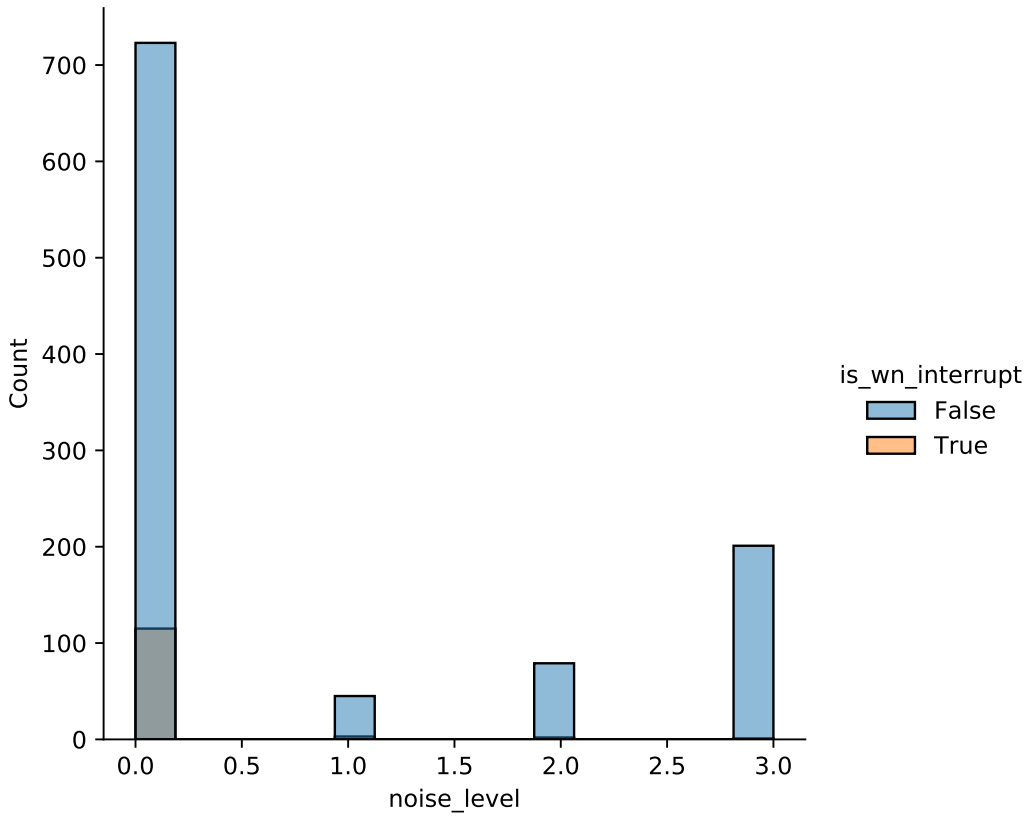


Figure 7: Histogram for the ad labels

# 4 Experiments

The task of AD can be done in multiple ways. In this work two main approaches were applied and compared to each other. An unsupervised one-class SVM and a deep method particularly designed to perform anomaly detection. The deep method was decided to be an autoencoder. The methods for prototyping though (random forest and SVC) were supervised. These two approaches differ in a very fundamental way.
For the unsupervised method the motivation is to let the model decide on features that capture the distribution of the data and detect deviations from this learned distribution. So technically speaking what is done here is not pure unsupervised training, but semi-supervised. We are not interested in revealing general cluster structure, but in the training phase show the network only samples that were classified by a human as 'normal', so no anomaly contained. In our case this would be a recording with noise level 0 and 'clean' as the only ad label.
Every point (not just cluster) deviating from this distribution should be detectable as something different than the 'normal' seen before. So in that case we somewhat get around of clearly defining what an anomaly is to us. In the supervised case this (by design) is not possible. There one has to have a clear labeling of what class a sample belongs to.

To sum this up, the supervised methods initially tried to perform a classification of a predefined label, whereas for the semi-supervised methods the hope is that they generalize the specific phenomenon occurring in the data to somewhat take 'care of the unexpected'.

## 4.1 Detecting noise

**Preparing the data for training**

The first experiment naturally arises from the noise level labels.

*How well can we train a model to reason about white noise-like noise contained in a ten second recording of a sewer pipe?*

This would allow to filter out recordings where it is not possible to extract any sound emitted by the liquid flow.
To make the model fit better in the framework of AD the output is not noise level itself, but 'is the data clean or is it not'.

$$\begin{gathered}\text{noise level} \rightarrow \text{class label} \\ \text{s.t.} \\ \{0\} \mapsto 0 \\ \{1,2,3\} \mapsto 1\end{gathered}$$

This way the task is modeled as binary classification. For this experiment all recordings containing disturbing short term events were not included in the training and testing.
First a design choice had to be made concerning the representation of the data. Since all noticeable sudden events had been thrown out of the data set for now, the spectrum stayed almost perfectly constant the entire 10 seconds. This was also justified by comparing the Fourier transform (FT) of the 10 seconds with the FT of an arbitrary subpart of it.
This motivated to just use the Fourier transform of the entire ten seconds signal in addition to a max compression, since a rough coverage of the curve should be enough for detecting noise. The max compression just took the maximal value every 5000 points. So similar to down sampling it is equidistant, but it is not taking every 5000th value. An example for a clean recording is shwon in Figure 8.
This max compression plus normalization is applied to the entire dataset. For the normalization also a max compression had been chosen, because in this case it puts higher emphasis on the high frequency end. This normalization takes the max value of a dimension and scales the axis such that this value becomes one. This is performed for all the dimensions. Note that the normalization has a huge impact on the accuracy of the model. There will be more about it later.

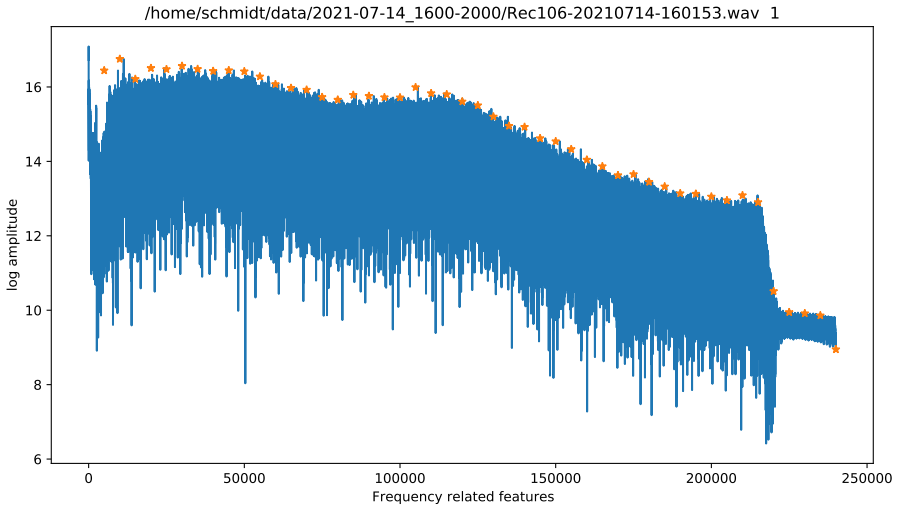


Figure 8: Blue curve: Fourier transform of a clean sample. Orange points: maximum of every 5000 point chunk

### Deciding for the models and the training setup

The methods chosen were random forest, SVC and the one class SVM (here also often called ND, short for novelty detection). For all methods the train test split ratio was 80 : 20. The training algorithm was the default choice of sklearn which is performed using the *.fit()* module and for the performance measure, the accuracy was defined in the following way

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \mathbf{1}(\hat{y}_i = y_i)$$

with $y$ being the prediction of the network on the entire dataset, $\hat{y}$ the according ground truth, $n_{\text{sampes}}$ the number of samples and the identity function $\mathbf{1}(\cdot)$ results to 1 when the argument is true and 0 if false. This measure was chosen, because it is easily implemented, interpreted and easy to debug. Note that this performance measure does not take precision and recall into account, i.e. depending on the balance of true and false samples in data learning a constant function might achieve a high accuracy. This issue was addressed by checking it manually afterwards. Also Figure 7 gives an understanding that the balance between zeros and ones in the labels should be balanced well enough. Now to the individual methods.

A **random forest** with 100 decision trees, no limits for the max depth for each tree and Gini split as splitting criterion was trained initially. The out of the box performance of this method should give a first impression of what accuracy should be also possible with the other more sophisticated methods. The first shot already achieved an accuracy of about 96 %.

Since out of the box performance with default parameter settings of the ND was below 10 % even on the training set the next step was to train a supervised **SVC**. In order to get decent performance, optimizing the hyperparameters was necessary. For that the grid-search-cross-validation method with 5 folds was used. To be very sure of not learning constant functions here the F1-score was used to compare the performance of the networks. For the following set an accuracy of 97 % on the test set was reached.

```
{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
```

The SVC gave confidence and a direction of how to increase the performance of the semi-supervised **ND** approach. Here again the same grid search algorithm was used, however with other hyperparameters

(no 'C', but 'nu') and smaller step sizes. For this the training data had to be modified in a way that it only contains samples with label 0, because we want the network to learn its distribution. This also achieved an accuracy of 97 % for

```
{'kernel': 'linear', 'nu': 2/3}.
```

In this case there is no 'gamma', because the kernel was linear.

These results suggest that detecting noise can be easily performed, even with such a drastic reduction of information in the data, due to the max compression. The next subsection further starts investigating the sudden anomalies.

## 4.2 Detecting short term phenomena

Now that we can detect noise levels in a ten second recording the following question came up

*When we split the ten seconds into smaller intervals, can we use the previous models to detect if interrupting white noise events occurred?*

This not only would give the possibility to filter out these uninformative events, but would also allow to deviate from the ten second length, which seemed not to be a natural choice for this type of data in the first place.
Since the SVC served as a intermediate step between the supervised random forest and the semi-supervised novelty detection it will not be used in the further investigations. In terms of evaluating the performance it is not as easy as for the noise detection, due to the nature of the ad label lists in the data from the DataFrame. There it only states when some sudden event was present in the recording. Therefore one has to manually check, for example with a spectrum, if the detected noise aligns with all the occurrences.
The only automated way of checking the performance would be to check if for recordings with noise level 0 the algorithm would detect white noise in it or not. That way one loses some information. For example for a recording where there are many noise interruptions, this method would only value it if some white noise was detected, but not if all occurrences in time were captured correctly.

For the split a length of 0.2 seconds was chosen, which is still above the lower bound that would allow to still resolve 20 Hz, due to the Nyquist theorem. Also, as one would expect it, the (properly scaled) FT of those audio files didn't deviate from the original signal, when there are no sudden events.

So when we apply both the ND and the random forest on every 0.2 seconds to check if noise is contained to try to reproduce the 'wn_interrupt' ad label the following accuracy is reached:

| | accuracy |
|---|---|
| RF | 72 % |
| ND | 67 % |

One can clearly see, that for such short term events the accuracy drops significantly. After manually checking some examples to investigate into the reason of that drop the following tendency was observed. When the random forest was triggered it was very likely, that there is an interruption, however there were some examples where there was noise, but it didn't triggered the RF. For the novelty detection it was the other way around. It also got triggered by events that seemed unrelated to noise.
This raised the following question:

*Is the trained novelty detection able to generalize the type of distortion for flagging anomalies like the break, siren or drip sounds?*

Since for those labels there were only a few examples, it just was checked visually. It turned out that this generalization did not work for all of the anomalies.

To compare the so far made experiments with the next method that differs a lot, the next experiment will try to answer the same questions from a different perspective.

### 4.3 Deep approach

The implementation and design of the now presented approach is the baseline approach of the DCASE 2020 Challenge Task 2 - 'Unsupervised Detection of Anomalous Sounds for Machine Condition Monitoring' [13]. There the underlying question was similar from the scenario here in the sense that anomalous events were aimed to be detected from a default sound event that was considered to be 'normal'. In their case is is the sounds of machines.
Their approach differs fundamentally from the way the input data is represented and what the output stands for. All the experiments done until this point were designed to make a binary decision in the end. Is the element part of the normal class or anomaly class. The objective here is a different one. Here the so called anomaly score is computed using an autoencoder. This score should be larger when the recording seems to be more anomalous. This is therefore no longer a pure classification task. It could be reduced to one by just setting a hard limit as a desired threshold. Or it could also be used for a container-like labeling much like the noise level. For simplicity reasons the anomaly score is not further processed. Lets first talk about the data representation and architecture of the autoencoder. All the following design choices were taken from the mentioned baseline approach, so for more in depth explanation.

The data is the one captured by the DataFrame without any splitting, so again the entire ten seconds. For the representation a log-mel-spectrogram of the ten second recordings is computed. For that an analysis frame of 64 ms is used in combination with 128 log-mel energy bands. With a context window size $P = 2$ and the application of concatenation this results in a 640 dimensional according to $D = F \cdot (2P + 1)$, with $F$ being the number of mel-filters. To make it very clear, none of the above used techniques like max compression and max normalization were applied here.

The Autoencoder therefore has an input and output dimension of 640. Further it has nine hidden layers. A block of four dense layers with 128 units, batch normalization[14] and the ReLU activation function each, followed by the bottleneck layer of 8 units, also with batch normalization and ReLU, and finished by another block of four dense layers with same properties.
The idea behind using an AE for anomaly detection is the following. The AE is trained on 'normal' recordings, so it learns to reconstruct those log-mel-spectrograms. The idea is when there energy introduced in frequencies that was not observed in the training the AE is assumed to fail reconstructing the image, and the resulting output deviates more from the input than it would be the case for a normal recording. This situation suggests the following definition for the above mentioned anomaly score

$$\mathcal{A}_\theta(x) = \frac{1}{DT}\sum_{t=1}^{T}\|\psi_t - \mathrm{AE}_\theta(\psi_t)\|_2^2 \,.$$

Here T stands for the number of time-frames in the log-mel-spectrogram, $\psi_t$ is the concatenated acoustic feature, $\mathrm{AE}_\theta$ an autoencoder with parameters $\theta$ and $\|\cdot\|_2$ is the $\ell_2$ norm.

For training the following parameters were chosen: 100 epochs with data shuffling between epochs, batch size of 512 and adam[15] as optimizer with a learning rate of 0.001. The results of this experiment are depicted in Figure 9.
These are some of the observations made:

- There is a tendency that the recordings with higher noise level get highre anomaly scores, but they overlap a lot.
- Most of the recordings reaching anomaly scores over 20 are clean recordings with white noise interruptions, but there are also many interrupted ones reaching the lowest scores relative to the other observations.
- The one siren observation indeed got a high anomaly score, however since there is just one such observation this can possibly be a coincident.
- the break and drip observations seem to be ignored entirely by the network. For the break this could be due to the log in the spectrum. This could have compress significant peaks in the energy of a single frequency in a way that suppresses its uniqueness.

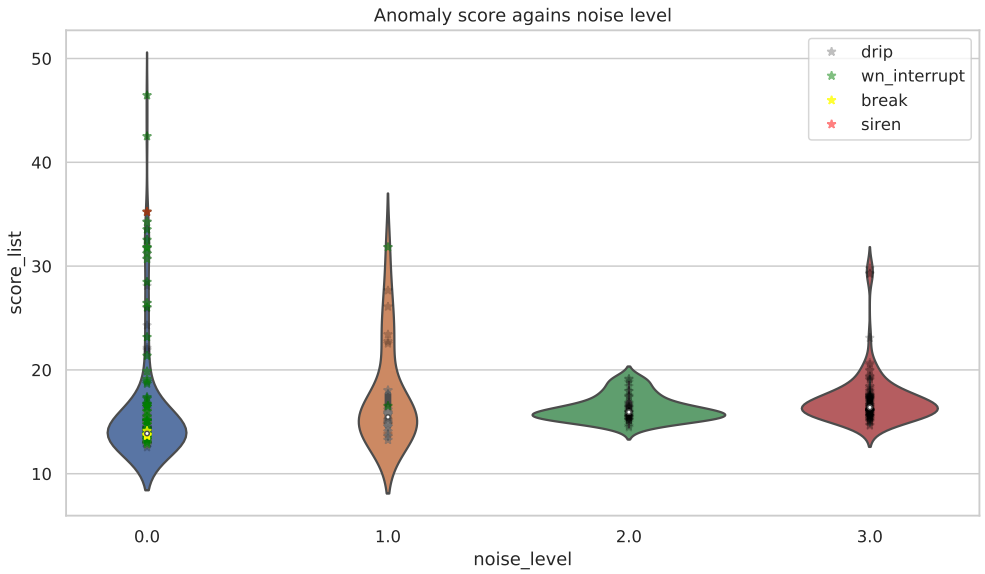


Figure 9: Violin plot depicting the computed anomaly score of certain recordings marked by noise level and

# 5 Conclusion and Discussion

Using machine learning techniques detecting white noise was easily done with an accuracy of at least 97 %. Though this performance dropped significantly when applying it on data with shorter time intervals of 0.2 $s$. There the performance and behavior for random forest and novelty detection was different (see the table of page 10). Also the hope of novelty detection generalizing to other kinds of anomalies was not fulfilled.
The attempt to achieve this generalization with an autoencoder based approach did not reach a precision useful for application.

The achieved precision is not sufficient to do the job for the application. The fact that the precise requirements for the flow rate prediction were not given, influenced the amount of attention that went into details. Investigating multiple questions and implementing multiple architectures for them probably resulted in this trade-off.
Since this work did not aim to optimize intensively on the chosen data representation or network architecture, there were many choices made out of intuition. Therefore there is a lot one could optimize further which would clearly improve the achieved performance. This will be discussed next.

# 6 Future Work/ Outlook

This last section lists some suggestions for deviating from the made decisions that might improve the reached performance. The list has no particular order.
Note that this also contains some more in depth discussion on topics not mentioned in the discussion above. This is because those details would have interrupted the reading flow unnecessarily without adding to the big picture. Therefore it appears in the outlook, although it gives another points of view on made decisions.

**max compression**

This way of representing the data was motivated by the visual impression that the individual points do not make up the characteristic curve of the sound, but only the envelope of it. This is of course no criterion that is based on signal processing concepts or any other convention. One could try many

things here: just down sampling, averaging or other types of compression. There is also no particular reason why the distance between compression points should be equidistant.
For example when applying principal component analysis (PCA) to reduce the dimension of the data set, which is basically data compression, one gets the result depicted in Figure 10. There the cumulative

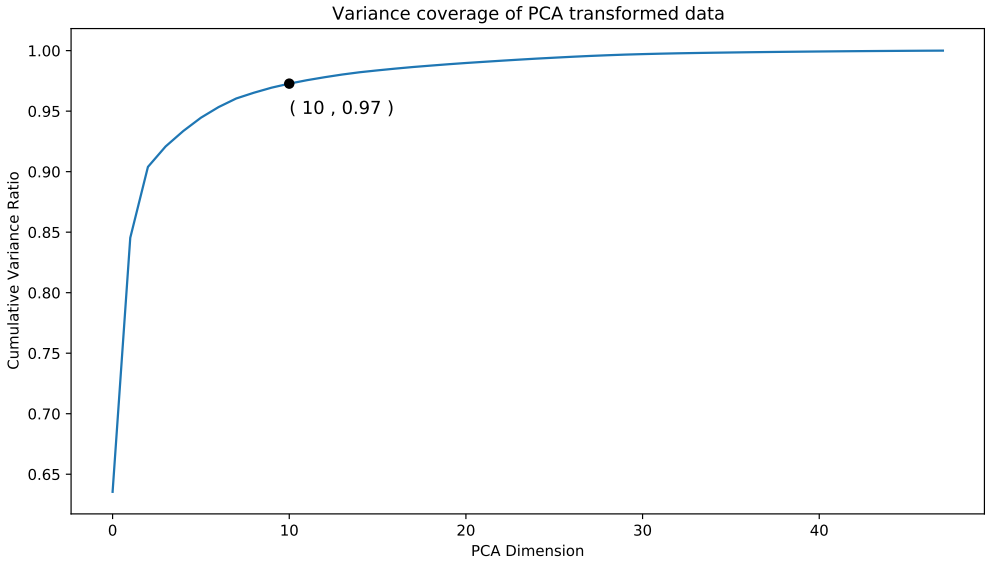


Figure 10: Using PCA to verify if an equidistant spacing is the optimal choice for the 48 dimensional frequency-feature representation chosen for the first experiments.

ratio of variance is plotted against the principal components. Roughly speaking this can be understood as the amount of variance that a principal component captures. For a mathematical definition of the cumulative ratio of check out the appendix.
So for our case already the first 10 principal components were able to capture 97 % of the variance. This suggests that most features do not enter the computation with much priority. So for optimizing the data representation one could further investigate this.

### Normalization

Another crucial aspect of data representation is the normalization chosen. Here a max normalization was chosen, because (again noticed visually) it puts stronger emphasis on the higher frequencies and therefore is more sensitive to the region belonging to frequencies above 20 kHz, which was the part that had set the white noise apart from the clean audio the most. Therefore its not a huge surprise that the breaks were not detected, because the way the data was represented didn't favor such events specifically.
Another problem of this type of normalization is that it was made relative to the observations. This means that the normalization of the entire data set in the beginning resulted in a different scaling per frequency feature than the one performed on the individual parts of the 0.2 s splitting. This distorts the data further. For example an event like break (which is approximately one sine wave without harmonics) causes a high peak for one frequency in one (or two max) 0.2 s chunks. Max normalization would than scale this peak to 1 and compress all the others. Clearly this distorts the phenomenon.
Probably the most promising area for improvement lies here, especially because calibrating it had the largest impact on the achieved accuracy in the experiments above. One could try other standard scaling mehtod or specifically design one for the given purpose.

### Definition of an anomaly

The chosen definition of an anomaly assumes that there is no useful information contained in an almost pure white noise signal. However, this is not entirely true. When one observes a continued raise of volume level peaking with a white noise signal, one could definitely argue, that there is a lot going on in the sewer pipe. As shortly mentioned in the beginning, depending on the way the flow rate will be computed later this could give insights. That way a white noise like signal wouldn't be an anomaly, but expected behaviour.
Since the sent files actually contain some flow rate measurements one could check this hypothesis.

### Accounting frequently appearing anomalies specifically

Besides the (within ten seconds) constant in time events like noise level and the short term events like the car break sound there might be other types of anomalies in such a recording. For example the horn of a car might locally look similar to a siren. So the alternating equal time horn like sound events with a relative distance of a musical perfect fifth makes up the impression of a siren. But what happens if the distance is not a perfect fifth, but an octave? Or if one tone gets held longer than the other? Such considerations of course go too far from the actual objective here, but it demonstrates that anomalies might not be fully captured by the combination of checking frequencies contained in larger chunks and checking for short term events. In order to check for for such events again longer observation times are needed again, but at specific frequencies so a distinction between horn and siren can be made. Detecting events like this could lead to a preprocessing step that masks such events, in order to still use the remaining frequencies. Nevertheless, due to the fact that for the sewer pipes it does not change much if one waits for a minute with recording, for this particular situation just waiting until any horn/siren like events stops, since they rarely continue for more than a few seconds, would be the easier way of achieving equal precision.

### Attention guided anomaly detection

This one is related to the normalization, but does it in a deep and automated way. The attention mechanism tries to mimic cognitive attention in a way that could be understood as non-linearly scaling the data depending on what it is. This technique could also be used here for improving the performance. An image based approach on attention guided anomaly detection can be found here [16].

### Deep One-Class Classification

Instead of computing an anomaly score with autoencoders as used for this task, one could try a so-called Deep Support Vector Data Description[17]. The second approach is and uses a deep neural approach. It is an unsupervised approach that specifically designed for anomaly detection and trains a neural network while minimizing the volume of a hyper sphere that encloses the network representations of the data. For a visualization of this see Figure 11).

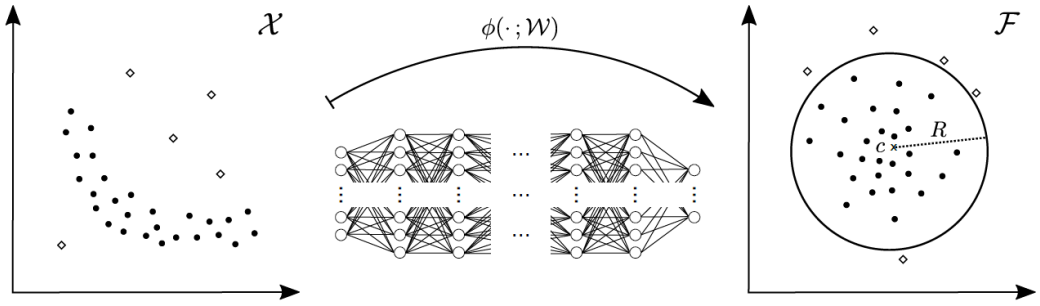


Figure 11: Learning an embedding where AD can be performed in feature space by separating data through hyper sphere. (Figure from [17])

# Appendix

## Cumulative Ratio of Variance

The task of performing PCA can be reduced to solving an eigenvalue problem. Each eigenvalue than represents the variance of the projection on the corresponding eigenvector. When trying to decide if dimensionality reduction is an appropriate decision one could check how the distribution of variance is. For example if all directions carry the same variance, there would be an equal information loss independent of the principal component when throwing it away. However, when nearly all of the variance is captured by some amount of principal components one can throw away the others without much loss of information. One nice way to quantitatively capture and depict this behavior is done through the cumulative ratio of variance. It is defined as

$$r^k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i},$$

where $\lambda_i$ denotes the $i$th eigenvalue, $p$ the dimension of the data and $k$ integer with $0 \leq k \leq p$. Note that the lower sum just sums over all eigenvalues and the upper the first $k$. Also note that due to the way the projection matrix is defined it is sorted by value, i.e. the first principal component corresponds to the direction of the largest variance, the second principal component to the second largest, and so on. This way the cumulative ratio of variance is normalized to 1 if all the variance remains and 0 if no variance is captured ($r^0 = 0$).
When visualizing this quantity one one gets a clear impression of what the principal components capture. If it raises fast early one can be sure that dimentionality reduction can be applied.
The in depth theory of how to use and visualize PCA for dimensionality reduction and more can be checked in I.T. Jollifees book about PCA[18]. Instead of Cumulative Ratio of Variance he calls it 'Cumulative Percentage of Total Variation' and scales it with a factor of 100 to get a percentage (see Section 6.1.1 there).

# References


[1] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87 – 90, IOS Press, 2016.

[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[3] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[4] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[7] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[8] M. Waskom, O. Botvinnik, D. O'Kane, P. Hobson, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, C. Fonnesbeck, A. Lee, and A. Qalieh, "mwaskom/seaborn: v0.8.1 (september 2017)," Sept. 2017.

[9] W. Koehrsen, *Blogpost*. Will Koehrsen, 2017. last opened 12.07.22 - https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d.

[10] A. Cutler, D. Cutler, and J. Stevens, *Random Forests*, vol. 45, pp. 157–176. 01 2011.

[11] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating support of a high-dimensional distribution," *Neural Computation*, vol. 13, pp. 1443–1471, 07 2001.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Y. Koizumi, Y. Kawaguchi, K. Imoto, T. Nakamura, Y. Nikaido, R. Tanabe, H. Purohit, K. Suefusa, T. Endo, M. Yasuda, and N. Harada, "Description and discussion on DCASE2020 challenge task2: Unsupervised anomalous sound detection for machine condition monitoring," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020)*, pp. 81–85, November 2020.

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[16] S. Venkataramanan, K.-C. Peng, R. V. Singh, and A. Mahalanobis, "Attention guided anomaly localization in images," 2020.

[17] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, 2018.

[18] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.