Detecting Scratch Noise in Vinyl Playback Using Machine Learning

Sean Daly - 001013392-5

April 26, 2021

A Project Report for the Degree of Computer Science, created with love... and LaTeX

Key Words: Artificial Intelligence, Machine Learning, Vinyl Record, Noise, Audio Signals, Audio Noise, Classification

Abstract: With the resurgence of vinyl coming back in to popularity, many people will be looking for a solution to reduce the noise that can be present in the playback of vinyl records. In this proposal, a project outline will be traced for the creation and implementation of a machine leaning model to detect audible noise in digital recordings of vinyl records. The project will explore previous work in related areas, analyse possible implementations, design experiments to find the optimal solution and investigate possible future uses for this solution. A model is used to predict if sound files do contain noise and will demonstrate how it can achieve an accuracy of 87%, with an F1 score of 0.88.

Acknowledgements: I would like to personally thank Yasmine Afra, my supervisor, for her guidance and help throughout the project. Without her guidance, this project would cease to exist. I would also like to thank Atsushi Suzuki, my secondary supervisor. As always, in anything I do, my parents have been of huge support throughput the course of the project and have put up with me hiding in my room for a year, working on this project. I also want to thank my nan for also being supportive and caring as always and also my granddad for giving me his vinyl collection, without his generosity this project would not exist.

The notebook containing the code used in this project will be uploaded to Kaggle (username: Sean Daly), please feel free to explore the code and try to improve on the results. As for the dataset, some checks will be required but will be uploaded to Kaggle if permitted by copyright and distribution laws etc.

Contents

List of Figures

List of Tables

Listings

0.1 Introduction

0.1.1 Preface

The music industry has seen an increase in the sale of Vinyl records, in what has been seen as a revival in the once extinct format. Vinyl record sales have increased by 223 percent since 2008 (Peoples & Crupnick 2014). When applied to music retail, customers has helped move vinyl sales, both as a political sentiment tied to independent, local consumption and a hip fashion accessory for fans who do not find digital streams, mp3s or CDs as compelling (Harvey 2017). This resurgence has seen a wave of new users and collectors of the format, with a majority learning about the characteristics of vinyl, one particular being how prone to dust and static this format is. With continuous spins, a vinyl record becomes charged with static and begins to attract airborne dust particles. These can come into contact with the stylus on the record player and can produce sounds of fuzzing and clicking. As well as dust, mishandling the record can scratch the surface leaving scratch marks, causing the grooves set in the vinyl to be removed. In acoustics, this is considered as "noise" and can disrupt and diminish the quality of the sound in playback.

Some listeners may see this as a characteristic of the format, giving the music authenticity and acting as an aesthetic. While others may see this as annoying: increased fidelity in sound is one of the many draws to vinyl, making it the favourable choice of audiophiles alike. To gain a clearer picture, one must understand why noise may be unwanted.

0.1.2 Noise in audio

In data collecting and analysis, noise is the term used to describe data that is unwanted. Noisy data can cause anomalous results that stray far from the mean, increasing variance and can reduce clarity and accuracy in model training and prediction. In the area of Audio, noise is used to describe data that is not music. This includes background noise, salt and pepper noise, impulse noise and electrical humming and fuzzing, caused by electrical equipment. Steps are taken in audio production and recording to minimise the amount of noise collected.

In the case of vinyl playback there are a number of different sources for noise to occur. These include static charge on the vinyl record, dust and debris on the record surface and also scratches which cause impulse noise. These sources can reduce sound clarity and can even obscure the music entirely if not dealt with. There are many tips and tricks that can be used to reduce noise in playback, like cleaning the records and upgrading the stylus on the record player e.t.c. Hardware noise filters have been implemented in circuits to detect spikes in acoustic signals and block them from being played out of the speaker (Noise Gates). Software implementations have also been created which can filter noise out as well. One area that could be looked into to find possible solutions is that of Machine Learning and Artificial Intelligence, which could lead to more accurate

0.1.3 Machine Learning and Classification

Machine Learning (ML) and Artificial Intelligence (AI) are areas of computer science that look into mimicking human intelligence using computer programs and algorithms. ML has successfully automated processes that once could only be done by hand and have allowed for insights into data and areas that were not humanly observable.

One of the uses of ML/AI is to sort non-organised data into groups of similar characteristics called classes. This process is known as Classification. Classification has a far reach in to different domains: from image classification to handwriting classification. Classification is usually a supervised learning problem, where the model is shown examples of each class first to train with and is then handed unlabelled data to then label independently. Classification problems can have two or more classes into which data can be labelled and grouped by. The more classes, the more complex the problem domain becomes. A majority of processes can be modelled and seen as a classification problem, just like the problem of audible noise detection. The two classes in this situation would be noise or unwanted sound and music, the sound that is wanted. ML could be used to detect and remove noisy data by classifying noisy frames in a data stream and then removing them before using smoothing to buffer any interruption caused by the removal of frames. It is in this project that this potential idea is explored.

0.2 Literature Review

As part of the research in to the topic domain for this project, previous research and work was looked into that fitted into specific key areas that belonged within the problem domain. This group of previous work will not only allow us to construct ideas to use for the project but will also highlight any gaps of knowledge in the topic domain. This will act as a basis for the project. Prior to the creation of this bibliography, the project domain was decomposed in to a set of key areas that require further research:

1. Classification
2. Noise Detection
3. Signal Preservation
4. Testing/Training data

0.2.1 Classification

Detecting noise in a stream of audio data requires the program to be able to recognise the difference between The aim of classification is to group data items with numerous character features into a specific class or group of data that have similar characteristics. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown (Kotsiantis et al. 2007). In his paper, Kotsiantis et al. (2007) reviews a number of classification methods used with supervised machine learning. In the paper, a number of techniques are named: Support Vector Machines(SVMs), k Nearest Neighbour, Naive Bayes and approaches using Perceptron networks (Artificial Neural Network). These different methods will be looked into to find the one best suited for working with audio data, but it is clear that the problem domain could be considered a classification problem.

0.2.1.1 SVMs

SVMs are learning machines that conceptually implement the idea of mapping input vectors in to a higher dimensional feature spaces (Cortes & Vapnik 1995). A simple way to picture how they work is that they calculate the function of a line or plane that best separates multiple classes of data points when plotted on a graph one dimension greater than that of the data. SVMs have been implemented in to a number of projects focusing on audio classification. In their paper, Chen et al. (2006) use a SVM to classify audio features. This model is then compared with three other classifiers: k Nearest Neighbour, Neural Network and Naive Bayes. In their testing, SVM outperforms all four classification models in terms of accuracy. In another study, Rong (2016) also use SVM in their audio classification model. Their proposed model is tested on two datasets of standard sounds and audio scenes. According to their tests, the model obtains a classification accuracy of between 86.3-87.6 percent. It is visible from these studies that Support Vector Machines are outstanding in the field of audio classification. One would use this as a basis for using SVMs in this project, with a sufficient enough decision surface, the SVM should be able to classify noisy data from music. When combined with Decision Trees, Wu (2019) proves that SVM can be even more effective at classifying audio. A combination of the two classifiers proved to increase the accuracy of classification. Another approach to the problem would be to treat it like an outlier detection problem. In this context, only one class would be identified by the classifier and therefore only one feature set. Aurino et al. (2014) use one class SVM (1-SVM) to detect burst-like audio events like gunshots, screaming and glass breaking. The audio is separated into overlapping frames which are then passed through individual classifiers, each classifying one class of sound (gunshot, screaming e.t.c). This setup could be solved with one multi class classifier, however using multiple one class classifiers maintains flexibility and accuracy as more classes are added. This could not be maintained with a multi-class classifier. This work also show that one- class classification is feasible and it may be worthwhile detecting noise from within a track of music rather than classifying music and noise separately.

0.2.1.2 Neural Networks

Another approach would be to use Neural Networks to detect and classify noise. Neural Networks mimic the structure of neurological networks found in biology. The networks are built up in layers and use weights and activation functions to determine an output based on the input data. In 'A Review of Deep Learning Based Methods for Acoustic Scene Classification', Abeßer (2020) analyse the models of Neural Networks commonly used in Acoustic Scene Classification(ACS) and Acoustic Event Detection (AED). In the paper, it is stated 'ASC algorithms mostly use CNN (Convolutional Neural Networks) based network architectures' and that 'AED algorithms commonly use convolutional recurrent neural networks (CRNN)'. In the scope of this project, there would be greater interest in AED since the goal of the model to classify noise "events" that interrupt music. RNNs would be theoretically more accurate as the built in feedback loops would help with sequential data and scenarios that require comparison to prior data points. One point to consider with RNNs is the "vanishing gradient problem" which occurs with large sequential datasets. In his paper, Hochreiter (1998) proves that the vanishing gradient problem can be combated using the Long Short Term Memory (LSTM) variant of RNN. He also concludes that other methods came with disadvantages like being only applicable in "discrete problems". In another piece of work, Stallmann & Engelbrecht (2016) uses a feed forward neural network to detect and restore noise in gramophone records. LSTM neural networks could deem effective at detecting noise from music in an mixed audio signal, however the fine tuning of the weights and other hyper parameters, as well as configuring multiple layers to work together will increase the complexity of the model drastically. This is definitely something to consider in the deciding of the model to use.

0.2.1.3 K-Nearest Neighbours

One of the simplest classifiers in machine learning is k-nearest neighbours. It calculates a data points class by comparing and analysing the "k" number of nearest data points (neighbours) and their class. As part of their conclusion, Cunningham & Delany (2020) points out some pros and cons of using k-nearest neighbours. One of its major strengths is its simplicity and ease of implementation, which could prove useful in a project like this. However, k-nearest neighbours processes all of the training data at run-time. This makes it a poor choice for working with large training and testing datasets, a point that could be the deciding factor depending on what our datasets are going to look like. Tamatjita & Mahastama (2016) conducted an study on classifying music genre based on a number of audio features. In the study, K-Nearest Neighbours was compared to the Nearest Centroid Classifier (NCC) and their performance was compared. The results showed that in all experiments performed, NCC outperformed k-Nearest Neighbours.

0.2.1.4 Naive Bayes

Lastly, Naive Bayes assumes that features are independent from the given class, simplifying learning (Rish et al. 2001). From his paper, Webb et al. (2010) describes a couple advantages of using Naive Bayes. Firstly, the training time is linear in respect to the number of training examples and attributes. This means that when processing a large number of samples, it will take a long time. This means that the suitability of Naive Bayes depends on how large the training and testing datasets become. Secondly, because Naive Bayes uses all attributes for all prediction, it becomes desensitised to noise and missing values. This could be useful for handling data in music, but could become problematic trying to detect noise, so using something more sensitive may be better tuned for the problem domain. Due to not using searching algorithms, Naive Bayes has a low variance at the cost of high bias. This could be problematic with audio data, depending on the range of signals that need processing.

The model chosen will have a major impact on how the methodology of the whole project will look. The choice of which classification method to use will come down to a number of things. Firstly, performance in audio data domains: how well the model will work on audio classification. Secondly, ease of use: will the model be easy to train? Or will it pose a problem. In the end, this will be an important factor for the whole project and is something not to take lightly.

0.2.2 Noise Detection

Another major part of the subject area is the ability to detect noise in a stream of varying data. The challenge here is differentiating noise from useful information that is just more varied than most. In their paper, Gamberger et al. (2000) propose a solution to removing noise from data used to train models, achieved through multi class saturation filtering, described as a filter for two-class learning problems which can eliminate noisy data.

In another application, Leal et al. (2018) use noise detection to single out noisy data in phonocardiograms, which can hinder interpretation and analysis of heart sounds. Their technique uses a clean sample of noise free HSs which is divided in to four second windows and then compares this clean sample with a reading, also separated in to four second windows, to decide if it is contaminated or not. A similar technique could be used for this project. A clean, noise-free sample could be compared with the vinyl recording window by window to decide if the window is contaminated or not. However, this will require the user to obtain an noise free sample of the music that they want to process.

Another example from Chandra et al. (1998) uses the Signal Dependant Rank-Order Mean (SD-ROM) algorithm to compare samples in a sliding window in order to detect and remove noise samples. Although this example does not use machine learning or AI to detect noise, it further proves that algorithms can be used to solve this problem domain. Not only can algorithms and software approaches be used to solve this problem, but also hardware implementations can prove useful as well. Herzog (2013), proposed a real time application of a median filter on a Digital Signal Processor. The filter is implemented as a doubly linked list in assembly language and is able to remove audible clicks. In his conclusion,Herzog (2013) states "In combination with a click detection algorithm it can be successfully applied as a real time click and crackle removal tool for vinyl playback". Unfortunately for the project, the implementation falls outside of scope for the project, which is focusing on applications using AI/ML techniques. However, this work shows there is interest in the removal of audible noise from vinyl playback and shows that noise detection in audio signals is feasible.

To maintain usability in future use-cases and depending on how the project is structured, the noise detection may be implemented as an auto-tagging feature for new and unseen data. In further development, audio removal techniques could be studied and added to the program, effectively creating a noise removal system.

0.2.3 Signal Preservation

Some outlying music signals in the data my be mistakenly misclassified as noise. In a practical implementation, this would lead to a reduction of signals in the music and a overall reduction in sound quality. In Cortes & Vapnik (1995), the Soft Margin Hyperplane is proposed for cases where data cannot be separated without error. The soft margin allows some errors in the classification, meaning that some errors will be allowed, reducing the loss of music data.

Another solution would be to restore this lost data. In Smaragdis & Raj (2007), a methodology is proposed to expand and restore frequencies cut off from bandwidth reduction methods. Their method uses latent component analysis to expand frequencies lost from the upper and lower bandwidth cut-off. This could be implemented in to the solution as a restorative after the model has worked on the music data. Sadrizadeh et al. (2020) propose another method for signal restoration, authough in their paper they tested their method on image signals, they conclude "the proposed algorithm works for i-D and 2-D signals that are corrupted by salt-and-pepper noise, random-valued impulsive noise and unknown random missing samples". Both variants of noise are common in vinyl playback, making this proposition feasible to implement. However, because the aim for the project is purely classification, no data is going to be removed.Therefore bandwidth restoration is unnecessary as the bandwidth will be preserved. In a future usage where the program does remove noise from the recording, bandwidth restoration may be required in order to maintain the input quality, so although signal preservation is not needed now, it may be implemented in a future use-case.

0.2.4 Testing/Training data

A major part of the project that can determine how well the model performs is the training data used to fit the model. The data will need to contain the correct features to enable us to train the model to be effective in classifying music from noise. In his overview of audio signal classification, Gerhard (2000) lists a number of commonly used features used in audio classification problems. He describes the Zero Crossing Rate: the rate at which the sound crosses between positive and negative values. He also explains the Fundamental Frequency which is the lowest frequency that repeats. However this only works as a valid feature to extract in repeating signals. Music does have repetitive elements like drums and the chorus etc. But this is broken up by the lyrics and other sections of the song. Therefore it makes no sense in using it for our project. In her article discussing working with audio files for machine learning, Baheti (2021) also discusses these features and explains in more depth about the Spectral Rolloff, the frequency under which the cutoff of the total energy of the spectrum is contained. This can be used to distinguish between harmonic and noisy sounds, thus a potential feature to use in the model.

The data will also need to be in a structure that is suitable for the system to read, work and analyse with. In one project (Lu et al. 2010), the data is separated in to sections using a sliding window to pre-segment the audio. These segments are then classified into one of the five audio events described in their paper. This approach seems effective in breaking up the audio stream into manageable chunks rather than processing the whole stream which would become hardware draining and time consuming.

The file type used to store the data should also be considered since some filetypes can compress data. In terms of audio, this could mean losing fidelity in sound. This is common with filetypes like MP3 which prioritise small filesizes over audio clarity. WAVE file formats do not compress data and even support floating point values for samples (Dobson 2000) allowing for greater accuracy when trying to represent samples. Another part in which the testing dataset is going to be used for, is the evaluation of the models effectiveness. In his web article about evaluating ML classification models, Tao (2021) discusses multiple methods to use. The one method that would best suite our needs in this domain is the measure of precision and recall. It uses the number of true positives, false positives, true negatives and false negatives to calculate how well discrete classifiers are at classifying and detecting classes. Precision and Recall can be calculated as shown below, with TP as the number of true positives, FP as the number of false positives and FN as the number of false negatives.

$$Precision(p) = \frac{TP}{TP + FP}$$

$$Recall(r) = \frac{TP}{TP + FN}$$

Two other values that are useful for evaluating models are the accuracy and the F-score. The accuracy is how often a model is correct and the F-score can be seen as an average of precision and recall. Miller (2020) provides us the formulas for accuracy and the F-score:

$$Accuracy(a) = \frac{TP + TN}{TP + FP + TN + FN}$$

$$F - Score(F1) = 2 \times \frac{p \times r}{p + r}$$

As well as this, the area under a receiver operating characteristic (ROC) graph can visualise the performance of classifiers. In his paper, Bradley (1997) describes the area to mean "the probability that a randomly chosen positive example is correctly rated (ranked) with greater suspicion than a negative sample". These graphs can be used as a more visual representation of a models accuracy than a confusion matrix. These values will allow the evaluation of the models performance as well as providing a performance metric in which can be compared with other models and classifiers.

0.2.5 Technology Review

In this section, already existing tools and software packages that can be used to help us develop and implement a solution are analysed. A number of tools will be required in order to create, analyse and test the final product solution for the problem domain.

0.2.5.1 software tools

To save time, reduce workload and error it is best practice to use pre-built and tested tools. There are two areas where program tools will be needed. Firstly, to analyse sound clips of noise in vinyl recording and secondly in the programming, training and testing of the model. For the first part, music creation and editing software will be useful for analysing the frequencies present during noise clicks and in general music. It will also be useful for in the case of having to create our own testing/training data. One tool that is free and open source is Audacity[1] which will allow us to view waveforms of the music to detect any peaks where scratches and popping may be present. As well as waveforms, Audacity also can represent the data in a Spectrogram, by running the data through a Fast Fourier Transform. This will allow us to view what frequencies are present and at what decibel readings they are achieving, which can be useful when selecting features to extract. Audacity also includes tools for editing, recording and exporting audio files as well.

0.2.5.2 software packages and libraries

The second area where software tools will be needed is in the development and training of the machine learning model. The tools required will mainly be in the form of pre-existing programming libraries. It should be noted that the programming language that will be used in the project will be Python. Python is a programming language used by a majority of researchers and scientists in the field of Machine Learning and Artificial Intelligence. There are a number of libraries aimed at helping with AI development in python. The first package, Numpy, is a library that gives access to "Numpy" arrays. The NumPy array is a data structure that efficiently stores and accesses multidimensional arrays (also known as tensors), and enables a wide variety of scientific computation (Harris et al. 2020). Numpy is also used by a lot of other domain-specific libraries, so even if Numpy arrays are not used specificity, there may be libraries that wish to utilise the use Numpy API. Another package that will be useful to us is MatplotLib, a package allowing users to create graphs and other mathematical plots to display and explore data. Hunter (2007) describes MatplotLib as "a 2D graphics package used for Python for application development, interactive scripting and publication-quality image generation". This could become useful for displaying the training and testing performance of our model in a graphical format. It can also be used for simple data exploration of our datasets and outcomes.

Another package that should deem useful is Pandas, a package allowing users to utilise advanced data structures and data tables called "data frames" similar to the ones used in the R programming language. Wes McKinney (2010) describes pandas as "a new Python library of data structures and statistical tools initially developed for quantitative finance applications" however the tools pandas provides can be used in other areas outside of financing. Pandas includes tools for for statistical datasets, data alignment, handling missing data and combining and joining datasets.

Although the above packages could deem useful, they are very broad and non-specialist for the problem domain, i.e they can be used in a variety of problem domains and are not very

[1] Audacity® software is copyright © 1999-2021 Audacity Team. The name Audacity® is a registered trademark.

domain specific. There will be a need for some more specialist and domain focused packages to aid us in the project. One such library that focuses on AI and Machine Learning is SciKit-Learn, a library providing access to neumerous tools for applications of Machine Learning and AI techniques. Pedregosa et al. (2011) says that the package "focuses on bringing machine learning to non-specialists using a general-purpose high-level language" It includes tools for preprocessing, modelling and classification techniques like Nearest neighbours and SVM. This library will probably end up being used as it provides a use-case for a critical part of the project.

Librosa is a python library aimed at audio information retrieval and audio analysis. Designed to have a low barrier entry for researchers familiar with MATLAB, Librosa relies on numpy datatypes and functions (McFee et al. 2015). Librosa also has easy to use functions for the Short-Term Fourrier Transform which can be used to analyse the frequencies in the signals in each clip. Using this library would require using Numpy, but this is the case with a lot of libraries since Numpys API acts as an foundation for most scientific libraries used in python.

Ultimately, most of these libraries will be used to some extent, with some others depending how the requirements change for the final product. Using tried and tested libraries is better than free-hand coding for a number of reasons. Firstly it saves time not having to code functions that have already been created. Secondly, using libraries reduces debugging: these libraries have been tried and tested before being publicly available. And because most are open-source they are under the scrutiny of a large community, in the tens of thousands, correcting errors in the source code. This means more man hours can be spent on the library ensuring it is sound rather than a couple of people working on a closed source project.

0.2.6 Conclusion

To conclude the literature review and technology review, a discussion is presented showing how the project has been shaped by the literature read above.

To begin with, the review has lead to the decision of SVM being the classifier of choice. Multiple pieces of literature have concluded saying that its performance was better than the other models being compared in the study. The only other model type that came close was the Neural Network, however NNs have a significantly greater amount of hyper parameters to consider. Parameters such as number of layers, inputs, outputs, layer types etc. Finding the optimal settings for all of these could be considered its own project. In the time frame that is set for this project, this would be too demanding. The "black box" structure of SVMs sacrifice customisation for ease of use and ease of application. This is the most attractive feature of this model type, along with its strong performance in similar problem areas make it the ideal choice.

Although it was touched upon in the introduction, noise removal seems to fall outside of the scope and would also complicate the project further. This can be seen a future use-case for the project and could be a direction for this particular implementation to grow towards. In terms of the datasets to be used, they will need to be segmented into sections to minimise memory usage. As for file types, WAVE format seems appropriate as it is flexible in use and does not compress data. It also has support for floating point values as well.

The technology review has provided valuable libraries to use as well as two valid choices of IDE to use for the programming sections of this project.

0.3 Analysis

In this section, the findings from the literature Review are revised and discussed. Also, the data that will be used to provide insight into the problem domain and analyse the potential routes to take to solving the problem will be analysed.

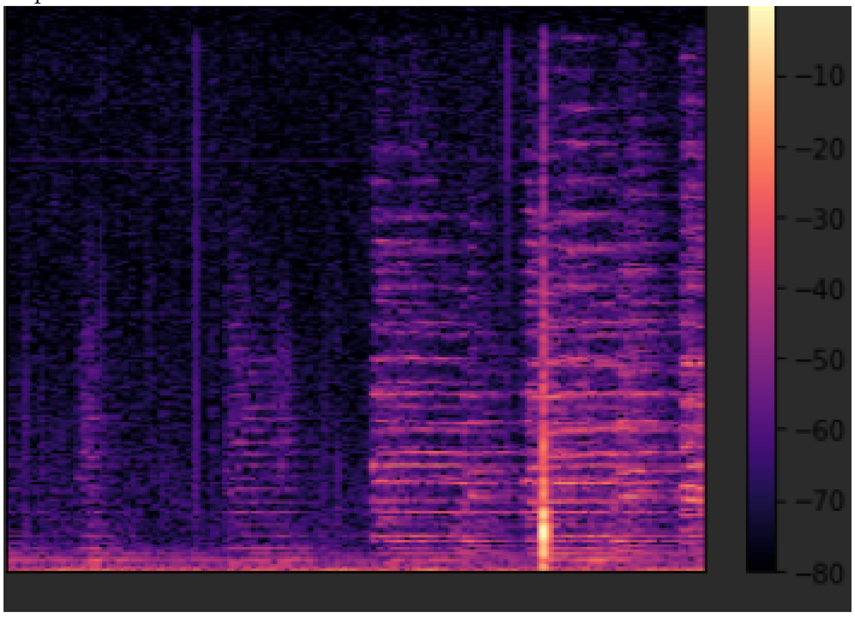
The findings from the literature and technology reviews have shaped ideas about potential approaches to take. Firstly, in deciding what model classifier to use for the project, an SVM looks to be the likely choice. In all the literature read as part of the review, SVMs seemed to perform well compared with other variations of classifiers. To classify noise, the correct features must be chosen which will allow the model to detect noise from music. In terms of how many classes to train for, it will be no more than two, however a single class classification / outlier detection approach is feasible and may be an effective way of flagging scratch noise. This includes the added benefits of reducing training data since the model is being trained to detect noise rather than music, which can be more varied depending on the instruments used, genre of music and the artist performing. However, this will require a different approach when it comes to the training data. There will be a need to shape the model to data that is purely composed of the noise that is to be detected. When it comes to structuring our data, the audio will be broken down into manageable chunks which can be handled easier by the model and also reduce memory usage.

Another major decision that will need to be made is the choice of hyperparameters to use for the model. The most important being the kernel function, the "mathematical trick" that allows SVM to operate on n-dimensional data in higher dimensional domains (Patle & Chouhan 2013). Agreeing with them, Kotsiantis et al. (2007) quotes: "The selection of an appropriate kernel function is important, since the kernel function defines the transformed feature space in which the training set instances will be classified". The kernel will have to be a decision made through a procedure of experiments to determine which kernel offers the best performance.

Aforementioned at the start of this section, the structure and form of the data that would be using for this project needed to be analysed. Two sets of data are required for this project: a training set and a testing set. The training set trains and shapes the model, where the testing set is used to validate the models performance. The training set was created using high quality recordings of vinyl playback, full details of the recording process are discussed in the implementation section. Originally, the dataset consisted of nine albums, each recorded at 44100 Hz and cut into four second intervals. This equated to over 4000 files each with 176400 frames to analyse. It was becoming apparent that the size of the dataset was far too large to handle. To combat this, some of the albums were removed, reducing the total to six and the recordings were re-sampled to 22050 Hz. The dataset now consisted of 2286 files without noise and 1143 files with noise. To balance the dataset, half the clean files were removed and the whole dataset of 2280 files split into three subsets of 760. This process is explained in detail in the implementation section.

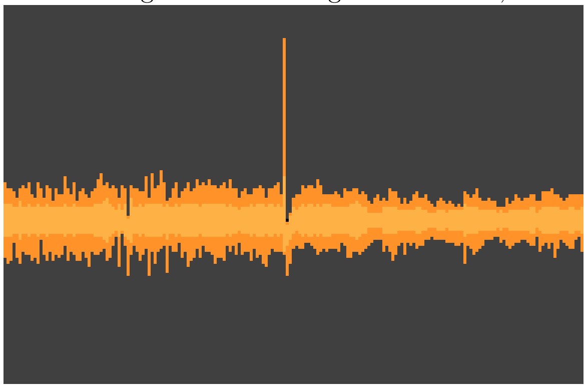
The data was then analysed using Spectrograms to analyse the sound to see how the frequencies changed in the presence of a scratch. It is clear in the Spectrograms that there is a drastic shift in the energies of multiple frequencies, hence the large white lines where the scratching occurs.

Figure 1: Spectrogram showing the change in energy (Db) of frequencies present in a sound file with a scratch present



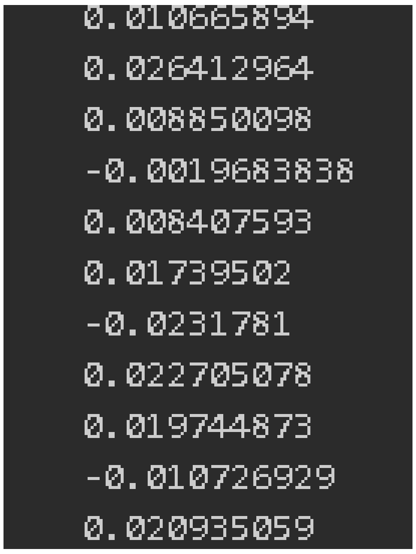
These scratches also appear in the waveform when loaded in Audacity, showing as large irregular spikes. If a spike is particularly bad, it can clip outside of the track and can be detected by Audacity, which will mark the location with a red line. In this instance the noise is not loud enough to clip outside of the track boundaries, however it is enough to audibly disrupt the flow of the music.

Figure 2: A waveform segment containing scratch noise, shown in Audacity



The last thing that needed observing was how the data in the WAV files was structured. This would determine any prior steps needed to take before feeding the data into the model (preprocessing). Upon inspection, the data in the files was of signed floats. There is one number per frame ($22050Hz \times 4Seconds = 88200Frames$) defining the magnitude of the waveform at that moment in time. This has also brought up the realisation that this has now become an unsupervised problem. Due to the amount of frames per file, labelling the frames by hand is impractical and near impossible. This finalises our choice of SVM to using a one class, outlier classifier. This is because unlike the two-class usual variations, it only needs training samples from one class.

Figure 3: The structure of the data inside of the WAVE file



```
0.010665894
0.026412964
0.008850098
-0.0019683838
0.008407593
0.01739502
-0.0231781
0.022705078
0.019744873
-0.010726929
0.020935059
```

0.4 Requirements / Specification

The analysis of our findings and from the literature review have finalised the requirements for our project. These requirements are put in place to act as a method for evaluating the project and as a "to-do list". In this section, the requirements are outlined for what the outcomes of the project are expected to be, followed by the specification: how these requirements are to be met. It should be noted that these specifications and requirements are not concrete and may be changed during development if technical or physical problems arise.

0.4.1 Requirements

The requirements were briefly outlined in the introduction, however in this section there will be more detailed descriptions of the expected outcomes of this project. These requirements may change during the implementation stage if technical or physical conflicts call for it.

Firstly, the main requirement is to produce an AI/ML model that can detect or flag scratch noise in recordings of vinyl playback. This means that the output will have to alert the user in some way of the presence of noise in the sample passed to the model. The model will also need to be trained using a dataset of sorts. As part of the review into past work, pre made datasets were looked into, however none deemed suitable for this project. Instead, one will have to created by hand. To prove that plausibility of using SVM to detect scratch noise, performance metrics will need to be recovered from an overall performance benchmark. To summarise the requirements of the project:

- **Construct model**
- **Construct testing and evaluating datasets**
- **Train model and tune hyper-parameters**
- **Collect metrics to evaluate performance**

0.4.2 Specification

Now that the requirements have been set, a technical plan as how the requirements are going to be met is needed.

- **Construct model:** The model will be constructed using the Scikit-Learn package which comes with a whole library of models ready to use. These models just require being fitted to a training dataset and then can be used for predicting with the testing dataset.
- **Construct testing and evaluating datasets**: The datasets will need to be created by hand using recordings from albums in my personal vinyl collection. The recordings will be edited, partitioned and exported via Audacity in WAV format.
- **Train model and tune hyper-parameters:** This can all be done through experimentation to determine the best settings inside of an Python programming environment. Initially, Google Colab was going to be used, however there were problems importing the datasets. Fortunately, pyCharm Pro supports iPython notebooks (also known as Jupyter notebooks) allowing us to import the dataset on the desktop rather than being confined to uploading the dataset to the cloud.
- **Collect metrics to evaluate performance:** This will be achieved through a final test to calculate metrics like accuracy, recall and precision to determine how well the model can detect noise in files of music.

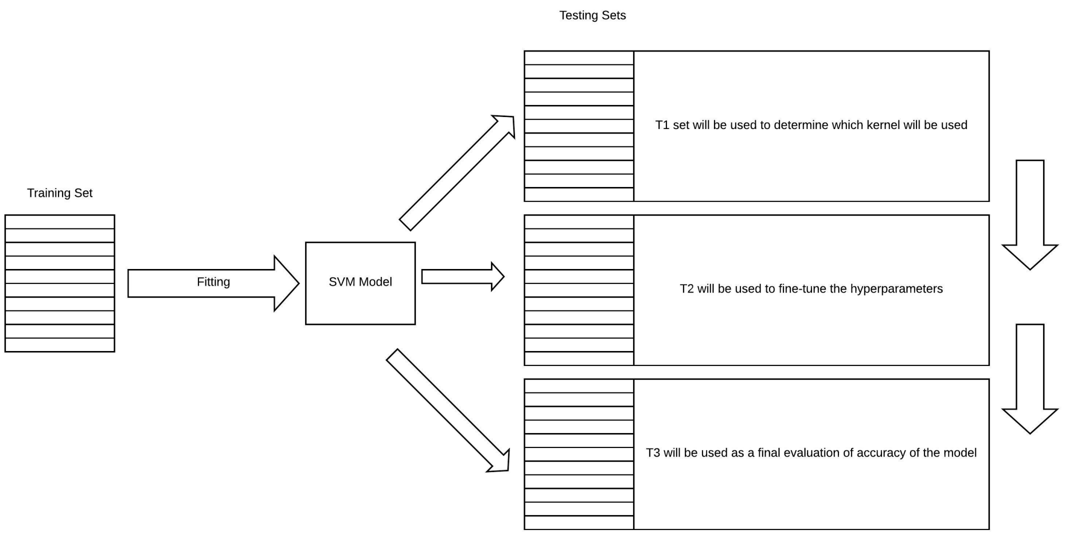
0.5 Project Structure

This section outlines the flow and structure of the project, explaining how it will be put together and how it is designed to create a solution that will meet the requirements.

Because the project is bases inside an iPython notebook, the usual programming design principles do not apply because of the open, explorable structure a notebook offers. This fits perfectly for data exploration and data analysis work-flows similar to this one. This does not prohibit the need to design methods and algorithms to automate processes that repeat themselves. This project is structured in four parts: the implementation stage, Test 1, Test 2 and finally the evaluation stage.

The implementation stage is where all components are built into the programming environment: the dataset, the SVM model and any methods used to view results. As mentioned in the Analysis section, the dataset that was created is massive, so in an attempt to minimise the scale the dataset will be split into three groups: S1, S2 and S3. Each subset has a balanced amount of noisy and clean recordings. Each subset will be used for one of the three experimental sections.

Figure 4: Diagram showing how the the three test sets will be used with the model


Testing Sets
Training Set
Fitting
SVM Model
T1 set will be used to determine which kernel will be used
T2 will be used to fine-tune the hyperparameters
T3 will be used as a final evaluation of accuracy of the model

The first section of the project will see the implementation of the data and model in code form. the dataset, dataframes and any methods that will deem useful will all be implemented in this section. There will be a single train/test cycle to check that everything is working. Once the implementation stage is completed the two test stages can begin.

Test 1 is in two parts: part 1 will see experimentation with the Testing data to see how well the model can perform without any changes to the model itself. The training set will be edited, increased and decreased to optimise performance. Different features will be extracted and used to see if it differs performance. Once the model is performing suitably part 2 will determine which Kernel will be most appropriate. Multiple kernels will be tested to see which one is quickest in operation and can detect noise most effectively. Hand picked sound files from the S1 subset will be used to see if the model is suitably trained. In the event where two kernels are similar in performance, both will be utilised in Test 2.

Test 2 will see the hyper parameters being tuned to further optimise performance, similar to Test 1, hand-picked files will be used to determine how well the model performs. Once testing is completed, the model will move on to the evaluation stage.

The Evaluation stage involves the model observing every file in S3 and flagging which ones have noise in them. These "predictions" will be taken and compared with the true labels to calculate the number of true/ false positives and true/false negatives. From these values, accuracy, precision etc. can be calculated.

0.6 Implementation

At this point in the project, the model and datasets can now begin to make shape. In this section, the dataset (already created for the Analysis section) will be inserted into data frames and into the programming environment. Models will be created and methods programmed to create a space ready for experimentation.

0.6.1 Testing/Training Data Collection

In this section, the procedures and steps that were taken to create the necessary training and testing datasets required for the model are explained. For each dataset, the tools used and the process steps taken to capture the data and enable it to be processed in a software environment will be outlined. It should be known that although the data is coming from albums from other artists, no EDA processes were needed to be carried out due to the fact that this data was collected by the author of this paper and is not going to be publicly available.

0.6.1.1 Testing Data

The testing data was the first requirement to be implemented as part of the analysis into the problem domain. As mentioned in the analysis section, the training data was split in to three subgroups. This means that one group for validating overall performance and the other groups for model tuning. The ideal medium to use would be music captured from vinyl records which would have scratches on the surface. However, in the case where the album is in good condition and has no scratches, noise clicks can be manually added to the captured data. The finished capture along with the added noise get segmented into four second sections which then are labelled depending on what album they are from and if the section is a noisy sample or clean sample. The process in creating the dataset was as follows:

- **Hardware:** A number of different pieces of hardware were used to extract the data. Firstly a turntable with USB connectivity was used to allow the vinyl playback to be captured digitally. An amp and headphones were used to listen to the playback whilst the record was playing, the amp does not effect the recording of the vinyl since the computer is receiving input from the turntable directly. Of course the final piece of hardware needed are the vinyl records themselves. A number of different records were used of varying genres and artists to vary the style, instruments used and general sound. Below is a list of the albums used in the Testing dataset, however (as explained in the analysis section) some were removed:
  - Chris Rea - The Road To Hell (Slow Rock, Country)
  - Pink Floyd - Dark Side Of The Moon (Rock)
  - Various Artists - Selections From The Soundtrack Of Girl Groups: The Story of a Sound (Pop, Mo-town)
  - King Crimson - In the Court of the Crimson King (Rock, Alternative) REMOVED
  - Led Zeppelin - Led Zeppelin 2 (Rock) REMOVED
  - Madonna - The Immaculate Collection (Pop) REMOVED
  - Prince - 1999 (Pop)
  - The Beatles - Revolver (Rock, Pop)
  - John Williams / London Symphony Orchestra - Star Wars : Original Soundtrack (Classical, Cinematic)

  It should be noted that all the vinyl used are first editions, meaning that no remastering or tampering for re-issue has occurred. This gives us the purest signal and sound closest to what the artist and studio intended. Secondly all records were wiped down with an antistatic brush to remove dust and static which could create more unwanted noise.

- **Software:** Only one piece of software was used in this method: Audacity. Audacity is a multi-track audio editor allowing users to mix and edit tracks. This software will allow us to create and export sound files as well as labelling them.

- **Method:** In creating the testing set, a vinyl record is taken and placed on the turntable. It gets brushed with the anti-static brush to ensure the surface is clean and discharged of static. Audacity is opened and a new track is created and record is pressed. The needle is placed on the record and allowed to play all the way through. When the first side is finished, the record is flipped, however the recording continues. Only once the vinyl has finished playing on both sides can the recording stop. In the case where an album has two LP vinyl (four sides to play) only the first LP disk (the first two sides) will be played. Once the recording is finished, the unwanted sections are taken out. These sections include noise from lifting and positioning the needle, quiet sections between tracks and more noise from turning the record over. Once these are removed from the recording, pure music should only be left. Pre-existing noise in the recording, which is identifiable via the waveform in Audacity is analysed to make sure it is suitable for use. Using the Rhythm Track Generator built into Audacity and setting the beat sound to "Noise Click", sound that is closely similar to the sound that appears naturally in scratches on vinyl can be generated. These clicks are generated on a new track underneath the recording and get generated in long sections to create multiple noisy samples. These are spread evenly along the track to ensure there is an even amount of noisy and noiseless samples. The "Regular Interval Labels" tool is used to create labelled regions lasting four seconds with no gaps in-between. Each section is then labelled by hand with an acronym of the album (e.g. Dark Side Of the Moon has DSOTM) it is from and also a 1 or 0 to indicate the presence of noise in the sample. Once all regions are labelled, they are all exported to a folder with each labelled region in its own file. All sections are exported in WAV format for maximum clarity and resolution.

0.6.1.2 Training Data

The training data will play the role of fitting the model to detect the noise associated with scratches on the vinyl's surface. This dataset was implemented in the implementation phase once the requirements and structure were drawn out, also after the Analysis of the test dataset where it was clear that a one class SVM was most suitable. The data consists of scratch noise samples taken from both real vinyl playback and generated via software. There is no need to separate the file into chunks as the training data shouldn't need to be too large. The albums that were removed from the testing dataset were analysed for scratches in Audacity, which were then cut and pasted into a new audio track and then looped multiple times. Multiple training sets were created, each with an increasing variety of scratches to reduce over fitting, however only one file is needed to properly train the model. The training set was edited and tested until the trained models were providing good results without any changes to Kernel type or any hyper parameter adjustments. These adjustments to the training data included increasing/decreasing the amount of samples, increasing/decreasing the variety of samples and editing single instances of noise examples if they contain non-noisy data which can cause the model to detect music as noise.

0.6.2 Data Import

Now that the dataset has been created, it now needs to be imported into a python environment ready to be utilised with python code. Originally, Google Colab was going to be used as the IDE for the project, however does not support direct imports from the desktop. Instead files must be uploaded to Google Drive first before attaching the drive to the project and then loading the files with code. Due to the size of the dataset, upload speeds were incredibly slow, instead a desktop solution was needed. PyCharm professional supports both direct file imports from the desktop and also ipython projects, making it the best solution for this project.

```
_newDATA = np.array(os.listdir("C:\\Users\\Sean Daly\\Desktop\\
Final Year Project\\_newDATA"))
```

Listing 1: Path code allowing access to the dataset via Python code

Once the files were inside the project area, the files needed to accessible via code and then the quantities of clean and noisy files appropriately balanced. Firstly, the names of all the files were loaded into a Pandas DataFrame:

```python
data = {'File Name':  _newDATA}
df = pd.DataFrame (data, columns = ["File Name"])
```

Listing 2: Creating the initial DataFrame with all file entries

Also needed is a label for each row indicating the presence of noise and also the file path which will be used to load files in later use. This is achieved by lading the name of the file and checking the identifier digit (1 or 0) and using the 'os' library to find the absolute path of the file. Once this is done, new dataframes are made to hold the data and then are merged with the existing DataFrame:

```python
labels=[]
for file in _newDATA:
    l = file[len(file) -5]
    labels.append(l)
print(labels)
print(labels[10])
print(labels[11])
c = 0
for i in labels:
    if i == "0":
        labels[c] = "false"
    else:
        labels[c] = "true"
    c = c+1
temp = pd.DataFrame (labels, columns = ["isNoisy"])
_frame = pd.concat([df,temp], axis=1)

paths = []
for file in _newDATA:
    l = os.path.abspath(file)
    paths.append(l)
print(paths)
temp = pd.DataFrame (paths, columns = ["filepath"])
frame = pd.concat([_frame,temp], axis=1)
frame
labels=[]
for file in _newDATA:
    l = file[len(file) -5]
    labels.append(l)
print(labels)
print(labels[10])
print(labels[11])
c = 0
for i in labels:
    if i == "0":
        labels[c] = "false"
    else:
        labels[c] = "true"
    c = c+1
temp = pd.DataFrame (labels, columns = ["isNoisy"])
_frame = pd.concat([df,temp], axis=1)

paths = []
for file in _newDATA:
    l = os.path.abspath(file)
    paths.append(l)
print(paths)
temp = pd.DataFrame (paths, columns = ["filepath"])
frame = pd.concat([_frame,temp], axis=1)
frame
```

Listing 3: Code that labels the entries and adds their file paths to the DataFrame

Now that the files are imported, the data needs to be balanced. The ideal ratio would be a 50:50 split of clean to noisy files. To begin with, the number of file with each label is counted using;

```python
frame["isNoisy"].value_counts()
```

Listing 4: Viewing the split of noisy and clean data in the dataset

It shows that 2286 files are clean and 1143 contain noise. This is perfect as only half of the clean files need to be removed to gain a perfectly balanced set.

Once the set is balanced, it is divided into three subsets and shuffled, each being used for a separate part of the project.

0.6.3 The Model

The SVM model is easy to implement into the project. All is needed is to use `import sklearn as sk` to import the sci-kit learn library which includes SVM models to use. IN particular, it includes `SVM.OnceclassSVM()` which is a singular class version of SVM. Alongside the sk-learn library, the joblib libary allows us to save trained instances of models to a file and then reload them, saving huge amounts of time by removing the need to retrain the models. The models can be trained using `model.fit()` which is included in the sci-kit learn library. A number of models will need to be used, each with a different kernel so the performance can be compared in the first two experiments.

0.6.4 Observation methods for performance

To be able to understand how the model is behaving, performance metrics are needed in both numerical and graphical form. SVMs can be considered "black box" classifiers, in which the user does not have insight into the inner workings. All the user has full awareness of is the input and output data. This is unlike a Neural Network which the user can define the structure of (layers, inputs, outputs e.t.c). To understand what the SVM is doing, metrics are needed to observe its behaviour. For the first two experiments, the performance of the models needs to be observed, hence the need for a method that can visualise and quantify the prediction process:

```python
def predict(f, mod):
    model = mod
    sample = f
    test = librosa.stream(sample, block_length = 88200,
     frame_length = 1, hop_length = 1)
    val = []
    for line in test:
        for i in line:
            val.append([i])
    v = np.array(val)
    v.reshape(-1, 1)
    pred = model.predict(v)

    clean_count = 0
    noise_count = 0
    for flag in pred:
        if flag == -1:
            clean_count = clean_count + 1
        if flag == 1:
            noise_count = noise_count + 1
    print("Noisy frames : " + str(noise_count))
    print("Clean Frames : " + str(clean_count))
    pred.shape

    X = v
    Y = [range(0,X.size)]
    colors=['red' if x == 1 else 'blue' for x in pred]

    plt.scatter(Y,X, c = colors)

    stream = librosa.stream(sample,
                            block_length = 2048,
                            frame_length = 2048,
                            hop_length = 2048)
    for y_block in stream:
        D_block = librosa.stft(y_block, center=False)
        S_db = librosa.amplitude_to_db(np.abs(D_block), ref=np.max)
        plt.figure()
        librosa.display.specshow(S_db)
        plt.colorbar()

    ipd.Audio(f)
```

Listing 5: The predict method used to evaluate the models performance on individual files

The function above called `predict()` is a function that takes a sound file and trained SVM model as arguments and returns three things: First is a count of the total frames in the file along with the number of clean and noisy frames, secondly is a graph showing a scatter plot of the data in the sound file. These points intentionally recreate the waveform if viewed in audio software. The points are color coded depending on the prediction made by the classifier for that particular point. The color is blue for points representing music and red if that point is flagged as noise.

Figure 5: An example of a scatter plot produced, containing no noise

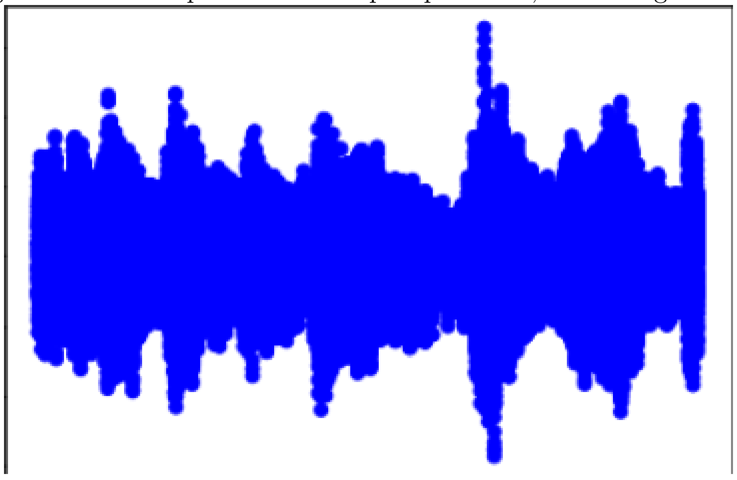
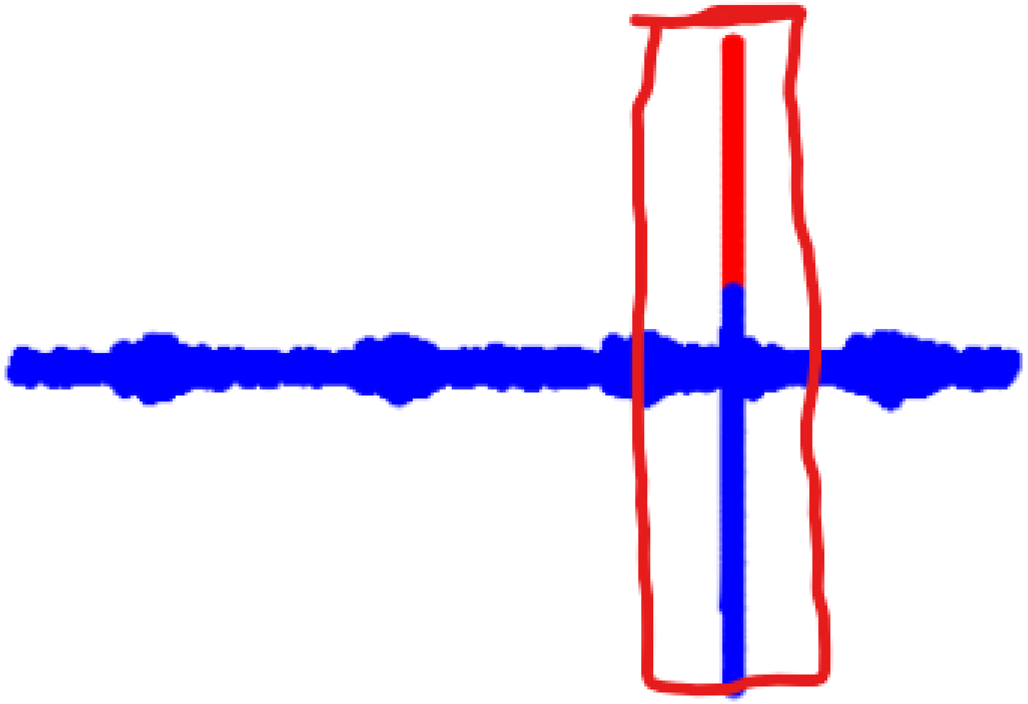


Figure 6: A scatter plot with noise(shown via the red square, drawn by hand) with the detected noisy frames (shown as the red dots, detected and plotted via the Predict method)



Lastly is an audio playback widget allowing us to listen to the music. Because the individual frames in each audio file are not labelled, this part of the project becomes an unsupervised classification problem. So the results need to manually checked and confirmed. The audio playback widget will allow us to listen out for noise in each file to double check that the model is working properly.

0.7 Testing and Experimentation

In this section sees the beginning of the first two experiments to find the optimal parameters for the model. As explained in the Project Structure, there will be two experiments carried out. In this section, the methodology as well as the results of those two tests will be discussed.

0.7.1 Test 1: Kernel Performance

As discussed in the Analysis section, the choice of kernel can have a significant impact on the performance of the model. In this experiment, four kernels were trialled in a series of runs on eight files hand-picked from the S1 group of sound files. The files consisted of both noisy and clean files.

0.7.1.1 Variables

There are two variables that are being observed in this experiment: the time it takes (in seconds) to make predictions and the number of frames that it flags as being noisy. The reasoning behind these choices vary. Firstly the time taken to make predictions could potentially effect the decision if the model was to be implemented into a real-time solution where speed would be desirable. Secondly, the number of frames flagged as noisy can be used as an indicator for lack of performance when compared with the other kernels. All other variables will remain the same apart from two: the model which has been trained with a particular kernel and the file that the models will be predicting. This experiment will see four different kernels, each with their own SVM instance, being used to make predictions. The four kernels that will be used will be Linear, RBF, Sigmoid and Polynomial. It should be noted that no other hyper parameters have been changed manually at this stage. All are set to the default values as defined in the Scikit-Learn library. As for the sound files being used from the dataset, the following files were picked:

- 168CR RTH sect0 (clean)
- 152SW sect0 (clean)
- 13DSOTM sect1 (noisy)
- 186GG sect1 (noisy)
- 110Rev sect0 (clean)
- 101CR RTH sect1 (noisy)
- 180DSOTM sect0 (clean)
- 122CR RTH sect1 (noisy)

0.7.1.2 Methodology

The methodology is rather simple, each model will take it in turn to predict each of the files. The `predict()` method will be used to visualise the performance. Once the output appears, the results are recorded.

0.7.1.3 Results

The results from the experiments showed that each of the kernels performed consistently across all eight of the files they tested on. Linear was the quickest kernel out of the four, providing predictions in as little as 18.8 seconds compared to 31.3, 64 and 65 seconds using polynomial, sigmoid and RBF respectively. Linear, polynomial and sigmoid all detected the same number of frames across all eight of the files tested. They all missed noise in T6 and T8 which were both labelled noisy files. RBF did not detect as many noisy frames in each file as the other three, even detecting some frames as noisy in a file labelled clean. However RBF did pick up noise in the two files that the other three missed. This is promising to see and is plausible that RBF is more effective, yet unoptimised, at detecting noise. Both RBF and Linear kernels will be taken into consideration in the second test. Both Sigmoid and Polynomial kernels have been eliminated by the Linear kernels performance.

Table 1: Results from Test 1

	T1 (Clean)		**T2 (Clean)**		**T3 (Noisy)**		**T4 (Noisy)**	
Kernel	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**
Linear	0	18.8	0	18.7	49	18.6	23	19.1
Polynomial	0	31.3	0	31.5	49	31.3	23	31.7
Sigmoid	0	64	0	64	49	64	23	64
RBF	0	65	0	65	13	65	39	65

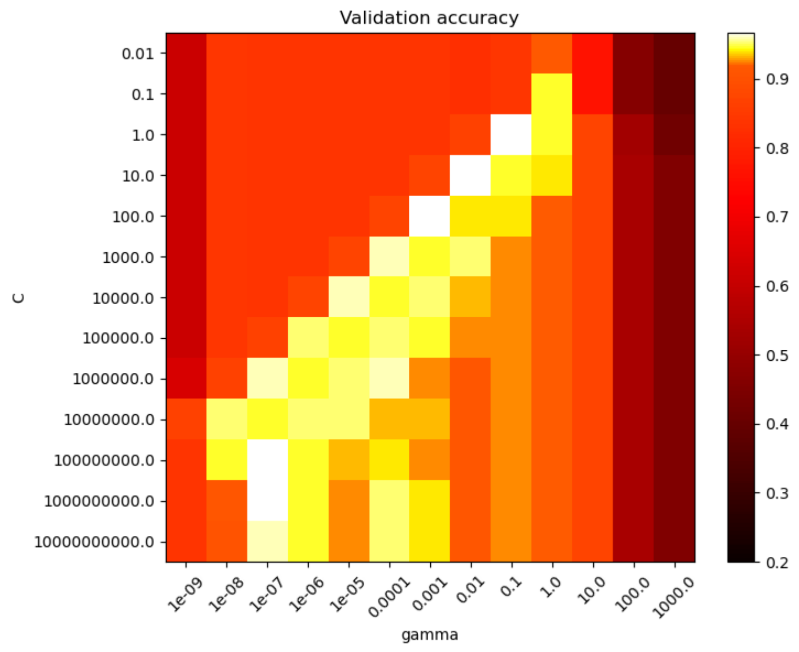
Table 2: Continuation of results from Test 1

Kernel	**T5 (Clean)**		**T6 (Noisy)**		**T7 (Clean)**		**T8 (Noisy)**	
Linear	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**	**Frames**	**Time (sec)**
Polynomial	0	19.2	0	19.1	0	19.1	0	19.1
Sigmoid	0	31.7	0	31.8	0	31.7	0	31.7
RBF	0	65	0	65	0	65	0	65
	0	65	10	65	4	65	41	65

0.7.2 Test 2: Hyper parameter Tuning

The next group of experiments will determine the optimal set of hyper parameters for best performance with each kernel. Both Linear and RBF kernels will be used, each time with different hyper parameters. According to the user documentation for the Scikit-learn library, the `OneclassSVM` method contains a variety of different parameters to tune. However most of them only effect polynomial and sigmoid kernels. RBF has two main parameters: gamma and C. C, or Nu in the case of OneClassSVM, effects how simple the decision surface is, where a high Nu will try to classify everything correctly with a complex surface and a low Nu will use a straighter surface, however may get some decisions wrong. Gamma effects the influence a single training point has on the model. According to the scikit-learn website, a gamma of between 0.001-0.1 and a Nu reading of between 100-1 provides the best accuracy. The exact values may be different in this scenario, hence the need for such an experiment. The Linear kernel does not have any hyper parameters that can drastically change the behaviour or its performance so will be left to its default parameters when compared to RBF. In a graph in their user documentation, the optimal values of C and Gamma for best validation accuracy are 0.1 - 100 and 0.001 - 0.1 respectively. However this is with a 2 class SVM, so this may not reflect well with our model.

Figure 7: Graph showing the validation accuracy of the RBF kernel with various values for C and Gamma (source: Scikit-learn user documentation, RBF SVM parameters)



0.7.2.1 Variables

The same variables apply to this experiment in the same way they did to the first. However there will be changes to the hyper parameters this time around. There will be six models using the RBF kernel: M1, M2, M3, M4, M5 and M6 each with the following starting values:

- M1: Gamma = 0.001
- M2: Gamma = 0.01
- M3: Gamma = 0.1
- M4: Nu = 0
- M5: Nu = 0.5
- M5: Nu = 1

The files being used in this experiment are from the S2 subset and again have been hand-picked to ensure a good variety of noisy and clean files. The files being used are:

- 396SW sect1 (noisy)
- 324CR RTH sect0 (clean)
- 344Rev sect0 (clean)
- 311GG sect1 (noisy)
- 33SW sect0 (clean)
- 31CR RTH sect1 (noisy)

0.7.2.2 Methodology

Unlike Test 1 which used quantitative results to back up the decisions made, test 2 will be analysing qualitative data such as the scatter plot produced and the number of frames detected, which then need validating by a human. Due to the amount of trial runs and number of variable changes, a more descriptive discussion will be used to present the findings. The experiment will first find the optimal value for Nu, since it has the biggest impact on performance. This will be done through trial runs with three models, each using a different value of Nu. The models will then make predictions with each of the models on each of the files to validate the performance. The models will then be retrained with different values of Nu,, adjusted according to the results. This will then be repeated, this time with a focus on Gamma. Finally, the two values will be used together in a single model to clarify that the model performs to expected levels. This process will be repeated with both RBF and Linear kernels.

0.7.2.3 Findings

The experimenting began with the RBF kernel. Models M4, M5 and M6 were used to find the optimal Nu value. It was found in the firs round that a value of 0.001 increased prediction times but made the model too sensitive, labelling frames of music as noise. A Nu value of 0.999 did the exact opposite, meaning the best value was somewhere between 0.001 and 0.5. Multiple rounds of tests narrowed the value down to somewhere between 0.25 and 0.37, with going any lower resulting in excessive amounts of frames being labelled as noise and larger values causing the prediction time to increase. It was eventually found to be 0.345. When tested on all the files, the RBF model still misclassified some frames. When it came to experimenting with the Gamma value, it was found to have no effect on the model, all results were exactly the same.

It was found that the linear Kernel missed a couple of noise scratches with the unmodified parameters. But changing the value to the newly found one (0.345) made the model detect the noise more accurately whilst maintaining a quicker prediction speed than that of the RBF kernel.

0.7.2.4 Results

The results from the findings and experimentation showed that the Linear model provides better performance, maintaining a greater ability to classify noise from music more accurately than the RBF kernel, even with parameter optimisation. This means that the final model will use the Linear kernel and a Nu value of 0.345. As for the Gamma value, no observable changes occurred with changing its value, therefore it will be left to its default value as determined by the scikit-learn library.

Figure 8: A scatter plot of the RBF kernel using a Nu value of 0.3525. You can see that it detects the scratched (marked via red squares) but also mistakes some of the music to be noise

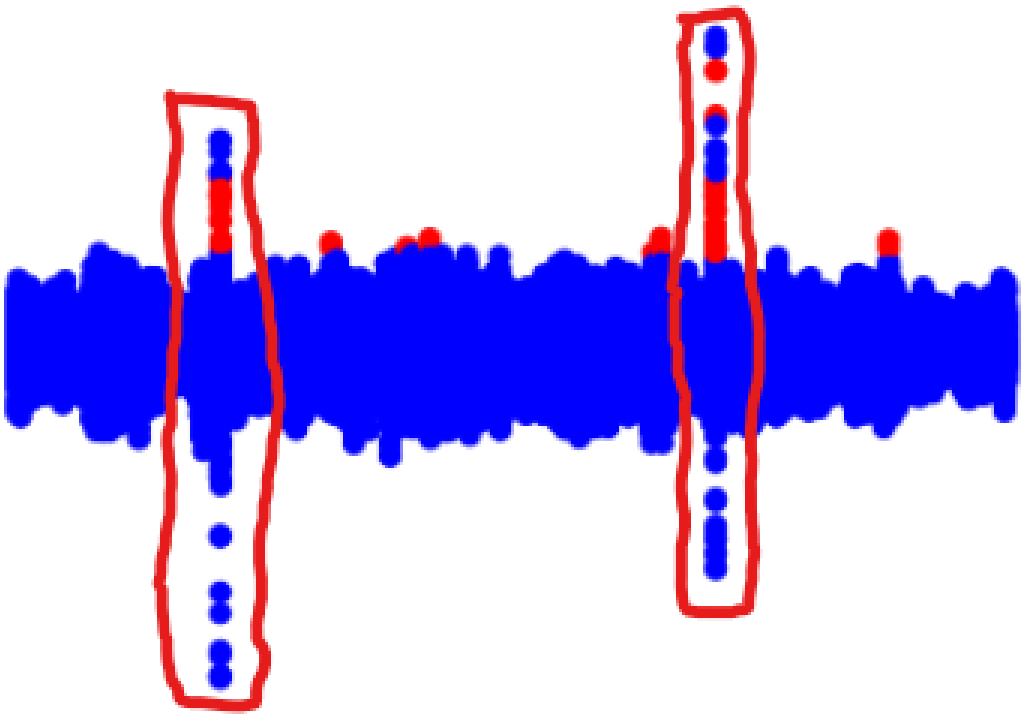
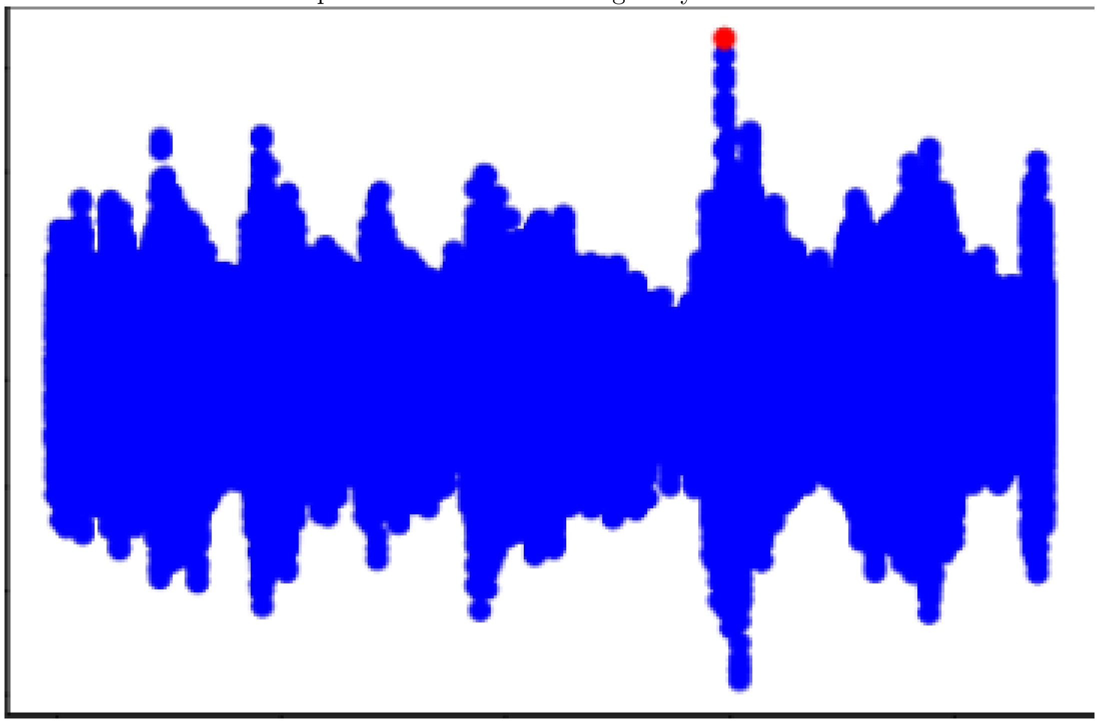


Figure 9: A scatter plot of the RBF kernel using a Nu value of 0.3525. This file has no noise, however the model seems to predict one frame as being noisy.



0.8 Model Evaluation

The final section of this project is a file-by-file test using the s3 subset. In this experiment, the final form of the model will be tasked with predicting if each file in the s3 set, all 762 of them, does or does not contain noise. These predicted labels will then be compared with the hand-made base labels to calculate the number of true/false positives and true/false negatives, in which can then be used to calculate, accuracy, precision, recall and an F1 score.

0.8.1 Methodology

This experiment will require the model to predict all 762 files by labelling them "True" of "False" on the question of if the file contains noise. tHis will be done by the following code:

```

file_predictions = []

for ind in _s3.index:
    flag = False
    print(_s3.loc[ind,"filepath"])
    sample = _s3.loc[ind,"filepath"]
    test = librosa.stream(sample, block_length = 88200,
        frame_length = 1, hop_length = 1)
    val = []
    for line in test:
        for i in line:
            val.append([i])
    v = np.array(val)
    v.reshape(-1, 1)
    pred = model_linear.predict(v)
    for p in pred:
        if p == 1:
            flag = True
    file_predictions.append(flag)
    print(flag)
```

Listing 6: The code used to evaluate the models performance at predicting sound files

The code seen above is simple in function. It iterates through the s3 (seen as _s3 in the code) DataFrame, taking the file path from each row and loading it into a Librosa Stream object. The data from the stream is then placed into a Numpy array and reshaped. The model is then used to produce an array of predictions per frame. The array is checked to see if any of those predictions were noisy, or "True". If so it returns "True" for the file, otherwise "False". These labels are stored in the array `file_predictions`. The array is then inserted into an array with the rest of the s3 dataframe to create a new dataframe called `Eval_Frame`.

```

t = pd.DataFrame(fp, columns=["Predictions"])
Eval_Frame = pd.concat([_s3,t], axis=1)

TP = 0
TN = 0
FP = 0
FN = 0

for ind in Eval_Frame.index:
    if Eval_Frame.loc[ind, "isNoisy"] and Eval_Frame.loc[ind,
            "Predictions"] == "True":
        TP = TP + 1
    if Eval_Frame.loc[ind, "isNoisy"] and Eval_Frame.loc[ind,
            "Predictions"] == "False":
        TN = TN + 1
    if Eval_Frame.loc[ind, "isNoisy"] == "True" and Eval_Frame.loc[ind,
            "Predictions"] == "False":
        FN = FN + 1
    if Eval_Frame.loc[ind, "isNoisy"] == "False" and Eval_Frame.loc[ind,
            "Predictions"] == "True":
        FP = FP + 1

precision = TP / (TP + FP)
recall = TP / (TP + FN)
accuracy = (TP + TN) / (TP + FP + TN + FN)
F1 = ((2 * precision * recall) / (precision + recall))

TPR = TP / (TP + FN)
FPR = FP / (FP + TN)

print("precision: " + str(precision))
print("recall: " + str(recall))
print("accuracy: " + str(accuracy))
print("F1: " + str(F1))
print("TPR: " + str(TPR))
print("FPR: " + str(FPR))
```

Listing 7: The summary code used to count the predicitions and calculate accuracy and precision etc.

The rest of the code compares the base labels with the predictions made. It compares the two labels, figuring out if they are true/false positives or true/false negatives, and keeps a count of each. With these values in place, the values of accuracy, precision, recall and recall along with the True Positive Rate (TPR) and False Positive Rate(FPR). These are calculated using the formulas for each. These values will help in evaluating the model as a whole.

0.8.1.1 Results

The results from the experiment counted 371 true positives, 297 true negatives, 84 false positives and 10 false negatives. Using these values, performance metrics could then be calculated. It is shown that the model was 87.66% accurate in predicting files. This means that 87% of the time, the model was correct. It also shows that 13% of the time that the model was wrong, clearly there is room for improvement. For the recall, a value of 0.9737 was achieved, which translates to the model having high ability to identify the positive class i.e. clean files. The precision came to 0.81 meaning that a true positive is more than certainly going to be a clean file. Overall this gives the mode an F1 score of 0.88 which can be seen as an average of precision and recall.

These values show that the model is adequate at labelling noisy files and being able to distinguish between noisy and clean files. However, from what has been observed in the later experiments, there is room for improvement when it comes down to a frame-by-frame basis. This is backed up by some of the kernels in Test 2 missing noisy frames.

0.9 Evaluation

In this section, the whole project is reflected on in terms of what the results mean, potential improvements, pitfalls and future use-cases in which the outcomes of this project will be useful and can be worked on.

0.9.1 Project Findings

If there is anything that should be taken away from this project, it should be that SVM is capable of detecting noise in audio recordings of vinyl playback. An accuracy reading of 89% shows that SVM is more than capable of predicting files. However when it came down to frame-by-frame accuracy, it is observable that there is room for improvement. This is also evidential by the number of false readings that the model gave. This can be improved upon with better and more varied training data. Overall, the model came out better than expected, a target of 80% accuracy was imagined as a goal to aim for, not to be exceeded. Further experiments and fine tuning of the hyper parameters will improve the models performance as well as a more varied training dataset.

0.9.2 Project Problems

Throughout the project there have been areas that have been well structured and have been advantageous. However they have been overshadowed by some of the weak points in this project. For any future work in this area to be of a greater success than this one, these problems must be addressed. Firstly, the dataset was not properly labelled. The labels that were produced were only suitable for classification on a file-by-file basis. When it came down to a frame-by-frame resolution, those labels became useless. The first two experiments were theoretically unsupervised learning scenarios which required human intervention to validate the results. In a future study, labels must be produced not only on a per file basis but also on a per frame basis.

Another major area in need of improvement is the training data that was used in training the model. More variation is needed to ensure that no scratch noise is missed. Also the quality of the samples could be improved to ensure no music data is accidentally included. These areas should accelerate project growth once fixed.

The final evaluation of the model in the file-by-file test could of produced some more graphical results. An ROC chart was going to be produced, however time and technical limitations resulted in this being impossible.

0.9.3 Future Use-cases

There are many areas that this project could lead to in the future, both as products and for future areas of research. In terms of future areas to conduct research, comparisons of different models with this one would be beneficial to see if better performance can be achieved. One particular model that was not looked into in this project was Neural Networks. Further study into this model or other models for audible noise detection is welcomed. This work can also lead to, and be the basis of, other audio based classification problems, which there are already plentiful of as seen in the Literature Review. A more direct extension from this work would be to implement noise removal algorithms, like the ones researched in the literature review.

In terms of physical implementations, real-time audio filtering products may be able to utilise Machine Learning to detect and remove noise. Further research is needed for detecting other noise types rather than the scratch noise that is focused in this study. This work could also be implemented as a plug-in or a feature in audio editing software for detecting noise. Audacity could use a plug-in that detects scratches automatically rather than having to be done by hand. This could also be a stand alone application that could be sold commercially.

0.10 Conclusion

To conclude this report, the study has shown that the model proposed in this paper can achieve 87% accuracy when predicting if the contents of an audio file containing a recording of vinyl playback includes noisy data caued by scratches on the vinyl's surface. Experiments have shown that whilst the accuracy of the model is high on a file-by-file basis, the performance is questionable at a frame-by-frame level, often misclassifying frames and missing some noisy frames. This leads to more research and experimentation needed.This will in turn improve the file-by-file accuracy. Overall, the model shows promise in its performance and could lead on to real-time applications which could remove the noise completely.

Bibliography

Abeßer, J. (2020), 'A review of deep learning based methods for acoustic scene classification', *Applied Sciences* **10**(6).

Aurino, F., Folla, M., Gargiulo, F., Moscato, V., Picariello, A. & Sansone, C. (2014), One-class svm based approach for detecting anomalous audio events, *in* '2014 International Conference on Intelligent Networking and Collaborative Systems', pp. 145–151.

Baheti, P. (2021), 'Working with audio signals in python'.
URL: *https://heartbeat.fritz.ai/working-with-audio-signals-in-python-6c2bd63b2daf*

Bradley, A. P. (1997), 'The use of the area under the roc curve in the evaluation of machine learning algorithms', *Pattern Recognition* **30**(7), 1145–1159.
URL: *https://www.sciencedirect.com/science/article/pii/S0031320396001422*

Chandra, C., Moore, M. S. & Mitra, S. K. (1998), An efficient method for the removal of impulse noise from speech and audio signals, *in* '1998 IEEE International Symposium on Circuits and Systems (ISCAS)', Vol. 4, IEEE, pp. 206–208.

Chen, L., Gunduz, S. & Ozsu, M. T. (2006), Mixed type audio classification with support vector machine, *in* '2006 IEEE International Conference on Multimedia and Expo', IEEE, pp. 781–784.

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Cunningham, P. & Delany, S. J. (2020), 'k-nearest neighbour classifiers–', *arXiv preprint arXiv:2004.04523* .

Dobson, R. W. (2000), Developments in audio file formats, *in* 'ICMC'.

Gamberger, D., Lavrac, N. & Dzeroski, S. (2000), 'Noise detection and elimination in data preprocessing: experiments in medical domains', *Applied Artificial Intelligence* **14**(2), 205–223.

Gerhard, D. (2000), 'Audio signal classification: an overview', *Canadian Artificial Intelligence* pp. 4–6.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J. et al. (2020), 'Array programming with numpy', *Nature* **585**(7825), 357–362.

Harvey, E. (2017), 'Siding with vinyl: Record store day and the branding of independent music', *International Journal of Cultural Studies* **20**(6), 585–602.

Herzog, S. (2013), Efficient dsp implementation of median filtering for real-time audio noise reduction, *in* 'Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13), Maynooth, Ireland'.

Hochreiter, S. (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6**(02), 107–116.

Hunter, J. D. (2007), 'Matplotlib: A 2d graphics environment', *Computing in Science & Engineering* **9**(3), 90–95.

Kotsiantis, S. B., Zaharakis, I. & Pintelas, P. (2007), 'Supervised machine learning: A review of classification techniques', *Emerging artificial intelligence applications in computer engineering* **160**(1), 3–24.

Leal, A., Nunes, D., Couceiro, R., Henriques, J., Carvalho, P., Quintal, I. & Teixeira, C. (2018), 'Noise detection in phonocardiograms by exploring similarities in spectral features', *Biomedical Signal Processing and Control* **44**, 154–167.

Lu, L., Ge, F., Zhao, Q. & Yan, Y. (2010), A svm-based audio event detection system, *in* '2010 International Conference on Electrical and Control Engineering', pp. 292–295.

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E. & Nieto, O. (2015), librosa: Audio and music signal analysis in python, *in* 'Proceedings of the 14th python in science conference', Vol. 8, pp. 18–25.

Miller, M. (2020), 'The basics: evaluating classifiers'.
URL: *https://towardsdatascience.com/the-basics-evaluating-classifiers-b0078a097732*

Patle, A. & Chouhan, D. S. (2013), Svm kernel functions for classification, *in* '2013 International Conference on Advances in Technology and Engineering (ICATE)', IEEE, pp. 1–9.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Peoples, G. & Crupnick, R. (2014), 'The true story of how vinyl spun its way back from near-extinction', *Billboard. com* **17**.
URL: *https://www.billboard.com/articles/business/6406630/vinyl-records-comeback-music-industry-record-store-day*

Rish, I. et al. (2001), An empirical study of the naive bayes classifier, *in* 'IJCAI 2001 workshop on empirical methods in artificial intelligence', Vol. 3, pp. 41–46.

Rong, F. (2016), Audio classification method based on machine learning, *in* '2016 International conference on intelligent transportation, big data & smart city (ICITBS)', IEEE, pp. 81–84.

Sadrizadeh, S., Zarmehi, N., Asadi, E., Abin, H. & Marvasti, F. (2020), 'A fast iterative method for removing impulsive noise from sparse signals', *IEEE Transactions on Circuits and Systems for Video Technology* .

Smaragdis, P. & Raj, B. (2007), Example-driven bandwidth expansion, *in* '2007 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics', IEEE, pp. 135–138.

Stallmann, C. F. & Engelbrecht, A. P. (2016), 'Gramophone noise detection and reconstruction using time delay artificial neural networks', *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **47**(6), 893–905.

Tamatjita, E. N. & Mahastama, A. W. (2016), Comparison of music genre classification using nearest centroid classifier and k-nearest neighbours, *in* '2016 International Conference on Information Management and Technology (ICIMTech)', pp. 118–123.

Tao, C. (2021), 'How to evaluate a classification machine learning model'.
URL: *https://towardsdatascience.com/how-to-evaluate-a-classification-machine-learning-model-d81901d491b1*

Webb, G. I., Keogh, E. & Miikkulainen, R. (2010), ‘Naïve bayes.’, *Encyclopedia of machine learning* **15**, 713–714.

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

Wu, D. (2019), An audio classification approach based on machine learning, *in* ‘2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)’, IEEE, pp. 626–629.