Sound Event Detection for Smart Cars

Master Thesis

Julian Linke

Supervisors:
Univ.Prof. Dipl.-Ing. Dr.techn. Alois Sontacchi (IEM)
Dipl.-Ing. (FH) Franz Lorenz (Harman)
Dr.Ing. Dipl.-Ing. Gerald Bauer (Harman)

Graz, August 1, 2019

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Julian Linke
Graz, Juli 2019

Danksagung

Liebe Eltern, Ihr habt mir diese Ausbildung ermöglicht. Vielen herzlichen Dank für Eure beständige Mühe, Hilfe und Unterstützung. Ohne Euch wäre das alles nicht möglich gewesen.

Vielen Dank meinen Geschwistern Ferdinand, David und Smilla. Ihr hattet durch mein ganzes Studium hindurch immer ein offenes Ohr für mich.

Liebe Oma, lieber Opa, vielen Dank, dass Ihr mich während der Vollendung meiner Arbeit immer herzlich bei Euch aufgenommen habt.

Herzlichen Dank meinen Freunden und Mitstreitern Jan, Franck, Nicolai, Simon W., Simon L. und Niklas für die vielen ausgezeichneten Unterhaltungen und aufschlussreichen Diskussionen. Danke auch dir, lieber Paul, für die schönen Gespräche in Straubing.

Insbesondere möchte ich mich bei meinen Betreuern Herrn Prof. Dr. Alois Sontacchi, Herrn Dipl.-Ing. Franz Lorenz und Herrn Dr.Ing. Dipl.-Ing. Gerald Bauer für die hervorragende fachliche Unterstützung und Betreuung bedanken. Zu jeder Zeit wurde mir auf jede meiner Fragen eine geduldige und hilfreiche Antwort gegeben.

Abstract

This thesis designs classification models from the area of artificial intelligence to distinguish between urban noise and siren sounds. Audio recordings from the urban environment and audio recordings capturing specific sound events are collected in a basic data set. With audio signal processing suitable audio features for four classification approaches are extracted and selected from the obtained data environment. Different machine learning classification algorithms are discussed for two classification tasks: Two approaches for a binary classification task and one approach for a classification task with three classes are presented. The last approach compares the best binary classification solution with a deep learning classifier in the sense of transfer learning. All classification models are tested with a self-recorded validation set including car microphone recordings from the urban environment and the so-called Martinshorn.

Zusammenfassung

In dieser Arbeit werden Klassifikationsmodelle aus dem Bereich der künstlichen Intelligenz entworfen um Klänge aus der urbanen Umgebung von Sirenenklängen zu unterscheiden. Audioaufnahmen aus der urbanen Umgebung und Audioaufnahmen, die ausgewählte Schallereignisse erfassen, werden in einem Basisdatensatz gesammelt. Mit Audiosignalverarbeitung werden für vier Klassifikationsansätze geeignete Audio-Features aus der erlangten Datenumgebung extrahiert und selektiert. Im Falle von zwei Klassifikationsaufgaben werden verschiedene Klassifikationsalgorithmen aus dem Bereich Machine Learning diskutiert: Es werden zwei Ansätze für eine binäre Klassifikation und ein Ansatz für eine Klassifikation mit drei Klassen vorgestellt. Der letzte Klassifikationsansatz vergleicht das beste Ergebnis der binären Klassifikation mit einem Deep Learning Ansatz im Sinne von Transfer Learning. Alle Klassifikationsmodelle werden mit einem selbst-aufgenommenen Validierungsdatensatz, welches an oder in einem Auto aufgezeichnete Mikrofonaufnahmen von der urbanen Umgebung und dem sogenannten Martinshorn enthält, getestet.

Contents

1 Introduction

The development of self-driving cars is a recent topic in industrial and scientific institutions. The topic area includes a variety of technical fields for which further development is necessary. In the area of computer vision much research is done to make self-driving cars as safe as possible.
At this point the question arises: Instead of machine vision, are there other possibilities to ensure the optimum driving safety? Are there other information in the driving environment, that can help self-driving cars to do even better?

In the end, self-driving cars are able to *see* their environment, but they are not able to *listen* to their environment. Hence, to provide the optimum driving safety, a smart car should recognize warning signals to respond correctly to the happenings in their environment.

This thesis discusses possible warning signals (e.g. from police cars, fire trucks, ambulances, . . .) and develops state-of-the-art machine learning algorithms to especially detect the so-called *Martinshorn*, which in particular frequently occurs in Germany.

Hence, two approaches for a binary classification task, which are able to detect a *Martinshorn*, are presented. Furthermore, one approach for a classification task with three classes, which additionally detects *Environmental Sirenes*, is introduced. Another approach compares the best binary classification solution with a deep learning classifier. The so-called *class zero* is in all cases the sound of the *urban noise*, which is always present.

For all approaches and with the knowledge of audio signal processing suitable feature spaces, which should characterize the signal specifications as well as possible, are generated. The focus lies especially on audio features, which are adjusted to the *Martinshorn* or more general to sirene sounds, to provide the best possible classification results.

The basic knowledge of all applied machine learning classification algorithms are described in the second section. The used data set, information about the classes and the *Doppler effect* are explained in section 3. The machine learning approaches are described in section 4 and the deep learning approach is described in section 5.

2 Classification Algorithms

In *Machine Learning* (ML) there are different *Tasks* which can be solved by specific models. This thesis focuses on *Classification Tasks* with different kinds of audio input data (see section 3). The main idea is the training and optimization of a specific model with a training and test data set. If a model has gratifying results, the prediction with new unseen data should confirm the *generalization* of this model. Every step in a ML-chain involves many different ways for optimization and it is the developer's task to make decisions for the best possible solution for a given problem. A basic machine learning model is illustrated in Fig. 2.2.

This thesis discusses and presents the following classification algorithms: Perceptron, Logistic Regression, Naive Bayes Classifier, K-Nearest-Neighbor classifier, Support Vector Machines and Multi-Layer-Perceptron.

Before explaining the different classifiers, the following paragraphs give an overview about *learning algorithms* in respect of classification described in [1–3].

Classification Task The task T describes how a machine learning system should handle an example. This example combines specific features, that have been measured from an event (here: audio data) and that should be processed. The features are defined by a feature vector $\boldsymbol{x}$, where each entry describes another attribute.
In a classification task an algorithm which finds out, if the presented input belongs to a category (or label) k, is developed with the help of a computer. The learning algorithm usually models a function $f\colon \mathrm{R}^n \rightarrow \{1, \ldots, k\}$. The function $y = f(\boldsymbol{x})$ for example assigns a numeric code y to the feature vector $\boldsymbol{x}$, which than can be classified. Other types of models find solutions with probabilities or distances.

Performance Measure The performance measure P rates the capacities of a machine learning algorithm. A trained model is tested with unseen data to evaluate its performance. In case of a classification task one measure is called the *Accuracy* of the model. Other common performance measures are called *Precision*, *Recall* and *F1-score*. With the help of a confusion matrix common performance measurements can be derived (see Table 1). The rows of a confusion matrix correspond to the actual class label and the columns to the predicted class label. Each entry of the matrix describes the percentage of how many samples of the actual class are classified as the predicted class. In case of a binary classification task the entries are called *True Positive* (`tp`), *False Positive* (`fp`), *False Negative* (`fn`) and *True Negative* (`tn`).

The *Accuracy* of the model describes how many input samples are classified correctly by the algorithm. In case of binary classification it can be formulated as:

$$\text{Accuracy} = \frac{\texttt{tp} + \texttt{tn}}{\texttt{tp} + \texttt{tn} + \texttt{fp} + \texttt{fn}}. \tag{1}$$

The *Precision* or also *true-positive-accuracy* (`tpa`) describes how many chosen events

		Predicted Classes
Actual Classes	True Positive	False Positive
	False Negative	True Negative

Table 1 – Principle of a confusion matrix in case of binary classification. True classifications are placed on the diagonal of the matrix and false classifications elsewhere [4].

are relevant and is formulated as:

$$\text{Precision} = \texttt{tpa} = \frac{\texttt{tp}}{\texttt{tp} + \texttt{fp}}. \tag{2}$$

The *Recall* or also *true-positive-rate* (`tpr`) describes how many relevant events are chosen and can be calculated as follows:

$$\text{Recall} = \texttt{tpr} = \frac{\texttt{tp}}{\texttt{tp} + \texttt{fn}}. \tag{3}$$

The *F1-score* combines the classification performance and can be derived from the measures *Precision* and *Recall* [5]:

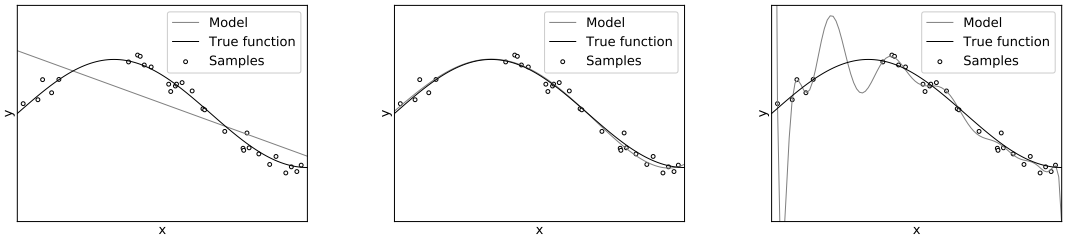
$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4}$$

This corresponds to the harmonic mean of the two performance measurements [6].

Experience The term machine learning combines *supervised* and *unsupervised* learning algorithms. In case of *unsupervised learning algorithms* the experience includes only a data set of features. Hence, the algorithm has to learn how the data set is structured by itself. In case of *supervised learning algorithms* the experience includes a data set of features and corresponding *labels* (or *targets*), which categorize each observation. Here, the target $\boldsymbol{y}$ helps the machine learning algorithm during the learning process.
In case of a *supervised learning algorithm* the feature data matrix $\boldsymbol{X}$ and the target vector $\boldsymbol{y}$ have the form

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ x_{2,1} & x_{2,1} & \dots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,m} \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \tag{5}$$

where N denotes the number of observations and m the number of features.



(a) Underfitting (b) Generalization (c) Overfitting

Figure 2.1 – An exemplary training set is fitted with three different models. The true function (black) is approximated by three models with different complexities. Underfitting occurs in case of a linear model function (a) and overfitting occurs in case of a too complex model function (c). No underfitting or overfitting occurs in case of a quadratic model function (b). [7]

Overfitting and Underfitting The main goal in machine learning lies in finding an algorithm which performs well on new unseen data (*generalization*). When evaluating the performance of the developed model with an unseen *validation set* the generalization of the model can be measured. In practice, the parameters of a machine learning model are adjusted in such a way, that the training error is kept small, but ”the expected test error is greater than or equal the expected value of [the] training error.” [1, p. 108]

On the one hand, the training error should be small and on the other also the difference between the training and the test error should be small. Those two conditions are related to the terms *underfitting* and *overfitting*. If the model is not able to keep the training error small, it is called *underfitting*. If the difference between the training error and the test error is large, it is called *overfitting*. An example in Fig. 2.1 shows how a true function is approximated by three different model functions. The first linear model function underfits the data (a), the second model function would generalize to unseen data (b) and the third model function overfits the data, because it is too complex (c).

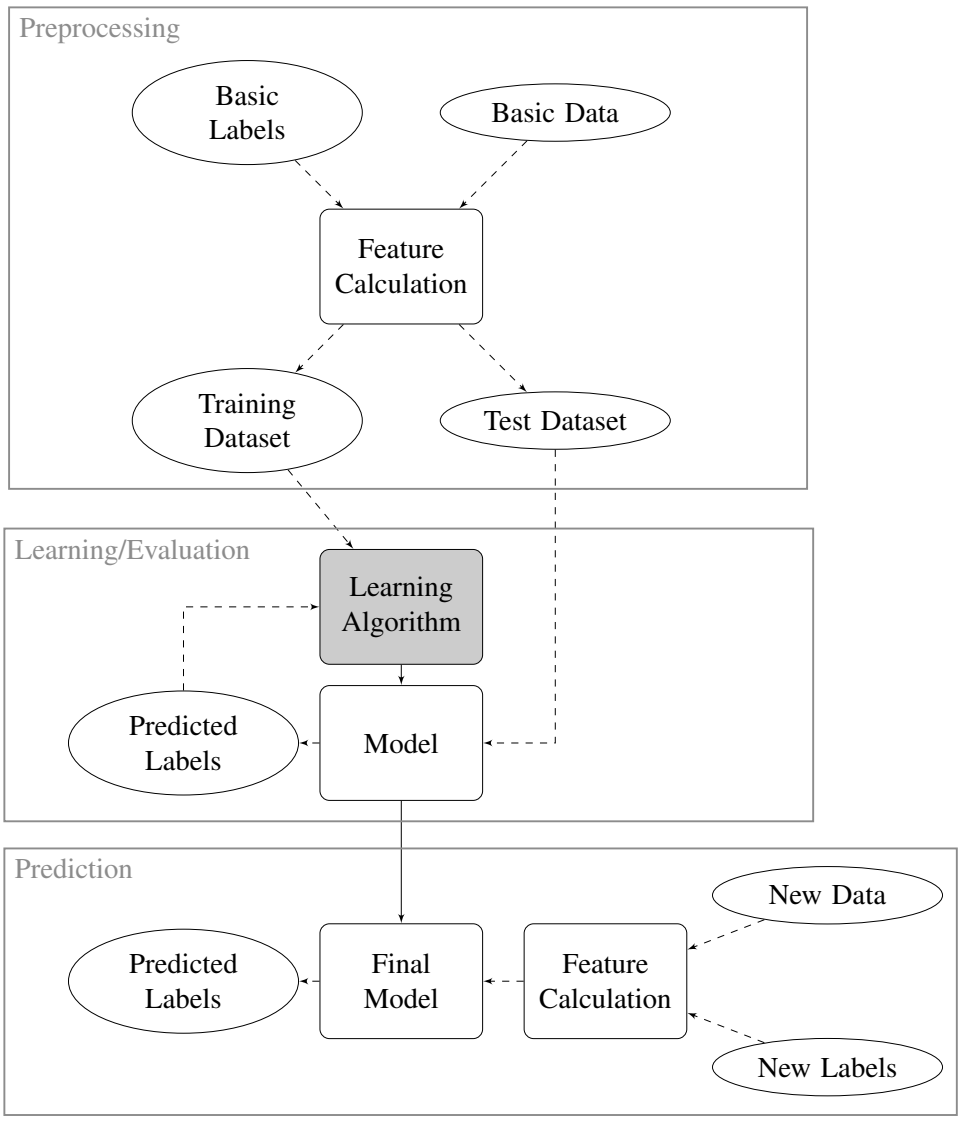


Figure 2.2 – Basic machine learning model divided in its individual blocks. Each block could involve additional processing steps: The block *Preprocessing* could for example include *Resampling*, *Feature Selection*, *Scaling* or *Dimensionality Reduction*. In case of *Cross-Validation* the feature set is divided into different training- and test-sets. The block *Learning* contains the steps *Model Selection* and *Performance Metrics* for evaluation. In the end the final model predicts labels with new unseen data and corresponding labels (*Prediction*).

2.1 Machine Learning Classifier

The following sections describe the applied machine learning classifiers. The Perceptron forms the basis for the *Multi-Layer-Perceptron* presented in section 2.2.1. The *Logistic Regression* resembles the model of a *Perceptron* but uses a *sigmoid function* for activation. The *Naive Bayes classifier* works with conditional probabilities and the non-parametric classifier *K-Nearest-Neighbor* calculates distances between the input samples. *Support Vector Machines* try to find *hyperplanes* and enable calculations in higher dimensions.

The presented algortihms are implemented with scikit-learn, which is a software package for machine learning in Python. The package involves different classification-, regression- and clustering-algorithms.

When explaining the applied classifiers in the next sections, it is partially referred to the type of implementations in the scikit-learn toolkit.

2.1.1 Perceptron

The concept of the Perceptron is based on the nervous activity, which was first described in [8]. The main idea is an *all-or-none* behavior when transmitting information, which involves a proportional logic. A schematic drawing of an artifical neuron is illustrated in Fig. 2.3. The nerve cell can be seen as a logic gate, which receives several signals at the *Dendritic Tree*, inserts them into the *cell body* and *fires* only when the arriving signal oversteps a specific threshold (see Fig. 2.4) [9]. The first mathematical description of the learning rule of a Perceptron was outlined in [10]. The rule finds optimal weight coefficients which are multiplied with the input signals. If those summed up weighted input features fulfill a certain threshold function, the neuron *fires*. Hence, the concept could be used as a classification algorithm in the sense of supervised learning.

Definition of a Perceptron: The input signals $\boldsymbol{x}_i$ and the weights $\boldsymbol{w}$ with $\boldsymbol{x}_i, \boldsymbol{w} \in \mathbb{R}^m$ and $i = [1, \ldots, N]$ input samples can be formalized as the vectors

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,m} \end{bmatrix} \tag{6}$$

and

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}. \tag{7}$$

The summed up net input $z_i = w_1 x_{i,1} + \cdots + w_m x_{i,m}$ is presented to the threshold function $\Phi(z_i)$. For a binary classification with a positive class $+1$ and a negative class

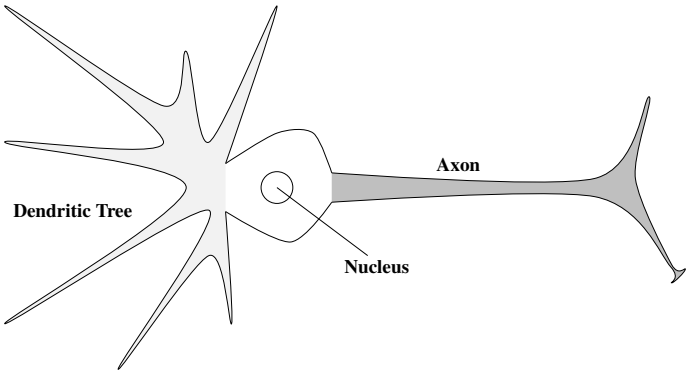


Figure 2.3 – Schematic diagram of an artificial neuron with its *Dendritic Tree* (*receiver*), the *Cell Nucleus* (*signals inserted*) and the *Axon* (*signal passing*).

-1 the output of the unit step function is

$$\Phi(z_i) = \begin{cases} 1 & \text{if } z_i \geq \theta \\ -1 & \text{otherwise.} \end{cases} \tag{8}$$

To enable a suitable decision rule an additional input and an additional weight are introduced: The weight w_0 is defined as the negative threshold $w_0 = -\theta$ and the corresponding input $x_0 = 1$. This leads to the new net input

$$\begin{aligned} z_i = \boldsymbol{w}^T \boldsymbol{x}_i = \begin{bmatrix} w_0 & w_1 & \dots & w_m \end{bmatrix} \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,m} \end{bmatrix} = \\ = w_0 + w_1 x_{i,1} + \dots + w_m x_{i,m} \end{aligned} \tag{9}$$

and therefore to the new decision function

$$\Phi(z_i) = \begin{cases} 1 & \text{if } z_i \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{10}$$

The weight w_0 is also called the *bias unit* [9].

Learning Rule of a Perceptron: There are two possible output states for an artificial neuron: It *fires* or it *fires not*. The learning rule in can be summarized as [9, 10]:

1. Initializing the weights $\boldsymbol{w}$ (0 or small random numbers)
2. Compute the output $\hat{y}_i$ for each training sample $x_{i,j}$, where $j = [1, \dots, m]$ features
3. Update the weights $\boldsymbol{w}$

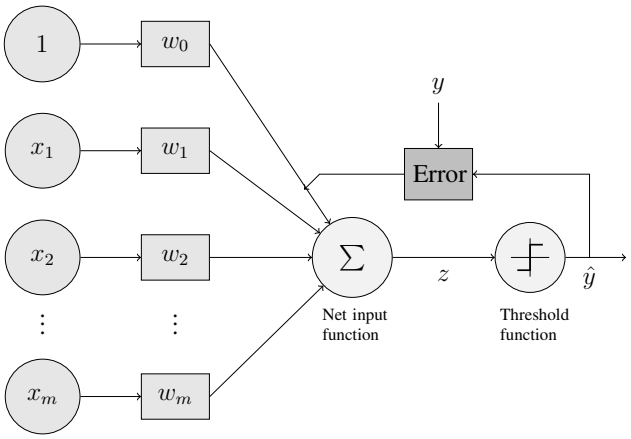


Figure 2.4 – Basic concept of the Perceptron with input features x_i and weights w_i, where $x_0 = 1$ and $w_0 = -\theta$ (*bias unit*).

The update of the weights w_j for each training sample $\boldsymbol{x}_i$ with feature values $x_{i,j}$ can be denoted as

$$w_j \coloneqq w_j + \Delta w_j, \tag{11}$$

where

$$\Delta w_j = \eta \cdot (y_i - \hat{y}_i) \cdot x_{i,j}. \tag{12}$$

The constant η describes the *learning rate*, y_i is the *true class label* and $\hat{y}_I$ the *predicted class label* of observation i. After the update of all weights Δw_j, the prediction $\hat{y}_i$ for the feature vector $\boldsymbol{x}^{(i)}$ is calculated. The Perceptron converges only if the classes are linearly separable, otherwise the weights updating would never stop. If the data is not linearly separable, a maximum number of *epochs* (number of passes over the whole training set) and (or rather or) a threshold for the number of permitted misclassifications can be set [9].

2.1.2 Logistic Regression

While the weights of the Perceptron only converge if the data is linearly separable, the binary classification with *Logistic Regression* is more powerful [9]. The *Logistic Regression* algorithm is part of *probabilistic supervised learning*, where a probability distribution $p(y|\boldsymbol{x})$ is estimated [1].

Definition of Logistic Regression: The main difference to the Perceptron is the use of an *activation function* before calculating a prediction (see Fig. 2.5). The natural logarithm of the *odds ratio* is defined as the *logit function*

$$L(p) = \ln\left(\frac{p}{1-p}\right), \tag{13}$$

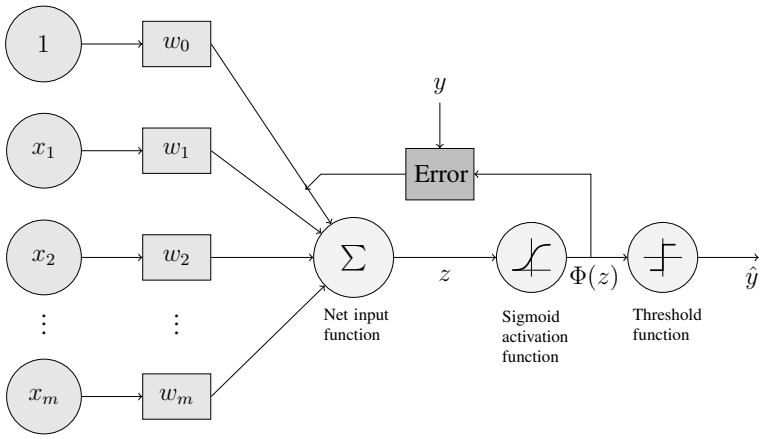


Figure 2.5 – Basic concept of the *Logistic Regression* with input features x_i and weights w_j. The activation function is a *sigmoid function*.

where p denotes the probability that an event may occur and $1 - p$ the complementary probability [11]. The input values (for one input sample) of the *logit functions* in the range from 0 to 1 (probabilities) are mapped to the entire real-number range, which leads to the following linear relationship [9]:

$$L\left[p(y_i = 1|\boldsymbol{x}_i)\right] = w_0 + w_1 x_{i,1} + \cdots + w_m x_{i,m} = w_0 + \sum_{j=1}^{m} w_j x_{i,j} = \boldsymbol{w}^T \boldsymbol{x}_i \tag{14}$$

with the conditional probability $p(y_i = 1|\boldsymbol{x}_i)$ saying that an input sample with its features $\boldsymbol{x}_i$ belongs to the class $y_i = 1$.

The inverse of the *logit function* is called the *logistic sigmoid function*

$$\Phi(z_i) = \frac{1}{1 + e^{-z_i}} \tag{15}$$

with $z_i = \boldsymbol{w}^T \boldsymbol{x}_i = w_0 + w_1 x_{i,1} + \cdots + w_m x_{i,m}$ ($x_0 = 1$ and bias w_0) [9]. In Fig. 2.6 both functions are sketched. With the *sigmoid activation function* the net input $\boldsymbol{w}^T \boldsymbol{x}_i$ is mapped to the interval $[0, 1]$ and interpreted as the probability, that this sample belongs to the class $y_i = 1$. As in the case with the Perceptron a *threshold function* can map the probabilistic outcome into a binary outcome, which is the predicted class [9]:

$$\hat{y}_i = \begin{cases} 1 & \text{if } \Phi(z_i) \geq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Learning Rule of Logistic Regression: When deriving a cost function $J(\boldsymbol{w})$ the likelihood estimator L with classes $y_i \in \{0, 1\}$ has to be maximized [12]. It is supposed, that the samples in the dataset are distributed independently:

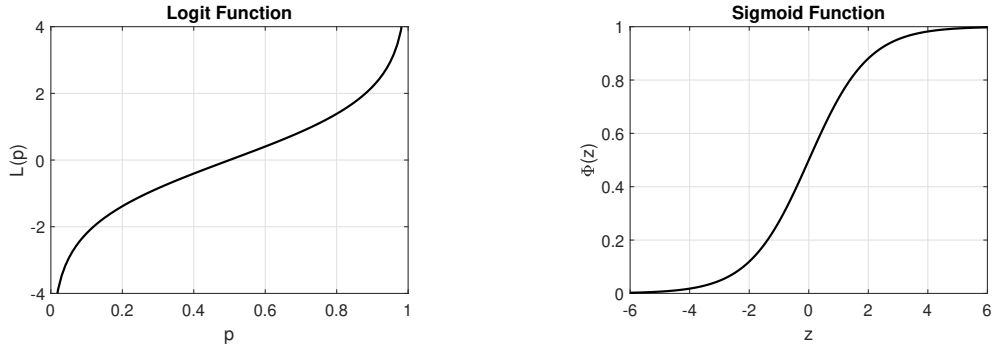


Figure 2.6 – *Logit function* $L(p)$ (left) maps probability values in the range $[0, 1]$ to the whole real-number range and its inverse the *logistic sigmoid function* $\Phi(z)$ (right).

$$L(\boldsymbol{w}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i) = \prod_{i=1}^{N} \left(\Phi(z_i)\right)^{y_i} \left(1 - \Phi(z_i)\right)^{1-y_i}. \tag{17}$$

Taking the natural logarithm of the likelihood estimator leads to the sum:

$$\ln\left(L(\boldsymbol{w})\right) = \sum_{i=1}^{N} \left[y_i \ln\left(\Phi(z_i)\right) + (1 - y_i) \ln\left(1 - \Phi(z_i)\right)\right] \tag{18}$$

This equation is equivalent to the loss function which needs to be minimized:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left[-y_i \ln\left(\Phi(z_i)\right) - (1 - y_i) \ln\left(1 - \Phi(z_i)\right)\right]. \tag{19}$$

In case of one given input sample this cost function could be formalized as:

$$J(\Phi(z), y; \boldsymbol{w}) = \begin{cases} -\ln\left(\Phi(z)\right) & \text{if } y = 1 \\ -\ln\left(1 - \Phi(z)\right) & \text{if } y = 0. \end{cases} \tag{20}$$

In Fig. 2.7 the loss function $J(\boldsymbol{w})$ for a input sample $\boldsymbol{x}$ with weights $\boldsymbol{w}$ is illustrated. The higher the output of the *activation function* $\Phi(z)$ for class $y = 1$, the lower is the cost $J(\boldsymbol{w})$ which should be minimal.

With the *Gradient Descent* method the local minimum can be found by moving along the gradients, which leads to the optimum weights $\boldsymbol{w}$:

$$\nabla J(\boldsymbol{w}) = -\sum_{i=1}^{N} \left(y_i - \Phi(\boldsymbol{x}_i)\right) \cdot \boldsymbol{x}_i, \tag{21}$$

where $\Phi(\cdot)$ is the *logistic sigmoid function* (see Eq. 15). Now, the weight update can be formalized as:

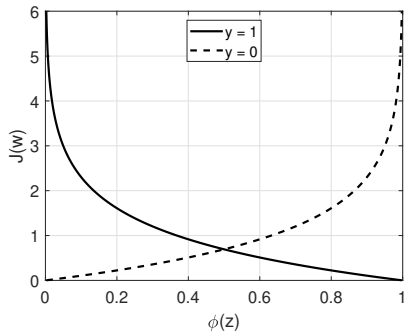


Figure 2.7 – Loss function $J(\boldsymbol{w})$ for a single sample (*feature vector*) with net input $z = \boldsymbol{w}^T\boldsymbol{x}$ for $y = \pm 1$ depending on the probabilistic output of the *sigmoid function* $\Phi(z)$.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta\boldsymbol{w} \tag{22}$$

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \eta \cdot \nabla J(\boldsymbol{w}) = \boldsymbol{w} + \eta \cdot \sum_{i=1}^{N} \left(y_i - \Phi(\boldsymbol{x}_i)\right) \cdot \boldsymbol{x}_i \tag{23}$$

Regularization Parameter for Logistic Regression There is a trade-off between high variance (overfitting) and high bias (underfitting) when designing a model [9]. If the model fits the parameter too closely and if it is not generalizing on new data it could be a sign for overfitting. To find a good solution, the *regularization parameter* λ is introduced by adding a regularization term with additional information (bias) to the cost function. This ensures, that the sum of all weights w_j are finite or rather minimal. In case of logistic regression the cost function with L2 regularization gets

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left[-y_i \ln\left(\Phi(z_i)\right) - (1 - y_i) \ln\left(1 - \Phi(z_i)\right)\right] + \frac{\lambda}{2} \sum_{j=1}^{m} w_j^2. \tag{24}$$

For a $\lambda = 0$ no penalty is added to the cost function, which could lead to overfitting. For high values of λ the weights get close to zero, which results in a simpler model with high bias [1].

2.1.3 Naive Bayes Classifier

The so-called *Bayes' rule* describes the conditional probability ([13], [1])

$$p(y|\boldsymbol{x}) = \frac{p(y)p(\boldsymbol{x}|y)}{p(\boldsymbol{x})}, \tag{25}$$

1. In scikit-learn the *regularization parameter* is introduced by the parameter $C = \frac{1}{\lambda}$. This parameter plays an important role for support vector machines, because high values of C increase the regularization strength which is equivalent to allowing a smaller margin (see section 2.1.5)

where $p(y|\boldsymbol{x})$ is called the *posterior probability* of class y with given feature vector $\boldsymbol{x}$, $p(y)$ the *prior probability* of class y, $p(\boldsymbol{x}|y)$ the likelihood of $\boldsymbol{x}$ with given class y and $p(\boldsymbol{x})$ the *prior probability* of $\boldsymbol{x}$.

The following classification rule assumes a positive and a negative class ($y = \pm 1$) and is described in [14]. The word *naive* signifies that the algorithm assumes independence between all attributes. The feature vector $\boldsymbol{x}$ is classified as the class $y = 1$ if and only if the *Bayesian classifier*

$$f_b(\boldsymbol{x}) = \frac{p(y = 1|\boldsymbol{x})}{p(y = -1|\boldsymbol{x})} \geq 1. \tag{26}$$

In case of independence of all attributes ($j = 1, \ldots, m$) with a given class variable y the likelihood

$$p(\boldsymbol{x}|y) = p(x_1, x_2, \ldots, x_m|y) = \prod_{j=1}^{m} p(x_j|y) \tag{27}$$

results in the *naive Bayesian classifier* (since $p(\boldsymbol{x})$ is constant)

$$f_{nb}(\boldsymbol{x}) = \frac{p(y = 1)}{p(y = -1)} \prod_{j=1}^{m} \frac{p(x_j|y = 1)}{p(x_j|y = -1)}. \tag{28}$$

In other words the probability that the observation $\boldsymbol{x}$ belongs to class y can also be formulated as [15]

$$p(y|\boldsymbol{x}) \propto p(y) \prod_{j=1}^{m} p(x_j|y), \tag{29}$$

which requests the *Maximum A Posteriori* (MAP) estimation

$$y_{MAP} = \arg\max_{y} p(y) \prod_{j=1}^{m} p(x_j|y) \tag{30}$$

with the predicted class y_{MAP} and learned estimated probabilities $p(x_j|y)$ [15].

The *Gaussian Naive Bayes* classifier assumes, that each attribute (or *feature*) is distributed by a *gaussian likelihood function*

$$p(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot e^{-\frac{(x_j - \mu_y)^2}{2\sigma_y^2}} \tag{31}$$

with during the learning process with the training data calculated standard deviation σ_y and mean value μ_y [15].

2.1.4 K-Nearest-Neighbor

The non-parametric classification with *K-Nearest-Neighbor* (KNN) compares the training data with the new data point by calculating the distance. The class is assigned by a majority decision by the next k-nearest points (see Fig. 2.8). The distances between the points could for example be calculated with the *Euclidean distance* or the *Mahalanobis distance*.

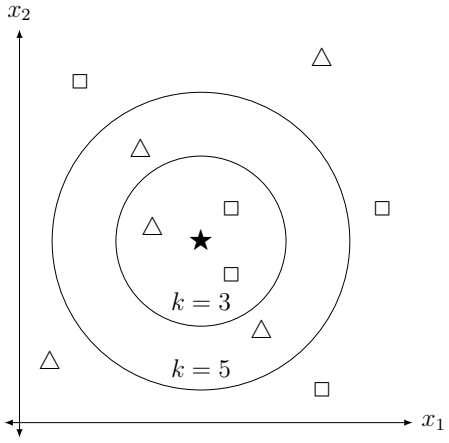


Figure 2.8 – *K-Nearest-Neighbor* classification with the classes $y = \triangle$ and $y = \square$. The circles indicate the $k = 3$ or $k = 5$ nearest neighbors to the data point $\boldsymbol{x} = \bigstar$, which needs to be classified. With $k = 3$ the classifier would choose the class $y = \square$ ($2\square > 1\triangle$) as a prediction. In case of $k = 5$ the class $y = \triangle$ ($3\triangle > 2\square$) would be the assigned class.

High values of k lead to smoother decision boundaries and ensure more robustness against noise, on the other hand higher computational costs have to be accepted. As a common practice the choice of $k = \sqrt{N}$ with N as the number of training samples has proven to work well [16]. The *neighbors-based* classification generates not a model, but simply stores samples of the training data set [17].

2.1.5 Support Vector Machines

The *Support Vector Machine* (SVM) is a widely used learning algorithm. The basic idea is the maximization of the margin, which describes the smallest distance between a separating hyperplane and specific data samples. The so called *Support Vectors* are the just described closest data points.

Case with Separable Data: The positive and negative hyperplanes can be described as

$$H_1 : \; b + \boldsymbol{w}^T \boldsymbol{x}_+ = 1, \tag{32}$$

$$H_2 : \; b + \boldsymbol{w}^T \boldsymbol{x}_- = -1. \tag{33}$$

The hyperplanes in Eq. (32) and (33) are parallel, because they have the same normal $\boldsymbol{w}$ [18]. The points for which the equalites hold (*support vectors*) lie all on one of the hyperplanes, where $\frac{|1-b|}{||\boldsymbol{w}||}$ is the distance to the origin for H_1 and $\frac{|-1-b|}{||\boldsymbol{w}||}$ the distance to the origin for H_2. A typical solution for the two-dimenstional space is depicted in Fig. 2.9.

The subtraction of those two linear hyperplane equations

$$\boldsymbol{w}^T(\boldsymbol{x}_+ - \boldsymbol{x}_-) = 2 \tag{34}$$

and a normalization of the normal vector

$$||\boldsymbol{w}|| = \sqrt{\sum_{j=1}^{m} w_j^2}. \tag{35}$$

leads to

$$\frac{\boldsymbol{w}^T(\boldsymbol{x}_+ - \boldsymbol{x}_-)}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||}. \tag{36}$$

The left side of Eq. (36) represents the distance between the two hyperplanes, which is defined as the *margin*. Hence, the summed up distances $d_+ = d_- = \frac{1}{||\boldsymbol{w}||}$ define this margin with length $\frac{2}{||\boldsymbol{w}||}$ [18]. The task requires the maximization of the term $\frac{2}{||\boldsymbol{w}||}$ on condition that the samples are classified correctly [9]. The following constraints must be fulfilled by the training data:

$$\begin{aligned} &b + \boldsymbol{w}^T \boldsymbol{x}_i \geq 1, \text{if } y_i = 1, \\ &b + \boldsymbol{w}^T \boldsymbol{x}_i \leq -1, \text{if } y_i = -1, \\ &\quad \text{where } i = 1 \ldots N. \end{aligned} \tag{37}$$

The two equations can be combined to one term:

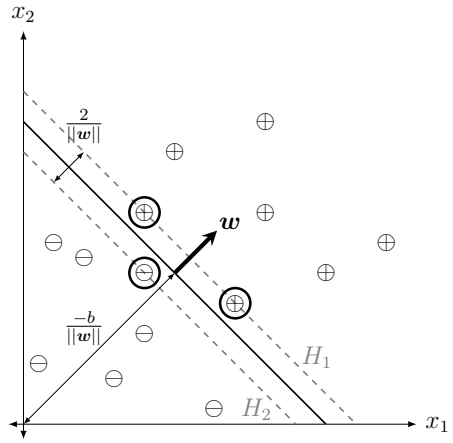


Figure 2.9 – Hyperplanes separating the points $\boldsymbol{x}_+$ ($y = 1$) and $\boldsymbol{x}_-$ ($y = -1$) in the separable case. The support vectors are marked with thick black circles.

$$y_i \cdot (b + \boldsymbol{w}^T \boldsymbol{x}_i) \geq 1, \ \forall i. \tag{38}$$

In practice, instead of maximizing the term $\frac{2}{||\boldsymbol{w}||}$, it is easier to minimize the objective function $\frac{1}{2}||\boldsymbol{w}||^2$ [9].

The problem can be solved with the *primal* Lagrangian L_P with positive Lagrange multipliers α_i:

$$L_P \equiv \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{l} \alpha_i y_i (\boldsymbol{x}_i \cdot \boldsymbol{w} + b) + \sum_{i=1}^{l} \alpha_i. \tag{39}$$

There are two set of constraints which formulate the *dual* problem [18]:

— The Lagrangian L_P has to be minimized with respect to $\boldsymbol{w}$ and b. Concurrent the derivatives of L_P with respect to the multipliers α_i have to disappear (with the constraints: $\alpha_i \geq 0$)

— The Lagrangian L_P needs to be maximized with the constraints that the gradient of L_P with respect to $\boldsymbol{w}$ and b vanishes (with the constraints: $\alpha_i \geq 0$)

To ensure that the gradient of L_P vanishes the conditions for $\boldsymbol{w}$ and b are:

$$\nabla_{\boldsymbol{w}} L_P = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x}_i \stackrel{!}{=} 0 \rightarrow \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i, \tag{40}$$

$$\nabla_b L_P = -\sum_i \alpha_i y_i \stackrel{!}{=} 0 \rightarrow \sum_i \alpha_i y_i = 0. \tag{41}$$

Substituting the conditions in Eq. (39) leads to the *dual* Lagrangian

$$L_D = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j - \sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j - \underbrace{\sum_i \alpha_i y_i b}_{=0} + \sum_i \alpha_i = \tag{42}$$

$$= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j. \tag{43}$$

The two Lagrangian L_P and L_D are formed by the same objective function $\frac{1}{2}||\boldsymbol{w}||^2$, but with different constraints. The solution can be achieved by minimizing L_P or by maximizing L_D. There exists a Lagrange multiplier α_i for every point: If $\alpha_i > 0$, the associated point is called a *support vector* lying on one of the hyperplanes (see Eq. (32), (33)). For all other points with $\alpha_i = 0$, the points lie either on one of the hyperplanes (see equality in Eq. (38)) or on the associated side of one of the hyperplanes (see inequality in Eq. (38)) [18].

The Karush-Kuhn-Tucker (KKT) conditions for the primal Lagrangian L_P result in a workable optimization problem. The conditions can be stated as [18]:

$$\frac{\partial}{\partial w_\nu} L_P = w_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0, \quad \nu = 1, \ldots, m, \tag{44}$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0, \tag{45}$$

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0, \quad i = 1, \ldots, N, \tag{46}$$

$$\alpha_i \geq 0, \quad \forall i \tag{47}$$

$$\alpha_i[y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1] = 0, \quad \forall i. \tag{48}$$

If the KKT conditions are fulfilled also the SVM problem is solved [18].

The decision rule can be formalized as

$$g(\boldsymbol{x}_{new}) = g(b + \boldsymbol{w}^T \boldsymbol{x}_{new}) = \tag{49}$$

$$= g(b + \sum_i \alpha_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x}_{new}), \tag{50}$$

where $g(z) = 1$ for $z \geq 0$ and $g(z) = -1$ otherwise.

Case with Non-Separable Data: The described algorithm gives a solution for separable data. If the data is not separable there is no manageable solution, therefore the constraints in Eq. (37) have to be adjusted when necessary. The objective function needs to be increased which can be done by introducing the slack variables ξ_i, $i = 1, \ldots, N$ to

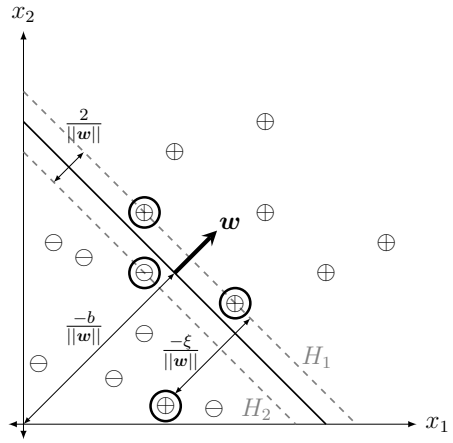


Figure 2.10 – Hyperplanes separating the points $\boldsymbol{x}_+$ ($y = 1$) and $\boldsymbol{x}_-$ ($y = -1$) in the non-separable case with consideration of the slack variable ξ.

the constraints [19]. Hence, the *soft-margin classification* with the introduced slack variable ξ_i relaxes linear constraints and enables nonlinear separation by adding the variable to the linear relations:

$$\begin{aligned} &b + \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i, \text{if } y_i = 1, \\ &b + \boldsymbol{w}^T \boldsymbol{x}_i \leq -1 + \xi_i, \text{if } y_i = -1, \\ &\quad \text{where } i = 1 \ldots N. \end{aligned} \tag{51}$$

The result of a non-separable case is illustrated in Fig. 2.10. Now, the objective function $\frac{1}{2}||\boldsymbol{w}||^2 + C \sum_i \xi_i$ needs to be minimized, where C is a constant. The variable C controls the penalty for misclassification [9]. For large values of C the margin gets smaller, which implies the permission of less misclassifications. On the other hand, small values of C allow more misclassifications. For this reason the parameter C is an important parameter for optimizing a machine learning model. In Fig. 2.11 the influence of the parameter C for a SVM with polynomial kernel (see below) is illustrated.

When maximizing the dual Lagrange from Eq. (43) with subject to $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0$ the solution is presented by $\boldsymbol{w} = \sum_{i=1}^{N_S} \alpha_i y_i \boldsymbol{x}_i$, with N_S as the number of support vectors [18]. The only difference is now the upper limit C for the Lagrange multiplier α_i.

In comparison to Eq. (39) the primal problem will now be extended to:

$$L_P = \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_i \xi_i - \sum_{i=1}^{l} \alpha_i [y_i (\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i] - \sum_{i=1}^{l} \mu_i \xi_i. \tag{52}$$

The Lagrange multipliers μ_i enforce positivity of the slack variable ξ_i [18].

The KKT-conditions, which need to be solved, can now be stated as [18]:

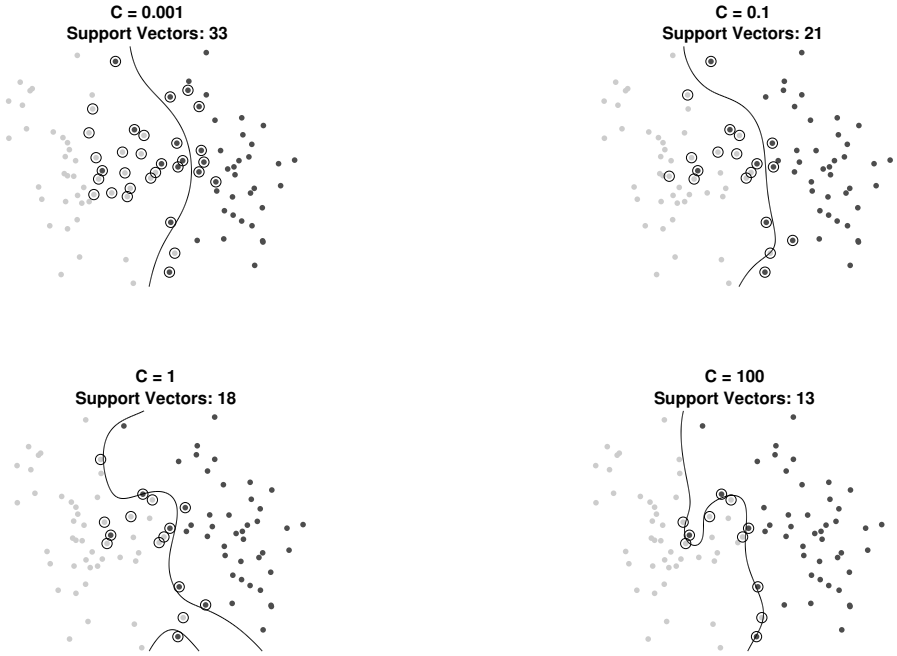


Figure 2.11 – Influence of the parameter C for a SVM with polynomial kernel (order = 5). Higher Values of C lead to a smaller margin, less misclassifications (less support vectors) and a *more complex* function. Small Values lead to a larger margin, more misclassifications (more support vectors) and a *more simple* function.

$$\frac{\partial}{\partial w_\nu} L_P = w_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0, \quad \nu = 1, \ldots, m, \tag{53}$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0, \tag{54}$$

$$\frac{\partial}{\partial \xi_i} L_P = C - \alpha_i - \mu_i = 0, \tag{55}$$

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0, \quad i = 1, \ldots, N, \tag{56}$$

$$\xi_i \geq 0, \quad \forall i \tag{57}$$

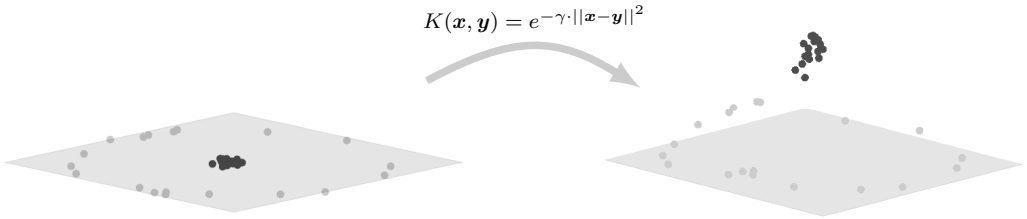
$$\alpha_i \geq 0, \quad \forall i \tag{58}$$

$$\mu_i \geq 0, \quad \forall i \tag{59}$$

$$\alpha_i[y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i] = 0, \quad \forall i \tag{60}$$

$$\mu_i \xi_i = 0, \quad \forall i \tag{61}$$

Kernel Function If the decision function (function which should classify the data as 1 or -1) is non-linear, the presence of the training data only as dot products (see Eq. (43)) can be exploited. If the data is mapped to the Euclidean space $\mathcal{H}$ with the mapping $\Phi : \mathbf{R}^d \longmapsto \mathcal{H}$, the algorithm is only based on a dot product of the data in the space $\mathcal{H}$. If



(a) Radial separable data (b) Transformed data with a radial kernel

Figure 2.12 – Exemplary *pulling apart* of radial separable data with a radial *Kernel function*. The similarities of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are *rated*. This means, that the distance or rather the accordance between those vectors is calculated. Hence, the projections are not really performed, but the images show, that the inner data points with a smaller radius can be thought of being adjusted upwards more strongly.

there is a *Kernel function*, the dot product can be solved in an arbitrary space:

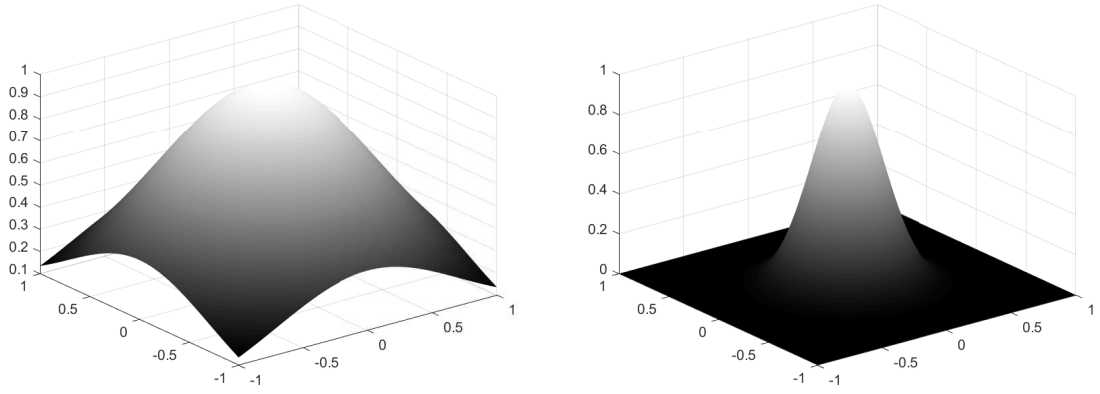
$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j). \tag{62}$$

The advantage of this *Kernel Trick* is that all previous thoughts of the algorithm hold, because the data is still separated linear, but in a different space [18].

Common kernels are linear, polynomial (degree p) or radial kernels (where $\boldsymbol{x} \cdot \boldsymbol{y}$ denotes the scalar product):

$$K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y}, \tag{63}$$
$$K(\boldsymbol{x}, \boldsymbol{y}) = (\gamma \cdot \boldsymbol{x} \cdot \boldsymbol{y} + c_0)^p, \tag{64}$$
$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma \cdot ||\boldsymbol{x}-\boldsymbol{y}||^2}. \tag{65}$$

The value γ in the radial kernel regulates the *flatness* of the multivariate Gaussian distribution (see Fig. 2.13). For higher values of γ the radial basis function (RBF) has a *sharper* peak. An example of the *pulling apart* of radial distributed data points with the radial kernel by *rating* their similarities is illustrated in Fig. 2.12.



(a) Gaussian with $\gamma = 1$.

(b) Gaussian with $\gamma = 10$.

Figure 2.13 – Bivariate Gaussian distribution with different values of γ. The value regulates the *flatness* of the RBF.

Multi-class In case of a multi-class classification with the number of classes $N_C > 2$, there are two common techniques: The *One-versus-Rest* (OvR) or the *One-versus-One* technique. In both cases a binary classifier is extended to a multi-class classifier.

With the OvR technique for each class a classifier is trained, where the specific class is labeled as the positive class and the other classes are labeled as the negative class. This leads to N_C classifiers, where the class label with the highest confidence is chosen. In case of a SVM e.g. the maximum distance to the decision boundary can be used as a confidence score.

With the *One-versus-One* technique $\frac{N_C \cdot (N_C - 1)}{2}$ binary classifiers are modeled. Therefore all combinations of two classes (e.g. 3 classes with string labels S, M and U lead to the 3 classification models, that distinguish between $\{\mathrm{S}, \mathrm{M}\}$, $\{\mathrm{S}, \mathrm{U}\}$ and $\{\mathrm{M}, \mathrm{U}\}$) build an own model. As a result, the combined classifier gives the decision by counting the number of positive predictions [20].

2.2 Deep Learning Classifier

Deep learning can be seen as a subcategory of *machine learning*. An artificial neuron (see classifier *Perceptron* in section 2.1) builds the basis of a *neural network*. Here, a set of developed algorithms should train different kinds of artificial neuron nets the most promising way [9]. Depending on the difficulty of a given classification problem, the amount of hidden layers determines the complexity of the developed function.

2.2.1 Multilayer Perceptron

The *multilayer perceptron* (MLP) is the quintessetial example of a feedforward deep learning network [1]. A MLP can be seen as a mathematical function which maps input values to output values. The more complex main function is a compilation of different simpler functions.

The following descriptions and notations of the MLP-classifier can be found in [9]. Looking at the *Logistic Regression* classifier (see section 2.1.2) and stating that the activation function $a = \phi(z)$ could also be a different function, builds the basis of a MLP-classifier. Here, the calculation steps are explained with an example neural net with one hidden layer (see Fig. 2.15). Nevertheless, the amount of hidden layers could of course be arbitrary high (network with more than one hidden layer results in a *deep artificial neural network*).

Activation Functions At first three common activation functions are presented. The first one is the *logistic sigmoid function*, which is already discussed in section 2.1.2. In the case of binary classification the logistic function maps the net input z to a probability, that it belongs e.g. to a positive or negative class. The function is defined as

$$\Phi(z) = \frac{1}{1 + e^{-z}}. \tag{66}$$

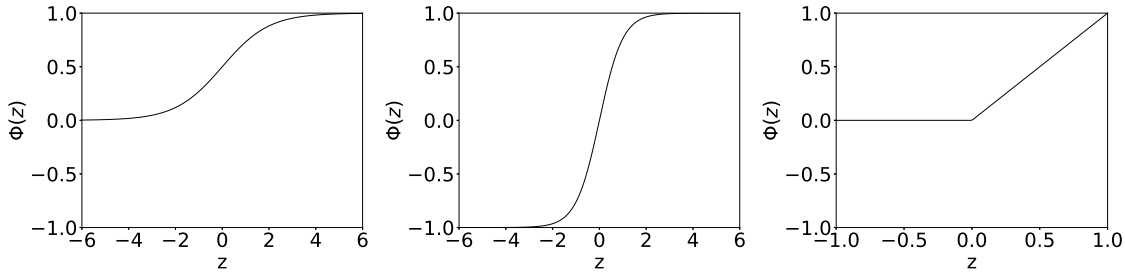
Another activation function is the hyperbolic tangent function, which is related to the logistic function. Here, the output scales the input to a wider range from $-1 \dots 1$:

$$\Phi(z) = \tanh(z). \tag{67}$$

A third activation function is the nonlinear function

$$\Phi(z) = \max(0, z) \tag{68}$$

with the name *Rectified Linear Unit* (ReLU). It maps an input smaller than zero to zero, and passes the input in case of positive values (see Fig. 2.14). This activation function can improve neural nets regarding acoustic models [21].



(a) Logistic function (b) Hyperbolic tangent function (c) Rectified Linear Unit

Figure 2.14 – Common activation functions used in artificial neurons. The logistic function maps the net input to values between 0 and 1, the hyperbolic tangent function has a similar shape, but maps the net input to values between -1 and $+1$. The ReLU-function is linear for $z \geq 0$ and 0 otherwise.

Softmax function In a binary classification task the logistic function is useful in the output layer, because it maps the net input to a probability. This holds not in case of a multiclass problem, because the sum of the output values would not be 1 anymore. Hence, the outputs could not be handled as probabilities. A *softmax function* on the other hand gives again probabilities at the output in case of a multiclass problem:

$$\Phi(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{N_C} e^{z_i}}, \quad j = 1, \ldots, N_C. \tag{69}$$

Forward Propagation The first input feature ($a_0^{(in)} = x_0 = 1$) and the first activation element in the hidden layer ($a_0^{(h)} = 1$) are defined as the bias units. All elements of the input layer are fully connected to the hidden layer and all elements of the hidden layer are fully connected to the output layer. The activation elements ${a_i}^{(l)}$ describes the i^{th} activation function in the l^{th} layer. Here, the input layer is defined with $l = 0$, the hidden layers could have $l = 1 \ldots L$ elements and the output layer is given with $l = L + 1$.

Referring again to the example network in Fig. 2.15, one hidden layer with the superscript $l = h$ and $i = 1 \ldots d$ activation elements is taken into account. The *activation functions* of the input layer describe the input features, which leads to the vector

$$\boldsymbol{a}^{(in)} = \begin{bmatrix} a_1^{(in)} \\ a_2^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} x_1^{(in)} \\ x_2^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}. \tag{70}$$

The connection between the k^{th} element of the l^{th} layer to the j^{th} element of the $(l + 1)^{(th)}$ layer is given with the weight vector $w_{k,j}^{(l+1)}$. Hence, both weight matrices can be formalized as

$$\boldsymbol{W}^{(h)} = \begin{bmatrix} w_{1,1}^{(h)} & w_{1,2}^{(h)} & \dots & w_{1,d}^{(h)} \\ w_{2,1}^{(h)} & w_{2,2}^{(h)} & \dots & w_{2,d}^{(h)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(h)} & w_{m,2}^{(h)} & \dots & w_{m,d}^{(h)} \end{bmatrix}, \boldsymbol{W}^{(out)} = \begin{bmatrix} w_{1,1}^{(out)} & w_{1,2}^{(out)} & \dots & w_{1,N_C}^{(out)} \\ w_{2,1}^{(out)} & w_{2,2}^{(out)} & \dots & w_{2,N_C}^{(out)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d,1}^{(out)} & w_{d,2}^{(out)} & \dots & w_{d,N_C}^{(out)} \end{bmatrix}. \quad (71)$$

The bias units are therefore given with

$$\boldsymbol{b}^{(h)} = \begin{bmatrix} w_{0,1}^{(h)} \\ w_{0,2}^{(h)} \\ \vdots \\ w_{0,d}^{(h)} \end{bmatrix}^T, \boldsymbol{b}^{(out)} = \begin{bmatrix} w_{0,1}^{(out)} \\ w_{0,2}^{(out)} \\ \vdots \\ w_{0,N_C}^{(out)} \end{bmatrix}^T. \quad (72)$$

In case of a binary classification task one element in the output layer is enough. If more than two classification results are needed, a generalized *One-versus-All* (OvA) multiclass classification can be performed. The one-hot encoding of three class labels could be formalized as

$$\boldsymbol{y}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^T, \boldsymbol{y}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^T, \boldsymbol{y}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T, \quad (73)$$

where e.g. each vector $\boldsymbol{y}_i$ stands for a different class (or sound), that needs to be classified. Those target vectors can then be compared to the estimated output $\boldsymbol{a}^{(out)}$. Encoding each label to a one-hot-vector enables an arbitrary amount of class labels.

Introducing the activation function in the input layer $\boldsymbol{a}^{(in)}$ (see above) with size $[m \times 1]$ and generalizing the layer to a training set with N input samples, the result is the input layer matrix

$$\boldsymbol{A}^{(in)} = \begin{bmatrix} a_{1,1}^{(in)} & a_{1,2}^{(in)} & \dots & a_{1,m}^{(in)} \\ a_{2,1}^{(in)} & a_{2,2}^{(in)} & \dots & a_{2,m}^{(in)} \\ \vdots & \vdots & \ddots & \dots \\ a_{N,1}^{(in)} & a_{N,2}^{(in)} & \dots & a_{N,m}^{(in)} \end{bmatrix} = \begin{bmatrix} x_{1,1}^{(in)} & x_{1,2}^{(in)} & \dots & x_{1,m}^{(in)} \\ x_{2,1}^{(in)} & x_{2,2}^{(in)} & \dots & x_{2,m}^{(in)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1}^{(in)} & x_{N,2}^{(in)} & \dots & x_{N,m}^{(in)} \end{bmatrix}. \quad (74)$$

with N rows as input samples and m columns as features.

The net input matrix of the hidden layer can be written as

$$\underset{N \times d}{\boldsymbol{Z}^{(h)}} = \underset{N \times m}{\boldsymbol{A}^{(in)}} \cdot \underset{m \times d}{\boldsymbol{W}^{(h)}} + \underset{1 \times d}{\boldsymbol{b}^{(h)}}. \quad (75)$$

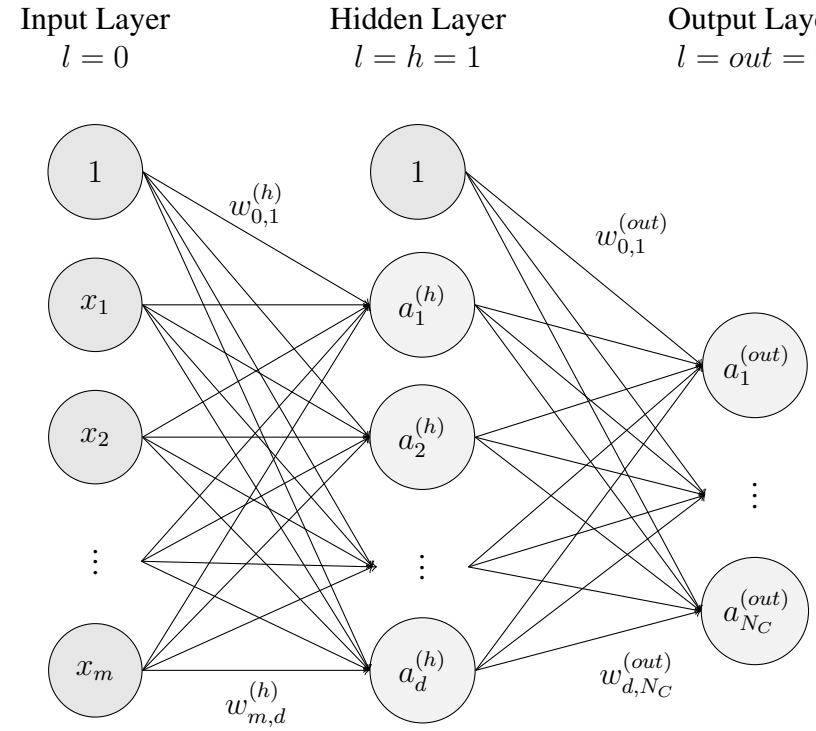


Figure 2.15 – Basic concept of the *multilayer peceptron* with m input features in the input layer, d activation functions in the hidden layer and N_C classification outputs in the output layer.

Knowing that

$$\boldsymbol{A}^{(h)} = \Phi\left(\boldsymbol{Z}^{(h)}\right) \tag{76}$$

leads to the net input matrix of the output layer

$$\underset{N\times N_C}{\boldsymbol{Z}^{(out)}} = \underset{N\times d}{\boldsymbol{A}^{(h)}} \cdot \underset{d\times N_C}{\boldsymbol{W}^{(out)}} + \underset{1\times N_C}{\boldsymbol{b}^{(out)}}. \tag{77}$$

Hence, the net output of a neural net with one hidden layer and N input samples is

$$\boldsymbol{A}^{(out)} = \Phi\left(\boldsymbol{Z}^{(out)}\right). \tag{78}$$

Logistic Cost Function The loss function of the logistic regression classifier (or the *cross entropy* [22, 23]) with the regularization term can also be formalized as

$$J(\boldsymbol{w}) = -\left(\sum_{i=1}^{N} y^{(i)} \ln(a^{(i)}) + (1 - y^{(i)}) \ln(1 - a^{(i)})\right) + \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2, \tag{79}$$

where N denotes the number of samples in the training set and m the number of weights (see section 2.1.2). Here, the superscripts denote the target values or rather the activation results for each training sample i.

In case of a multiclass classification an output vector $\boldsymbol{a}^{(out)}$ with N_C elements is returned, where each of them is compared to the target vector $\boldsymbol{y}_i$. A generalization of this cost function leads to a common cost function for a neural net, where all N_C activation units are involved:

$$J(\boldsymbol{W}) = \left(\sum_{i=1}^{N} \sum_{j=1}^{N_C} -y_j \ln(a_j^{(out)}) - (1-y_j) \ln(1-a_j^{(out)}) \right) + \frac{\lambda}{2} \sum_{l=0}^{L-1} \sum_{k=1}^{u_l} \sum_{v=1}^{u_{l+1}} (w_{k,v}^{(l+1)})^2. \tag{80}$$

The regularization term is now the sum of all quadratic weights of the network, where u_l is the number of units in layer l. With the regularization term overfitting can be avoided during training. With the backpropagation algortihm this cost function can be minimized by taking the partial derivatives with respect to each weight $w_{k,v}^{(l)}$. The weight matrix $\boldsymbol{W}$ is composed of various weight matrices (e.g. $\boldsymbol{W}^{(h)}$ and $\boldsymbol{W}^{(out)}$), with each having a different size. The size and the amount of those matrices depends on the size and the amount of the input layer, the output layer and the introduced hidden layers.

Backpropagation Algorithm The complex cost function can be calculated efficiently with the backpropagation algorithm. The idea of this algorithm is, that the calculations propagate from the right side of the network to the left side of the network. The following example shows, how the backpropagation algorithm is calculated with the given example network (see Fig. 2.15).

The error vector on the right side of the network is

$$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boldsymbol{y}, \tag{81}$$

where $\boldsymbol{y}$ stands for the true class label of one given feature vector. The error vector of the hidden layer can then be calculated with

$$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)} \left(\boldsymbol{W}^{(out)} \right)^T \odot \frac{\delta \Phi(\boldsymbol{z}^{(h)})}{\delta \boldsymbol{z}^{(h)}}. \tag{82}$$

The gradients of the cost function with respect to the weights and the gradients of the cost function with respect to the bias units at the outputs are

$$\frac{\delta}{\delta \boldsymbol{W}^{(out)}} J(\boldsymbol{W}) = \boldsymbol{a}^{(h)} \boldsymbol{\delta}^{(out)}, \tag{83}$$

$$\frac{\delta}{\delta \boldsymbol{b}^{(out)}} J(\boldsymbol{W}) = \boldsymbol{\delta}^{(out)}. \tag{84}$$

The gradients of the cost function with respect to the weights at the hidden layer are

$$\frac{\delta}{\delta \boldsymbol{W}^{(h)}} J(\boldsymbol{W}) = \boldsymbol{a}^{(in)} \boldsymbol{\delta}^{(h)}, \tag{85}$$

$$\frac{\delta}{\delta \boldsymbol{b}^{(h)}} J(\boldsymbol{W}) = \boldsymbol{\delta}^{(h)}. \tag{86}$$

Note, that the errors ($\boldsymbol{\delta}^{(out)}$, $\boldsymbol{\delta}^{(h)}$) can also be expressed as matrices by generalizing them to the number of input samples N. Hence, the gradients of the weights could be calculated with matrix operations by executing the dot product between the transposed activation layer matrices ($\boldsymbol{A}^{(in)}$, $\boldsymbol{A}^{(out)}$) and those error matrices.

After adding the regularization term only to the gradients of the weights

$$\frac{\delta}{\delta \boldsymbol{W}^{(l)}} J(\boldsymbol{W}) \leftarrow \frac{\delta}{\delta \boldsymbol{W}^{(l)}} J(\boldsymbol{W}) + \lambda \boldsymbol{W}^{(l)}, \tag{87}$$

the gradients of the weight matrices and the bias vectors can be updated with an optimization function $g(\cdot)$:

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} + g\left(\frac{\delta}{\delta \boldsymbol{W}^{(l)}} J(\boldsymbol{W})\right) \tag{88}$$

$$\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} + g\left(\frac{\delta}{\delta \boldsymbol{b}^{(l)}} J(\boldsymbol{W})\right). \tag{89}$$

Stochastic gradient descent In case of stochastic gradient descent the weights and the biases for each training observation are updated with the following optimization rule:

$$g\left(\frac{\delta}{\delta \boldsymbol{W}^{(l)}} J(\boldsymbol{W})\right) = -\eta \cdot \frac{\delta}{\delta \boldsymbol{W}^{(l)}} J(\boldsymbol{W}), \tag{90}$$

$$g\left(\frac{\delta}{\delta \boldsymbol{b}^{(l)}} J(\boldsymbol{b})\right) = -\eta \cdot \frac{\delta}{\delta \boldsymbol{b}^{(l)}} J(\boldsymbol{W}). \tag{91}$$

$$\tag{92}$$

Hence, this method finds the optimal coefficients by moving in the direction of the steepest descent of the error plane, where the parameter η describes the learning rate. For high values of η the new weights could go beyond the global minimum, which could result in finding no optimal solution. In case of small values of η the learning process could take a very long time.

Adam *Adam* is a stochastic optimization rule presented in [24]. For simplicity the weight matrices $\boldsymbol{W}^{(l)}$ and the bias vectors $\boldsymbol{b}^{(l)}$ are indicated as the parameter vector $\boldsymbol{\theta}$. *Adam* is similar to the idea of momentum, where a partition of the previous weight update vector is added to the gradient descent update rule [25]:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta \Delta\boldsymbol{\theta}. \tag{93}$$

Momentum tends to accelerate the gradients into the right direction, because it prevents oscillation. Therefore, the gradient of the loss function gets estimated more accurate. With *Adam* the mean $\boldsymbol{m}$ and the variance $\boldsymbol{v}$ is estimated by past gradients. To begin with, the two moment vectors $\boldsymbol{m}$ and $\boldsymbol{v}$ and a time step t are initialized to zero. During training, the time step t is incremented:

$$t \leftarrow t + 1. \tag{94}$$

Then, the *biased first moment estimate* and the *biased second raw moment estimate* are updated like this:

$$\boldsymbol{m} \leftarrow \beta_1 \cdot \boldsymbol{m} + (1 - \beta_1) \cdot \frac{\delta}{\delta \boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{95}$$

$$\boldsymbol{v} \leftarrow \beta_2 \cdot \boldsymbol{v} + (1 - \beta_2) \cdot \left(\frac{\delta}{\delta \boldsymbol{\theta}} J(\boldsymbol{\theta}) \right)^2. \tag{96}$$

After that, the *bias-corrected first moment estimate* and the *bias-corrected second raw moment estimate* are computed:

$$\boldsymbol{m} \leftarrow \frac{\boldsymbol{m}}{(1 - \beta_1^t)}, \tag{97}$$

$$\boldsymbol{v} \leftarrow \frac{\boldsymbol{v}}{(1 - \beta_2^t)}. \tag{98}$$

This leads to the update rule

$$g\left(\frac{\delta}{\delta \boldsymbol{\theta}} J(\boldsymbol{\theta}) \right) = -\eta \cdot \frac{\boldsymbol{m}}{\sqrt{\boldsymbol{v}} + \epsilon}. \tag{99}$$

The authors in [24] propose the choice of the parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\gamma = 0.001$ and $\epsilon = 10^{-8}$.

Stochastic, Batch and Mini-Batch learning A comparison of *stochastic* and *batch* learning is presented in [26]. In case of batch learning the updates of the parameters are calculated by evaluating the entire training set N_{train}. With stochastic learning the updates

are calculated with a single feature vector $\boldsymbol{x}$. With *mini-batch* training N_{mb} feature vectors from the randomized training set are taken into account, resulting in $\frac{N_{train}}{N_{mb}}$ parameter updates per epoch.

Epoch If the entire training set passed through the neural net it is called an *epoch*. Generally the training error decreases over time (amount of epochs), where the validation error begins to rise again from a certain time point [1]. In this case an *early stopping* can be initiated: The training process stops, when the validation error is not improving anymore.

3 Data Generation

The main focus of this thesis lies on the distinction of the warning signals *Martinshorn* (M) and *Environmental Sirene* (S). On the other hand those signals should always be distinguished from the class *Urban Noise* (U). However, data for other classes (*Accidents*, *Honks* and *Wail/Yelp*) were also made available for possible further development. In section 3.2 descriptions for the different classes are given. Section 3.3 describes a self-recorded validation data set at a fire station with microphones fixed on specific positions on a car.

3.1 Basic Dataset

Basic Dataset: The main dataset consists of 1970 wave-files, where each file has a different duration. The files were arranged with the software REAPER [27] and then saved in respective class-subfolders. In Table 2 a listing of the basic data is provided.

Class	Duration	Quantity
Accidents	00:42:24 h	979
Environmental Sirene	00:57:49 h	213
Honks	01:06:02 h	245
Martinshorn	01:00:29 h	183
Wail/Yelp	00:49:10 h	322
Urban Noise	01:12:55 h	28

Table 2 – Listing of the class names with the duration and the quantity of the consisting wave-files of the created basic dataset.

The basic dataset is a compilation of both, clean and undisturbed sound-files and noisy sound-files. Noisy sound-files are basically superimposed by environmental noise composed of traffic, people or bird sounds.

Working Dataset: The basic dataset gets divided into a working dataset, where 70% of the dataset is used for training and testing and 30% of the dataset is used for testing as a validation data set. Table 3 shows for each class the amount of data which is used for training and the amount of data which is used for validation [2].

2. Note, that the validation data set for class U has only 1 file, but on the other hand there is an additional self-recorded validation data set and also a k-fold cross validation with the entire training set is performed for each classifier

Class	Training (t)	Training (N_f)	Validation (t)	Validation (N_f)
Accidents	00:29:41 h	686	00:12:42 h	293
Environmental Sirene	00:40:39 h	150	00:17:10 h	63
Honks	00:46:40 h	172	00:19:21 h	73
Martinshorn	00:42:40 h	125	00:17:48 h	58
Wail/Yelp	00:34:31 h	224	00:14:39 h	98
Urban Noise	00:46:07 h	6	00:15:37 h	1

Table 3 – Working dataset with listed class names and the duration t and quantity N_f of each respective training- and validation-dataset (30%).

The developed machine learning models for the classes S, M and U show, that the amount of data is adequate for generalization. However, in this thesis the developed features of the best machine learning approach are also used for a deep learning approach (classification with a neural net) to give a comparison.

3.2 Classes

The particular data for each class is downloaded from online platforms [28],[29]. This chapter gives basic information about each class and describes their signal characteristics.

3.2.1 Description

Accidents It could be useful for drivers and it is necessary for smart cars to react when an accident is recognized. Sounds are taken from YouTube-videos recorded by microphones of *Dash-cams* (accidents on the road), customary cameras and mobile phones (mostly accidents on racetracks) or professional equipment (crash tests). Accidents are temporary incidents, which leads to predominant short durations for each sound file.

Environmental Sirene *Environmental Sirenes* in Germany are not consistent since the abolition of the "Zivilschutznetz" in 1992 [30]. Each state in Germany can decide for itself which signals they use and if it the signal is even triggered. Still, there are five types of signals which are commonly in use [31]:

— *Warning of the population*: Rising and falling tone sequence (RF) with an overall duration of one minute. Each RF lasts approximately 3s
— *ABC-alarm*: RF with a specific sequence: [12s RF, 12s break, 12s RF, 12s break, 12s RF]. This sequence is repeated after a 30 second break, which leads to an alarm with a duration of 150s in total.
— *All-clear signal*: Consistent permanent tone with approximately 1s attack and 1s release (PT) with an overall duration of one minute.
— *Fire alarm*: PT with an overall duration of one minute and a specific sequence: [12s PT, 12s break, 12s PT, 12s break, 12s PT].

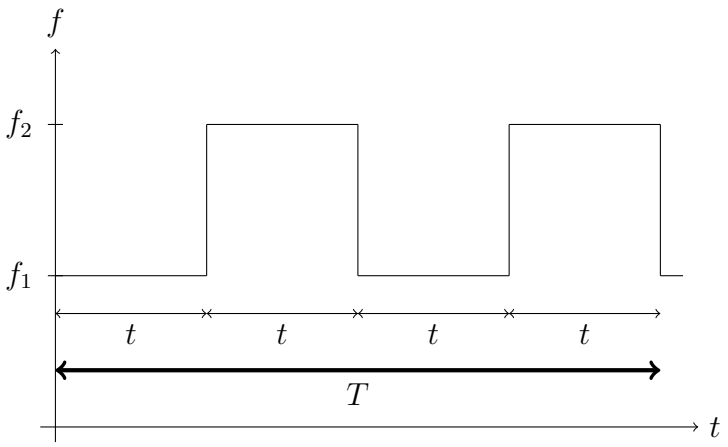


Figure 3.1 – Obligated sequence of sounds with *total sequence time* T (within (3 ± 0.5) s) of the sirene *Martinshorn* formalized in [33].

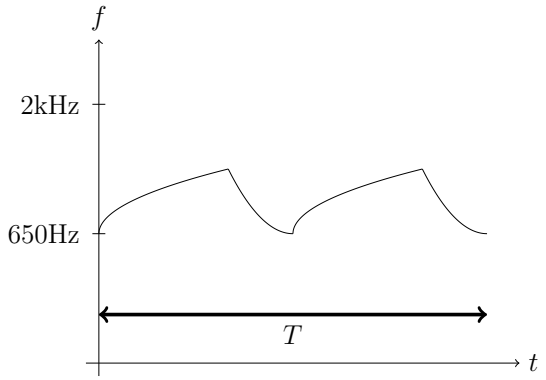
— *Practice alarm*: PT with a duration of 5s or 12s seconds.

The collected sound material from YouTube-videos mostly contains testings with different types of sirenes. Most of the data captures a sirene sequence which similar to the *fire alarm*. Therefore, a large part of the training data of class S involves a consistent permanent tone.

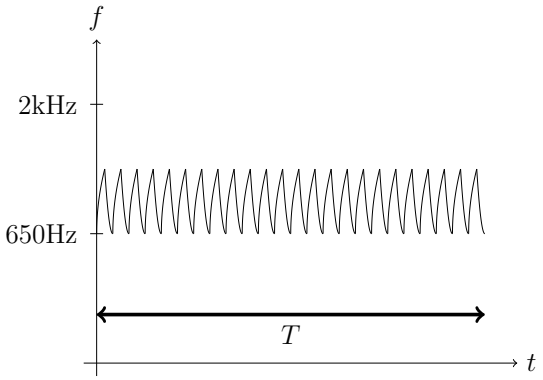
Honks Different kinds of vehicle honks (e.g. from cars or fire trucks) are taken into account. In Germany, each motor vehicle has to have at least one warning device. Honks or horns with a consistent fundamental frequency or harmonic chord are permitted [32].

Martinshorn The *Martinshorn* or in German the so called *Folgetonhorn* is an audible warning device and an approved *special signal*. An audible warning device produces a sequence of two sounds with two different fundamental frequencies (see Fig. 3.1). The fundamental frequencies have to have a frequency ratio of $\frac{4}{3}$ (musical interval of a fourth). The fundamental frequencies need to be within the bounds of the frequencies 360 Hz and 630 Hz. The *total sequence time* T of a signal cycle (each fundamental frequency has been played two times) has to have (3 ± 0.5) s [33]. This means that each sound could last in the range from 0.625s to 0.875s.

Wail/Yelp The class *Wail/Yelp* includes signals, that are more widespread throughout the world. In this case the dataset contains two in the United States widely used types of signals, which are characterized by their cycle rate and fundamental frequency range [34]. The tones of both, the *Wail-Signal* and the *Yelp-Signal*, increase and decrease with a specific continuous rate, where the *Yelp-Signal* is actually a speeded up version of the *Wail-Signal* [35]. The frequency and cycle rate requirements according to the Society of Automotive Engineers (SAE) J1849 (recommended practice) for both signals are depicted in Table 4 [34].



(a) Sketch of sirene sound *Wail* with $T = 6$s.



(b) Sketch of sirene sound *Yelp* with $T = 6$s.

Parameter	Wail-Signal	Yelp-Signal
Cycles per minute (lower limit)	10	150
Cycles per minute (upper limit)	30	250
Minimum range of fundamental frequency	850 Hz	850 Hz
Minimum fundamental frequency	650 Hz	650 Hz
Maximum fundamental frequency	2000 Hz	2000 Hz

Table 4 – Frequency and cycle rate requirements for the *Wail-Signal* and the *Yelp-Signal* according to SAE J1849 [34].

Urban Noise The class *Urban Noise* (U) includes all sorts of sounds, which could be perceived in a real world environment. To give a good representation of the urban environment, the attempt was made to cover as many as possible sounds, that could emerge. It involves for example urban sounds, traffic sounds, rain, speech, cars and birds. It was ensured that no *sirene-like* sounds are involved in this class.

3.2.2 Doppler Effect

When considering the *Doppler effect* regarding e.g. a sirene from an emergency vehicle, there are three possible situations during recognition:

— observer stationary, signal source in motion
— observer in motion, signal source stationary
— observer and signal source in motion (movement toward or away from one another)

A general description of the relations of the frequency behaviour is

$$f_{obs} = f_{source} \frac{c \pm v_{obs}}{c \mp v_{source}}, \tag{100}$$

where f_{obs} is the perceived frequency at the observer position, f_{source} the signal source frequency, v_o the velocity of the observer, v_{source} the velocity of the signal source and c

the speed of sound.

In Tables 5, 6 and 7 the impacts of the *Doppler effect* on the fundamental frequencies of the *Martinshorn* in different situations are demonstrated. Tables 6 and 7 illustrate how the lower and upper bounds of a bandpass region could change. The possible lower or rather upper frequency *limit* of the related fundamental frequency with a maximum velocity of 90 $\mathrm{km\,h^{-1}}$ depending on the driving direction is 311 Hz (for $f_1 = 360$ Hz) or 730 Hz (for $f_2 = 630$ Hz).

v_{source}	f_{source}	towards observer	away from observer	f_{source}	towards observer	away from observer
0 $\mathrm{km\,h^{-1}}$	360 Hz	360 Hz	360 Hz	630 Hz	630 Hz	630 Hz
10 $\mathrm{km\,h^{-1}}$	360 Hz	363 Hz	357 Hz	630 Hz	635 Hz	625 Hz
30 $\mathrm{km\,h^{-1}}$	360 Hz	369 Hz	351 Hz	630 Hz	646 Hz	615 Hz
50 $\mathrm{km\,h^{-1}}$	360 Hz	375 Hz	346 Hz	630 Hz	657 Hz	605 Hz
70 $\mathrm{km\,h^{-1}}$	360 Hz	382 Hz	341 Hz	630 Hz	668 Hz	596 Hz

Table 5 – Impacts of the *Doppler effect* on both possible fundamental frequencies $f_1 = 360$Hz and $f_2 = 630$Hz with the source in motion and $v_{obs} = 0$ $\mathrm{km\,h^{-1}}$ (e.g. waiting at junctions).

v_{obs} \ v_{source}	10 $\mathrm{km\,h^{-1}}$	30 $\mathrm{km\,h^{-1}}$	50 $\mathrm{km\,h^{-1}}$	70 $\mathrm{km\,h^{-1}}$	90 $\mathrm{km\,h^{-1}}$
10 $\mathrm{km\,h^{-1}}$	354 Hz	348 Hz	342 Hz	337 Hz	331 Hz
30 $\mathrm{km\,h^{-1}}$	349 Hz	343 Hz	337 Hz	332 Hz	326 Hz
50 $\mathrm{km\,h^{-1}}$	343 Hz	337 Hz	332 Hz	326 Hz	320 Hz
70 $\mathrm{km\,h^{-1}}$	338 Hz	332 Hz	327 Hz	321 Hz	315 Hz
90 $\mathrm{km\,h^{-1}}$	333 Hz	327 Hz	322 Hz	316 Hz	**311** Hz

Table 6 – Impacts of the *Doppler effect* on the lowest possible fundamental frequency $f_1 = 360$Hz with the source and the observer moving away from one another.

v_{obs} \ v_{source}	10 $\mathrm{km\,h^{-1}}$	30 $\mathrm{km\,h^{-1}}$	50 $\mathrm{km\,h^{-1}}$	70 $\mathrm{km\,h^{-1}}$	90 $\mathrm{km\,h^{-1}}$
10 $\mathrm{km\,h^{-1}}$	640 Hz	651 Hz	661 Hz	672 Hz	682 Hz
30 $\mathrm{km\,h^{-1}}$	651 Hz	662 Hz	672 Hz	683 Hz	693 Hz
50 $\mathrm{km\,h^{-1}}$	662 Hz	673 Hz	684 Hz	694 Hz	705 Hz
70 $\mathrm{km\,h^{-1}}$	674 Hz	685 Hz	696 Hz	706 Hz	717 Hz
90 $\mathrm{km\,h^{-1}}$	686 Hz	697 Hz	708 Hz	719 Hz	**730** Hz

Table 7 – Impacts of the *Doppler effect* on the highest possible fundamental frequency $f_2 = 630$Hz with the source and the observer moving towards each other.

Depending on the relative condition of movement, the range of the fundamental frequencies can be shifted, but the ratio $\frac{f_2}{f_1}$ is independent of this shift, because the shift affects both frequencies (f_1 and f_2) simultaneously.

3.3 Validation Data: Car Microphone Recordings

The final algorithm should be able to classify sirens while driving in a car. For this reason another validation set was recorded at a fire station with 5 microphones fixed at specific positions on or rather in a car: 2 microphones were placed in the engine compartment. From the driver's perspective 1 microphone was placed midway on the left side of the roof of the car, 1 microphone was placed on the lateral right side of the car (next to the door pull of the passenger door) and 1 microphone was placed at the back of the car (below the number plate).

Table 8 shows the durations and the quantity of the recorded classes M and U. There was a difficulty on one classification task with the recorded class U, which is discussed in section 4.2.5: The engine compartment of the ventilation of the car produced a *sirene-like* sound, which tended to be confused with class S.

Class	Validation (t)	Validation (N)
Martinshorn	00:25:20 h	24
UrbanNoise	00:23:05 h	5

Table 8 – Working dataset with listed class names, the duration t and quantity N of the self-recorded validation-dataset.

4 Machine Learning Approach

The different machine learning classification tasks are described in the following sections. At first a binary classification of the classes *Martinshorn* (M) and *Urban Noise* (U) is presented, which constitutes the main part of this thesis. Secondly classification algorithms for the three classes *Martinshorn*, *Environmental Sirene* (S) and *Urban Noise* are developed. In the end the new findings from the three-class-model are applied again to the binary classification task, which lead to a improvement of the results.

4.1 Binary Classification

This section shows the development of an analytic model, which is trained by specific conclusive features. At first the selected features and the resulting feature space are presented, then the feature selection is described and in the end the classification results are presented.

4.1.1 Preprocessing

When looking at the spectrogram of the exemplary audio-file (see Fig. 4.4) of class M it should be noted, that the *total sequence time T* of this signal cycle (playback of approximately 4 sounds or rather fundamental frequencies) is about 3s. To ensure that the entire signal cycle is captured, the length of a sound snippet for which a feature matrix (see section 4.1.3) is calculated is set to 4s and each wave-file with its different duration is cut to a multiple of 4. If a signal is too short, it is removed from the data set.
This consideration also takes into account, that a signal from the class S may in the case of shorter durations not be distinguishable from a signal from the class M. A distinction of those two classes requires larger time frames.
All files are *resampled* to the sampling frequency $f_s = 5000\text{Hz}$. The main reason for this is the focus on the signal information in the frequency range below the Nyquist frequency of $\frac{f_s}{2} = 2500\text{Hz}$ and of course the reduction of the computational costs. Hence, an analysis window with length of 4s leads to $N_{AW} = 4\text{s} \cdot f_s = 20000$ samples.
After that, an *A-weighting* is performed to each signal to reduce the energy in the lower frequency range and to emphasize the important frequency band in the area of 1000Hz. The preprocessing steps are illustrated in Fig. 4.1.

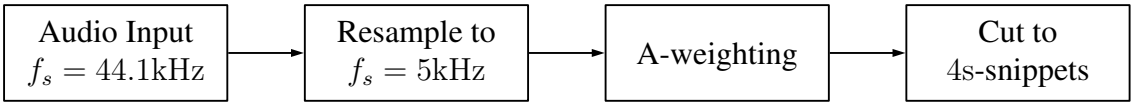


Figure 4.1 – Preprocessing of each audio input with resampling, A-weighting and cutting to snippets with length of 4s. Each input signal has a different length and belongs either to class U or to class M. The audio files are divided into blocks, where each has a length of 4s.

4.1.2 Conclusive Audio Features

This section describes the calculations of the conclusive audio features. For this analytic approach 9 audio features, which may include enough relevant information to classify the described sound events, are taken into account (see Table 9).

The majority of the calculated features has a higher time resolution than the duration of a snippet. For this reason the features are transferred as matrices to the feature space. The FFR is the only feature, which is calculated over the entire snippet. Hence, the result is a scalar, which describes one property of a snippet. Therefore a specific amount of copies of this calculated value is transferred to the feature space (see section 4.1.3).

Note, that the term *number of samples* in some descriptions of the x-label of all subsequent figures does not relate to the *sampled data*, but to the number of *training samples*, which is often used in the ML-literature.

Symbol	Description
STER	short-time energy ratio
LSTFT	logarithmic short-time fourier transform
C	chromagram
CE	chromatic entropy
MELS	mel-scaled spectrogram
MFCC	mel frequency cepstral coefficients
FFR	fundamental frequency ratio

Table 9 – Calculated audio features with corresponding symbol and their descriptions.

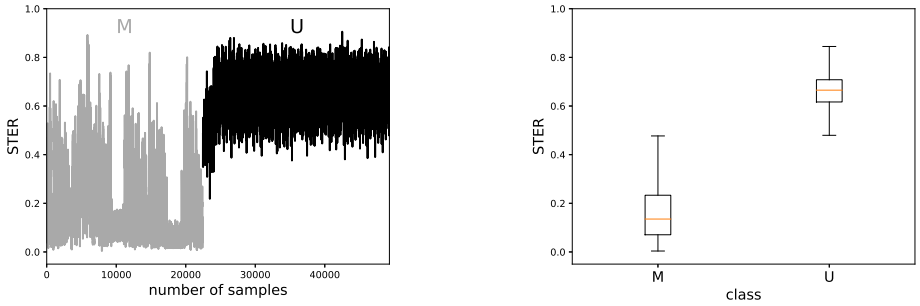
Short-Time Energy Ratio The short-time energy ratio (STER) is the short-time energy in a *bandpass-region* in relation to the short-time energy in a *baseband-region*. The short-time energy (STE) of a signal's frame with length N_{ste} is the assessed root mean square

$$\mathrm{STE} = \sqrt{\frac{1}{N_{ste}} \sum_{n=0}^{N_{ste}-1} x^2[n]}. \tag{101}$$

With the presumed *bandpass-filtered* signal $x_{bp}[n]$ the short-time energy ratio can be calculated by dividing the energy of the frequency band by the entire energy of the signal:

$$\mathrm{STER} = \frac{\sqrt{\frac{1}{N_{ste}} \sum_{n=0}^{N_{ste}-1} x_{bp}^2[n]}}{\sqrt{\frac{1}{N_{ste}} \sum_{n=0}^{N_{ste}-1} x^2[n]}}. \tag{102}$$

The frequency response with different orders of the used *Butterworth Band-pass filter* is illustrated in Fig. 4.3. The *bandpass-region* lies between $280\mathrm{Hz}$ and $670\mathrm{Hz}$, because the



(a) STER (b) Boxplot of STER

Figure 4.2 – Resulting short-time energy ratio (STER) and respective boxplots of the training set with classes M and U.

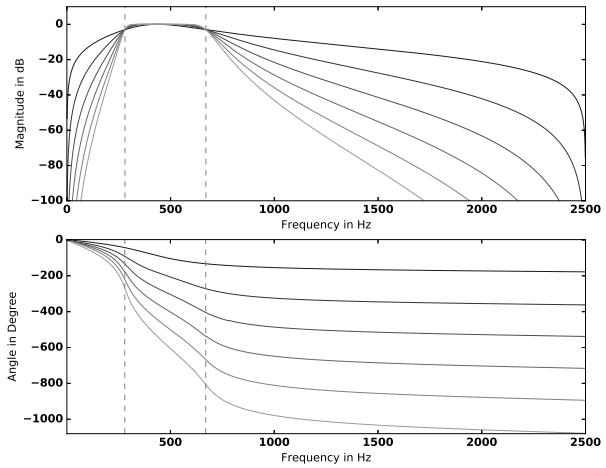


Figure 4.3 – Frequency response of the used *Butterworth Band-pass filter* with the orders 1 (black) to 6 (grey). The grey dashed lines show the *bandpass-region* in which the two fundamental frequencies of class M are located.

fundamental frequencies of a *Martinshorn* usually are located within those bounds [3]. For the order of the filter a value of 5 was chosen. Here, the frame length is set to $N_{ste} = 1024$. Fig. 4.2 shows the calculated STER of the training set with the two classes. The boxplot in Fig. 4.2b exhibit clearly separable median values of both classes.

Logarithmic Short-time Fourier Transform The energy distribution across the frequency range is represented by the spectrum [36]. For a digital signal $x[n]$, the spectrum is the discrete fourier transform

3. Note, that the *bandpass-region* was not precisely adjusted to the region, in which the fundamental frequencies could occur due to the *Doppler-effect*. One reason is that in the available data sets the fundamental frequencies primary lie in this region. Furthermore better results were achieved with those frequency limits.

$$X[k] = \text{DFT}\left(x[n]\right) = \sum_{n=0}^{N_{fft}-1} x[n] e^{-j\frac{2\pi nk}{N_{fft}}} = |X[k]| e^{-j\varphi[k]} = \text{Re}(X[k]) + j\,\text{Im}(X[k]). \tag{103}$$

The transformed signal $X[k]$ with its frequency component k is complex-valued and characterized by its magnitude $|X[k]|$ and phase $\varphi[k]$. The absolute value or magnitude at a frequency bin index k [37] is

$$|X[k]| = \sqrt{\text{Re}(X[k])^2 + j\,\text{Im}(X[k])^2}, \tag{104}$$

and the corresponding phase, which gives the phase spectrum, is

$$\varphi[k] = \arctan\left(\frac{\text{Im}(X[k])}{\text{Re}(X[k])}\right). \tag{105}$$

The magnitude spectrum $|X[k]|$ is often plotted on a decadic logarithmic scale to clarify relationships for lower values:

$$S_{dB}[k] = 20 \cdot \log_{10}|X[k]|. \tag{106}$$

In comparison with the *power spectrum* $S[k] = |X[k]|^2$, the logarithmic representation of the magnitude as a feature lead to better results in the use case. For this reason only the logarithmic spectrum is considered as a feature in the preselection of this approach.

The *short-time Fourier transform* (STFT) splits the data into usually overlapping frames. To avoid the *leakage effect* (discontinuities at segment boundaries), each frame m is multiplied with a N_{fft}-length *Hann window* $w[n]$:

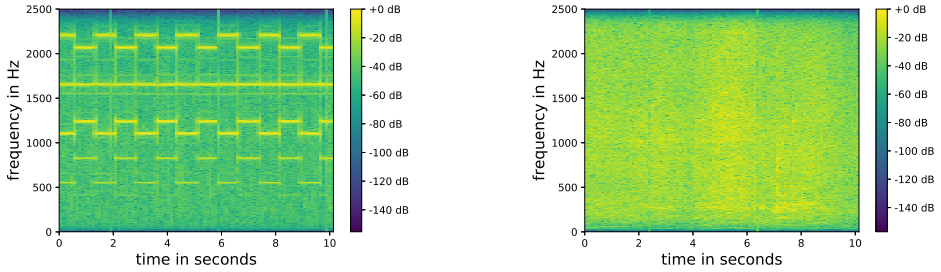
$$S[k,m] = \sum_{n=0}^{N_{fft}-1} x[n] w[m-n] e^{-j\frac{2\pi nk}{N_{fft}}}. \tag{107}$$

The feature LSTFT, which is used in this approach, is the logarithmic representation of the STFT:

$$S_{dB}[k,m] = 20 \cdot \log_{10}\left(|S[k,m]|\right). \tag{108}$$

Here, the frame length is $N_{fft} = 1024$ and the hop-size is $R = 512$. Fig. 4.4 shows the calculated LSTFT of exemplary time frames of the training data. The spectrogram of class M demonstrates its characteristic sound sequence (see section 3.2) and the spectrogram of class U shows the noisy environment.

Chromagram The energy distribution of a signal in relation to the 12 pitches of a musical octave can be described with a chromagram (C) [38]. The chromagram filter bank



(a) LSTFT of class M

(b) LSTFT of class U

Figure 4.4 – Exemplary time frames showing the logarithmic short-time fourier transform (LSTFT) of each class. The frame length is $N_{fft} = 1024$ and the hop-size is $R = 512$. The window shape is a *Hann window*. Spectrogram of class M demonstrates its characteristic sound sequence with its fundamental frequencies in the area of 500Hz. Class U has a noisy frequency distribution, which in this case includes inter alia street sounds, speech sounds and sounds of a restaurant.

$W[k, \tilde{k}]$ with $k \in \{1, \ldots, 12\}$ pitch classes (see Fig. 4.5) weights the power spectrum $S[\tilde{k}]$ with frequency bins $\tilde{k}$ resulting in the chromagram

$$C[k] = \sum_{\tilde{k}=1}^{N} S[\tilde{k}] \cdot W[k, \tilde{k}]. \tag{109}$$

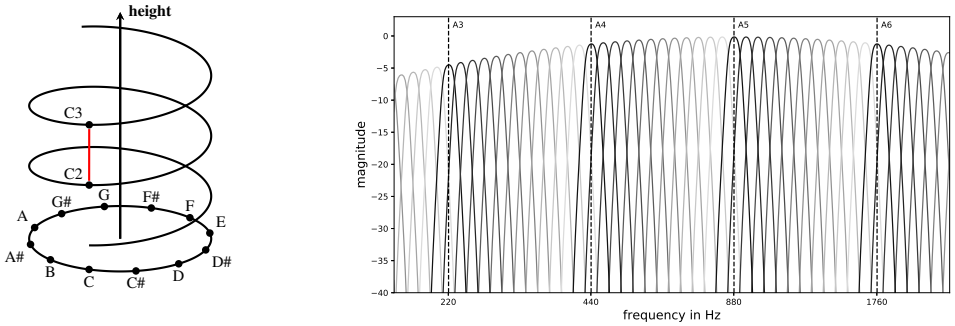
Fig. 4.6 illustrates the chromagrams of both classes. In comparison with the LSTFT (see Fig. 4.4), the harmonics of class M are combined to pitch classes. Dominant estimated pitches in this exemplary audio extraction in the beginning are the pitches *D*, *E*, and *A*. Later on the pitches *C#* and *G#* are more prominent (transition to next audio file of the training set with different pitches due to the *Doppler effect*). Pitch classes of class U are distributed similar over all pitches.

Chromatic Entropy Each value of each pitch class in a snippet of the chromagram can be seen as a probability. Here, for each snippet the chromagram is normalized leading to the interpreted *probabilities*

$$p[k, m] = \frac{\mathrm{C}[k, m]}{\sum_{k=1}^{12} \mathrm{C}[k, m]}. \tag{110}$$

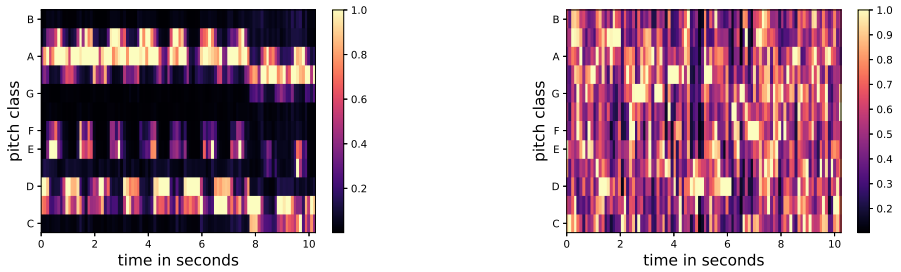
Thus, the sum over all *probabilities* in each frame m is $\sum_{k=1}^{12} p[k, m] = 1$. The chromatic entropy (CE) can then be calculated with the relation

$$\mathrm{CE}[m] = -\sum_{k=1}^{12} p[k, m] \ln\left(p[k, m]\right). \tag{111}$$



(a) Pitch helix (b) Chromagram filter bank

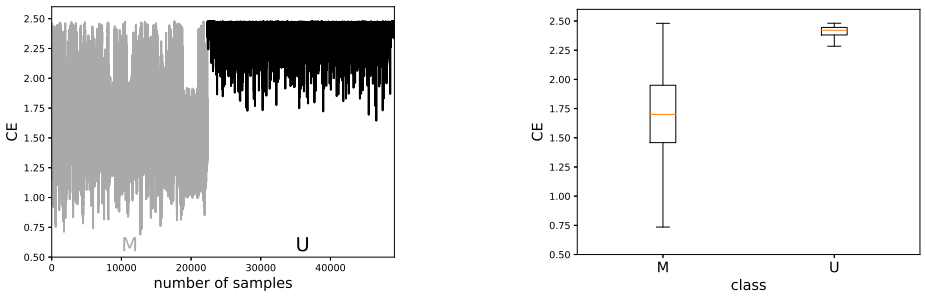
Figure 4.5 – Pitch helix illustrating the connection between octave intervals (above). An octave interval is defined by a spacing of 12 semitones (e.g. pitch C2 to C3). The link between those octaves is used for the calculations of the pitch classes of the chromagram. The power spectrum of each snippet is filtered with a scaled filter bank (below), where all related pitches (e.g. $A0, A1, A2, \dots$) in each frame are weighted by gaussian bumps and then are summed up to a pitch value for the specific pitch class. Dashed vertical lines show the locations of pitches A3, A4, A5 and A6 with its corresponding frequencies. The related bumps for each pitch are illustrated in different shades of grey. Hence, the filtering converts the power spectrum to a chromagram with its 12 pitch classes.



(a) Chromagram of class M (b) Chromagram of class U

Figure 4.6 – Exemplary time frames showing the chromagram of class M (left) and class U (right). In the chromagram of class M the dominant pitch classes of the fundamental frequencies can be seen. For class U usually each pitch class has a similar distribution.

It is assumed, that class U has a higher chromatic entropy than class M. In Fig. 4.7 the chromatic entropy of both classes are illustrated. Boxplots show, that the data of each class is distributed slightly different. For class M the median value is $\mathrm{CE}_{\mathrm{med}} \approx 1.7$, for class U the median is a bit higher with $\mathrm{CE}_{\mathrm{med}} \approx 2.42$. Hence, for class M between 4 and 5 dominant pitch classes with each having a probability of $p \approx 0.2$ are present in the training data set. This can be explained by the fact, that the dominant pitch classes are also surrounded by pronounced semitones (see Fig. 4.6a). In case of class U it can be expected, that each of the 12 pitch classes has a probability of $p = \frac{1}{12} \approx 0.083$.



(a) CE (b) Boxplot of CE

Figure 4.7 – Resulting chromatic entropy (CE) and respective boxplots of the training set with classes U and M.

Mel-Scaled Spectrogram The mel-scaled spectrogram (MELS) filters the power spectrum $S[\tilde{k}]$ with $\tilde{k}$ frequency bins based on the *mel scale*.

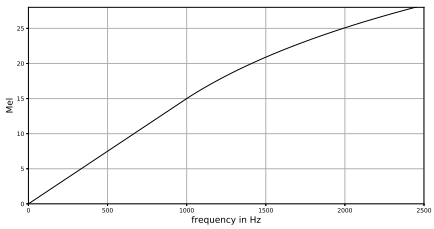
To understand the unit mel an introduction to the *frequency-to-place* transformation in the *cochlea* of the *inner ear* of a human being is given [39]. The cochlea includes the *basilar membrane*, where mechanical oscillations lead to impulses, which are then transmitted in the auditory nerve. The *unfurled* cochlea has a length of approximately $32\mathrm{mm}$. The vibrating air molecules enter the basilar membrane through the *oval window* and the *organ of Corti* (located above the basilar membrane) receives the vibrations with *hair cells*, which are then transmitted to the auditory nerve. At the end of the *basilar membrane* the so called *heliocotrema* is located. Higher frequencies are stimulated near the oval window and lower frequencies are stimulated near the *heliocotrema*. The transformations of those frequencies in the *basilar membrane* can be understood as a filter bank.

Hence, this psychoacoustic process leads to a frequency dependent spectral resolution in the hearing system of a human being, which results in the mel scale. In Fig. 4.8 the mel scale mapping and the resulting filter bank with 128 mel bins is illustrated. The feature was calculated with the python package librosa [40], which uses the mel scale from the *Auditory Toolbox* of Malcolm Slaney [41]:

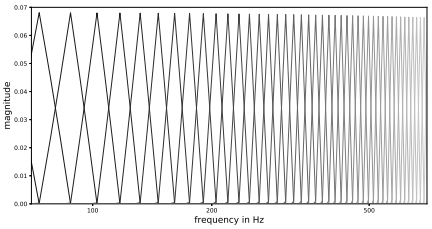
$$\text{Mel} = \begin{cases} \frac{f}{f_{sp}} & \text{if } f < f_c \\ \frac{f}{f_{sp}} + \log\left(\frac{f}{f_c}\right)\frac{27}{\log(6.4)} & \text{otherwise.} \end{cases} \tag{112}$$

The spacing frequency $f_{sp} \approx 66\mathrm{Hz}$ and the frequencies are mapped linearly to the mel scale below the frequency of $f_c = 1000\mathrm{Hz}$. The corresponding mel-scaled spectrogram with $k \in \{1, \ldots, 128\}$ bins is calculated by weighting the power spectrum $S[\tilde{k}]$ with the filter bank $W[k, \tilde{k}]$

$$X_{Mel}[k] = \sum_{\tilde{k}=1}^{N} S[\tilde{k}] \cdot W[k, \tilde{k}]. \tag{113}$$

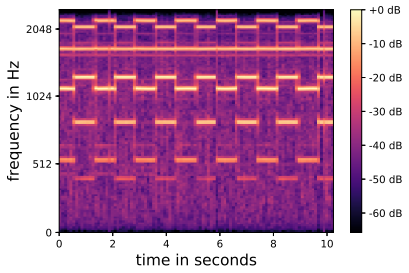


(a) Mapping frequencies from 0Hz to 2500Hz to the mel-scale in accordance with the *Auditory Toolbox* of Malcolm Slaney [41].

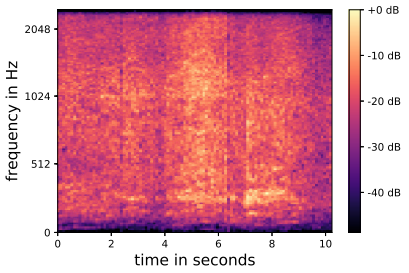


(b) Mel filter bank in the area of the fundamental frequencies of a *Martinshorn* used for calculating the mel-scaled spectrogram. 128 triangular filters shape the power spectrum $S[\tilde{k}]$ resulting in a mel-scaled spectrogram $S_{Mel}[k, m]$ with 128 mel bins.

Figure 4.8 – Frequency to mel-scale mapping (a) and used mel filters in the frequency area from 70Hz to 700Hz in case of 128 mel bins. In this area the frequency values are mapped linearly to the mel bins. It can be seen, that the magnitudes of the triangular filters decrease with higher frequencies to preserve the same amount of energy within a filter.



(a) MELS of class M

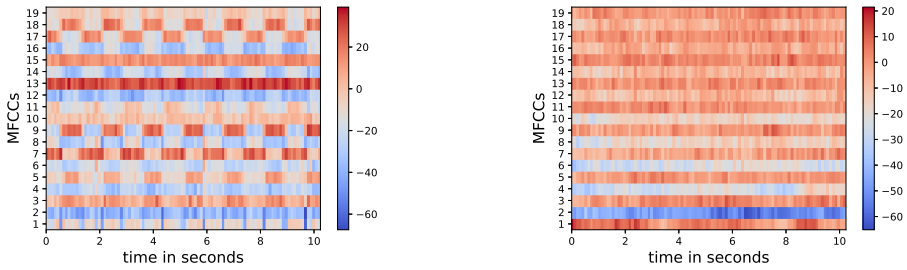


(b) MELS of class U

Figure 4.9 – Exemplary time frames showing the mel-scaled spectrogram (MELS) of each class. For a better visibility of the signal characteristics the spectrogram is presented logarithmic. As with the LSTFT the specific signal fundamental frequencies can be recognized in case of class M.

In Fig. 4.9 the mel-scaled spectrograms for both classes are illustrated (mel frequency bin labels k on the y-axis are converted to corresponding frequencies). The images resemble the illustrations of the LSTFT. Nevertheless, the harmonic frequency components of class M appear more blurred, but on the other hand less bins are necessary to transmit similar information. Even less mel bins could also have been chosen in this specific case. However, in this first approach the idea of keeping the frequency resolution high was paramount.

Mel Frequency Cepstral Coefficients The mel frequency cepstral coefficients (MFCC) describe the shape of the spectral envelope of a signal. The discrete cosine transform (DCT) of the logarithmic mel-scaled spectrogram $X_{Mel}[\tilde{k}, m]$ with $\tilde{k}$ mel frequency



(a) MFCCs of class M (b) MFCCs of class U

Figure 4.10 – Mel frequency cepstral coefficients (MFCCs) of two exemplary audio files. As with the chromagram in case of class M specific signal components stand out (e.g. cyclic ninth MFCC of class M).

bins results in the coefficients

$$\text{MFCC}[k,m] = \text{DCT}\left(20 \cdot \log_{10}(X_{Mel}[\tilde{k},m])\right) \tag{114}$$

with k mel frequency cepstral coefficients. In Fig 4.10 the first 19 MFCCs (without DC-component) of two exemplary audio files of classes M and U are illustrated. The repetitive structure of a *Martinshorn* can also be recognized in specific mel bins.

Fundamental Frequency Ratio This feature is optimized for the specific characteristic of the siren sound of a *Martinshorn*. In order to calculate the Fundamental Frequency Ratio (FFR) both defined fundamental frequencies of the *Martinshorn* should be present within the analysis frame. Therefore, the chosen audio snippet length is set to 4s. The fundamental frequencies of the sound event *Martinshorn* are located in the *bandpass-region* between 360Hz and 630Hz. Here, the signal is filtered with a *Butterworth Band-pass filter* within the region of 280Hz and 490Hz. The reason for choosing this frequency region is founded in the fact that in spite of the definition in the norm (see section 3.2.1) by looking at the data set the frequencies $\tilde{f}_1 = 330\text{Hz}$ and $\tilde{f}_2 = 440\text{Hz}$ most frequently occur. Consequently better results are found with this type of filtering. Next, the short-time fourier transform of the bandpass-filtered snippet is calculated. The frame length is $N_{fft} = 1024$ and the hop-size $R = 512$. Each frame was before windowed with a *Hann window*. Now the resulting spectrogram is smoothed with the following steps:

— Averaging each DFT-frame m over frequency with a moving average filter of the form $\frac{1}{7} \cdot \sum_{i=0}^{6} X_{bp}[k-i,m]$, where $X_{bp}[k,m]$ is the bandpass-filtered spectrum of each frame m with k frequency components
— Median-filtering of each frequency component over time (or rather frames) with order $N_{med} = 21$

In the end, the means of each frequency bin over time of the smoothed spectrogram is selected for the calculation of the fundamental frequency ratio (a calculation of the ratio in each frame did not improve the results) and $N_{buf} - 1 = 39$ copies of this calculated

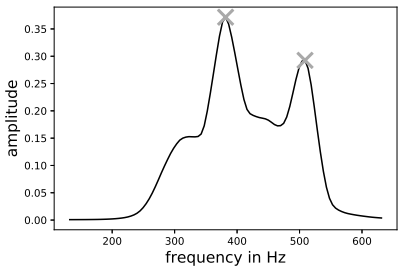
value are transferred to the feature space.

The evaluation criterion, which involves the ratio between the first two detected fundamental frequencies f_1 and f_2, is calculated in the following way [4]

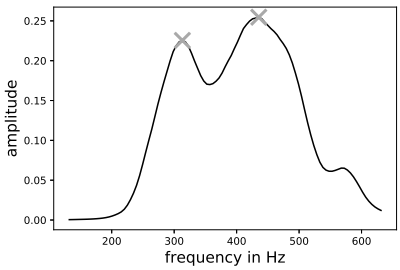
$$\text{FFR} = \left| \frac{f_2}{f_1} - 1.33 \right|^p = \left| \frac{f_2}{f_1} - 1.33 \right|^2 . \tag{115}$$

If the corresponding musical interval is close to the *perfect fourth*, the ratio would be $1.33 - 1.33 = 0$. In case of noisy data and a search of more distant located peaks (here at least 75Hz difference) it frequently occurs, that there are less than two peaks detected. In this case the ratio is calculated with $f_2 = 280\text{Hz}$.
Fig. 4.11 shows an exemplary detection process for training files of the two classes. In this case class M is near to the expected ratios of FFR ≈ 1.33 in both cases. However, in case of class U other ratios occur.
The resulting ratios of the fundamental frequencies of the training data set are illustrated in Fig. 4.12. Class M has a median of FFR ≈ 0.008 and class U has a median of FFR ≈ 0.39.

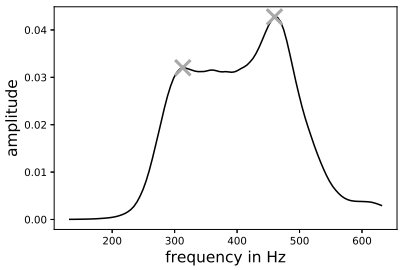
4. Also other attempts with different exponents $p > 2$ (stronger penalty) or a bias unit $\left(1 + \left|\frac{f_2}{f_1} - 1.33\right|\right)^p$ were made. However, those attempts did not improve the results sufficiently enough.



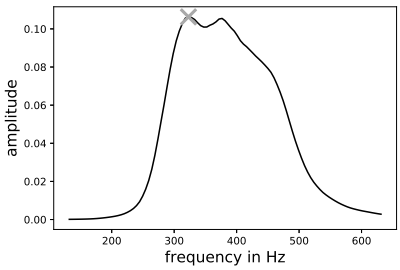
(a) Class M with FFR ≈ 0.00001



(b) Class M with FFR ≈ 0.00368

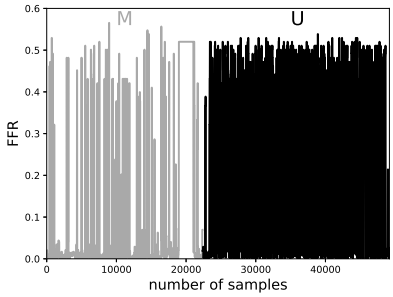


(c) Class U with FFR ≈ 0.01925

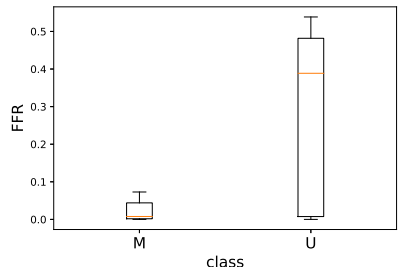


(d) Class U with FFR $= 0.21423$

Figure 4.11 – Detection of peaks ($\times$) of the fundamental frequencies f_1 and f_2 for the feature *fundamental frequency ratio* (FFR) of two exemplary files with class M (first row) and class U (second row). Exemplary results show that a FFR for class M is near to the desired ratio in both cases and a FFR for class U detects a different ratio (c) or only one peak, which leads to a more penalized ratio (d).



(a) FFR



(b) Boxplot of FFR

Figure 4.12 – Fundamental frequency ratio (FFR) of the training data set for each time frame described by the number of samples (left) and the corresponding boxplot (right).

4.1.3 Resulting Feature Space

The basic feature space for this approach consists of 675 features and 49179 observations (or rather $\frac{49179}{39} = 1261$ snippets with a length of 4s), which leads to a feature matrix $\boldsymbol{X}$ with size $[49179 \times 675]$ [5]. Here, each row of the feature matrix (observations) is characterized by a time frame of $N_{fft} = 1024$ samples and for each of those time frames a respective feature vector is delivered. The hop-size within each analysis window of 4s is $R = 512$.

The used *buffer-function* [42] divides the snippet as follows: The first 512 and also the last 480 values of each snippet with $N_{AW} = 20000$ samples are set to 0. This leads to $20000 + 480 = 20480$ samples which are buffered to $N_{buf} = 40$ frames (therefore the first frame involves 512 zeros followed by the first 512 samples of the signal and the last frame involves the last 544 samples of the signal followed by 480 zeros).

The calculated values of the first frame of each snippet are not transferred to the feature space. This leads to $N_{buf} - 1 = 39$ transmitted observations, where all samples of the snippet are involved, because also the second frame captures the samples $0 \dots 1024$ (see Table 10).

Frame	Time	Samples
1	0s . . . 0.1s	0 . . . 512
2	0s . . . 0.2s	0 . . . 1024
3	0.1s . . . 0.3s	512 . . . 1536
4	0.2s . . . 0.4s	1024 . . . 2048
⋮	⋮	⋮
37	3.5s . . . 3.7s	17920 . . . 18944
38	3.6s . . . 3.8s	18432 . . . 19456
39	3.7s . . . 3.9s	18944 . . . 19968
40	3.8s . . . 4s	19456 . . . 20480

Table 10 – Buffered snippet with 40 frames. The first column describes the frame number, the second column the corresponding time periods and the third column the associated samples. Both, the first and the second frame involve the starting samples of a snippet with a length of 4s.

The first 512 ($= \frac{N_{fft}}{2}$) features describe the amplitudes of the LSTFT with size $[39 \times 512]$, which in this case are mapped to the frequency range from 1Hz to 2500Hz. With the chosen sampling frequency the frequency resolution is $f_{res} = \frac{f_s}{N_{fft}} = \frac{5000\text{Hz}}{1024} \approx 4.88\text{Hz}$.

The chromagram with its 12 pitch classes and size $[39 \times 12]$ and the chromatic entropy with size $[39 \times 1]$ are the next attached features. Then, the mel-scaled spectrograms with size $[39 \times 128]$ and the resulting MFCCs with size $[39 \times 20]$ are added to the feature space.

5. In section 4.3.3 it is shown, that this kind of approach (time resolution of $\frac{N_{fft}}{f_s} = \frac{1024}{5000\text{Hz}} \approx 200\text{ms}$ per observation) also delivers good results in case of transitions between the two classes M and U.

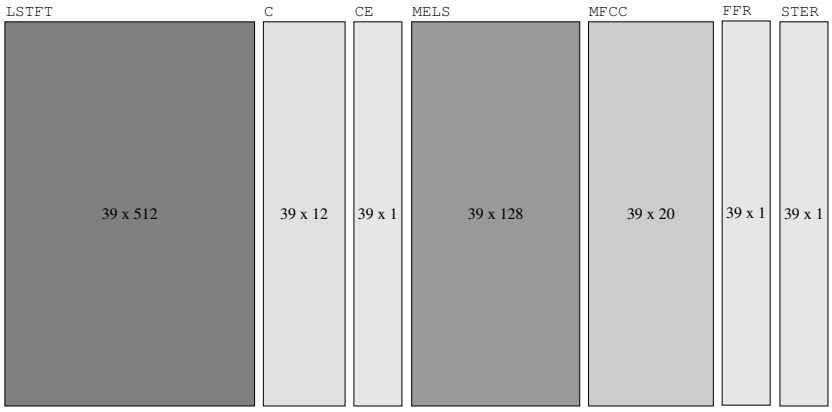


Figure 4.13 – Sketch of the feature matrix for each snippet with length of 4s illustrated by blocks with varying sizes. The logarithmic short-time fourier transform (LSTFT) with its 512 bins is the largest block. In comparison, the features chromatic entropy (CE), fundamental frequency ratio (FFR) and short-time energy ratio (STER) are defined by only one value per observation.

The two last features describe the FFR with size $[39 \times 1]$ and the STER with size $[39 \times 1]$. In case of the FFR feature there is only one value calculated for a snippet of 4s. Hence, there are $N_{buf} - 1$ copies of this value transferred to the feature space. The different feature blocks and its size in case of one 4s-snippet are depicted in Fig. 4.13.

4.1.4 Feature Selection

The applied feature selection algorithm computes the χ^2-statistics [43, 44]

$$\chi^2 = \sum_{i}^{N_C} \frac{(O_i - E_i)^2}{E_i} \tag{116}$$

of each feature, where χ^2 is Pearson's cumulative test statistic, O_i the number of observations of class i, E_i the expected number of class i and N_C the number of classes. The number of observations O_i is the sum over each feature per class i. The expected number E_i is the sum over each feature for all classes multiplied with the empirical probability $p(y_i)$ (percentage share of class i in the available data set).

Before applying the χ^2 feature selection the features were scaled into the range $[0, 1]$, because the statistic is designed for non-negative values. The scaling can be denoted as

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{117}$$

With the scaled features and with Eq. 116 the dependencies of the stochastic variables within each feature can be calculated.

Table 11 shows the χ^2-scores of the first 20 features which are calculated with the χ^2-selection algorithm. The worst rated features were especially specific frequency bins of the LSTFT greater than $\approx 1200\text{Hz}$. In the case of binary classification the features STER, FFR and almost all chromagram-features have the best scores. With regard to this feature selection and a simpler calculation in case of a future possible real-time implementation the models were trained with the features STER (1), FFR (1), C (12) and CE (1), which leads to 15 features in sum. It is assumed, that the generalization with new unknown test data be better afterwards, if all pitches of the chromagram are taken into account (also because of the *Doppler effect*). The feature MFCC_2 is not taken into account, because it did not improve the results (especially with regard to the self-recorded validation set) and simultaneously the calculation steps to obtain the MFCCs do not have to be made. Even if the feature CE has a lower score, its consideration in combination with the other features did improve the results [6].

Count	Symbol	Number	χ^2
1	STER	674	7463.35
2	FFR	675	7055.25
3	C_G	520	6480.37
4	C_C	513	6130.74
5	C_G#	521	4484.82
6	C_E	517	4039.09
7	C_C#	514	3714.61
8	C_D	515	3485.31
9	C_H	524	3397.43
10	C_F#	519	3134.95
11	C_D#	516	2965.08
12	C_F	518	2867.61
13	MFCC_2	655	1804.87
14	CE	525	1141.14
15	MFCC_4	657	1024.69
16	MELS_508Hz	551	975.51
17	MELS_684Hz	560	952.93
18	LSTFT_103Hz	21	889.22
19	MELS_2129Hz	634	799.93
20	MELS_527Hz	552	708.34
⋮	⋮	⋮	⋮

Table 11 – Feature Ranking with the 20 highest scores of χ^2 statistics.

6. Note, that all used feature selection algorithms served as an indication for a good feature set. Nonetheless, many other feature combinations were tried and tested *by hand*. In practice and especially with respect to the self-recorded validation set it turned out, that also other *at first sight irrelevant* features could make a good contribution in combination with *at first sight relevant* features.

4.1.5 Classification Results

In this first approach the two classes M and U are classified with the described feature selection. All applied classification algorithms are described in section 2. Information about the data set and the calculated features for the two classes are described in section 3 and section 4.1.2.

Finding a model which performs best in Recall in case of the self-recorded validation set is the main goal of all classification task. Each classifier was also tested with a k-fold cross validation to measure the ability of generalization: The training set is split into k stratified folds (test sets), where the percentage of samples for each class is preserved [45].

In a first attempt also the feature vector dimension was reduced with a *linear discriminance analyse* (LDA). The LDA tries to maximize the distance between the means (within-class scatter matrix) and minimize the spread within a class (between-class scatter matrix) [46]. However, this dimensionality reduction did not lead to a better performance with the self-recorded validation set. This lead to a rejection of this idea, because regardless of the computational costs a search for the best solution was paramount [7].

Also the mean values of the resulting 15 features for the 39 frames could have been chosen as one feature vector for a 4s-snippet. This attempt resulted in a slightly worse performance in the first place and was therefore not further investigated.

The data is standardized before the machine learning models are trained and tested. Hence, the mean value μ and the standard deviation σ of each feature is calculated from all observations of the training data set (here in each case 15 mean values and standard deviations). Then each feature is transformed to zero mean and unit variance:

$$X_{standard} = \frac{X - \mu}{\sigma}. \tag{118}$$

This standardization lead to better results with all machine learning models. Features with high variance could have a strong impact on the objective function, which could result in an inability to learn from other features [47].

The results of each developed model (Perceptron, Logistic Regression, Naive Bayes Classifier, K-Nearest-Neighbor and Support Vector Machine with a radial kernel) are described in the following paragraphs.

7. Nevertheless, at a later step (see section 4.3.1) a feature selection algorithm, which involves the principles of the LDA, is applied.

Perceptron Results for the classifier Perceptron (PPN) are listed in Tables 12, 13 and Fig. 4.14. In case of the self-recorded validation set this classifier is not able to classify M correctly.

Data	Accuracy
Test data from training set	0.977
Cross validation with training set	0.98 ± 0.02
Basic validation set	0.981
Self-recorded validation set	0.715

Table 12 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier PPN.

Class	Precision	Recall	F1 score	Quantity
U	0.968	0.99	0.979	8050
M	0.988	0.961	0.974	6704

(a) Test data from training set (30%)

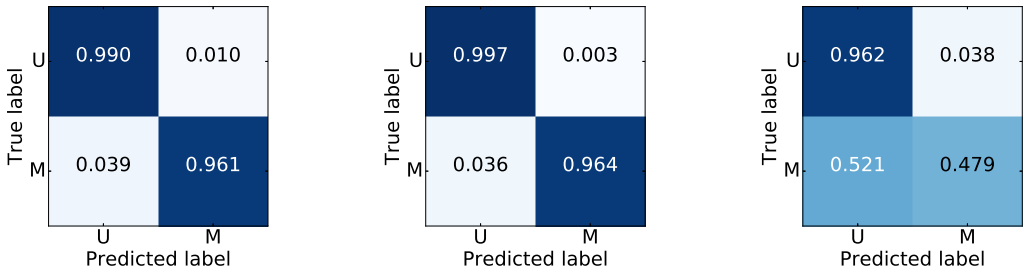
Class	Precision	Recall	F1 score	Quantity
U	0.965	0.997	0.981	9126
M	0.997	0.964	0.98	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.637	0.962	0.767	13455
M	0.93	0.479	0.633	14157

(c) Self-recorded validation data

Table 13 – Precision, Recall and F1 score of the classifier PPN for the different test data sets with respective amounts of feature vectors.



(a) Test data from training data (b) Basic validation data (c) Self-recorded validation data

Figure 4.14 – Confusion matrices of the different test data sets with the classifier PPN.

Logistic Regression Results for the classifier Logistic Regression (LR) are listed in Tables 14, 15 and Fig. 4.15. For the best over the two classes averaged Recall a parameter of $C = 0.3$ was chosen. In case of the self-recorded validation set this classifier is slightly better in classifying M correctly.

Data	Accuracy
Test data from training set	0.983
Cross validation with training set	0.98 ± 0.02
Basic validation set	0.983
Self-recorded validation set	0.822

Table 14 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier LR.

Class	Precision	Recall	F1 score	Quantity
U	0.979	0.991	0.985	8050
M	0.989	0.975	0.982	6704

(a) Test data from training set (30%)

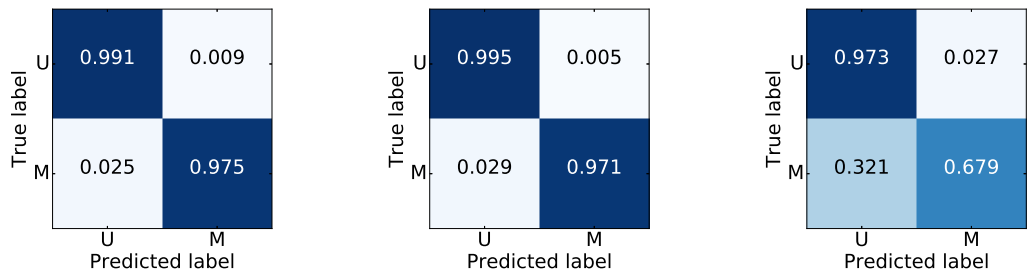
Class	Precision	Recall	F1 score	Quantity
U	0.972	0.995	0.983	9126
M	0.995	0.971	0.983	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.742	0.973	0.842	13455
M	0.963	0.679	0.796	14157

(c) Self-recorded validation data

Table 15 – Precision, Recall and F1 score of the classifier PPN for the different test data sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.15 – Confusion matrices of the different test data sets with the classifier LR.

Naive Bayes Classifier Results for the Naive Bayes (NB) classifier are listed in Tables 16, 17 and Fig. 4.16. This classifier leads to a good performance with the self-recorded validation set.

Data	Accuracy
Test data from training set	0.974
Cross validation with training set	0.97 ± 0.01
Basic validation set	0.973
Self-recorded validation set	0.848

Table 16 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier NB.

Class	Precision	Recall	F1 score	Quantity
U	0.969	0.984	0.976	8050
M	0.98	0.962	0.971	6704

(a) Test data from training set (30%)

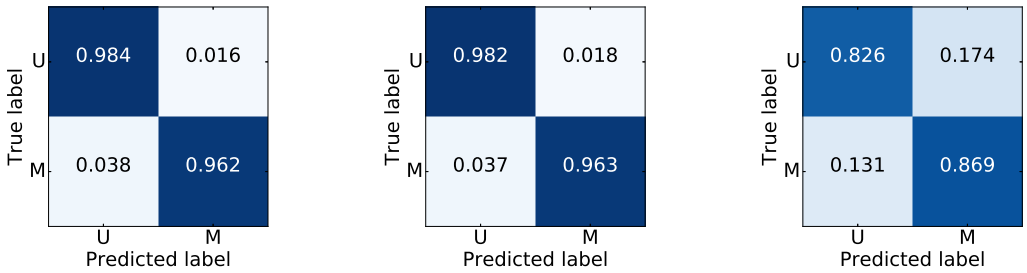
Class	Precision	Recall	F1 score	Quantity
U	0.964	0.982	0.973	9126
M	0.982	0.963	0.972	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.857	0.826	0.841	13455
M	0.84	0.869	0.854	14157

(c) Self-recorded validation data

Table 17 – Precision, Recall and F1 score of the classifier NB for the different test data sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.16 – Confusion matrices of the different test data sets with the classifier NB.

K-Nearest-Neighbor Results for the K-Nearest-Neighbor (KNN) classifier with $k = \sqrt{N} \approx 185$ are listed in Tables 18, 19 and Fig. 4.17. In case of the self-recorded validation set this classifier is comparable the classifier NB.

Data	Accuracy
Test data from training set	0.97
Cross validation with training set	0.96 ± 0.02
Basic validation set	0.965
Self-recorded validation set	0.854

Table 18 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier KNN.

Class	Precision	Recall	F1 score	Quantity
U	0.954	0.994	0.973	8050
M	0.992	0.942	0.966	6704

(a) Test data from training set (30%)

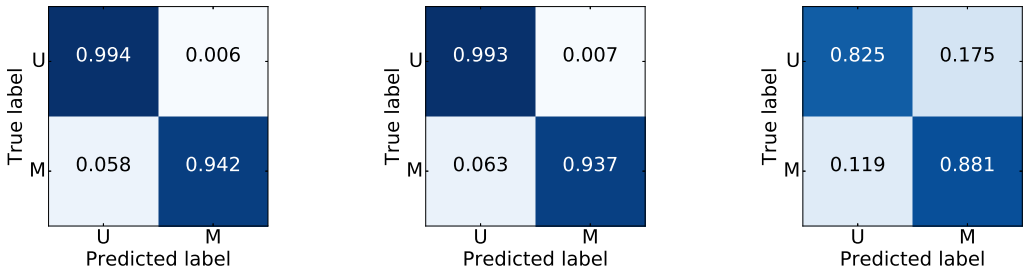
Class	Precision	Recall	F1 score	Quantity
U	0.94	0.993	0.966	9126
M	0.992	0.937	0.964	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.868	0.825	0.846	13455
M	0.841	0.881	0.861	14157

(c) Self-recorded validation data

Table 19 – Precision, Recall and F1 score of the classifier KNN for the different test data sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.17 – Confusion matrices of the different test data sets with the classifier KNN.

Support Vector Machine For the Support Vector Machine a radial kernel (SVMR) resulting in a non-linear decision boundary was chosen. The parameterization of C and γ is based on a grid search, which is depicted in Fig. 4.19. Results for the SVMR are listed in Tables 20, 21 and Fig. 4.18. On the whole, the SVMR lead to the best results.

Data	Accuracy
Test data from training set	0.993
Cross validation with training set	0.98 ± 0.03
Basic validation set	0.989
Self-recorded validation set	0.911

Table 20 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier SVMR.

Class	Precision	Recall	F1 score	Quantity
U	0.991	0.997	0.994	8050
M	0.997	0.989	0.993	6704

(a) Test data from training set (30%)

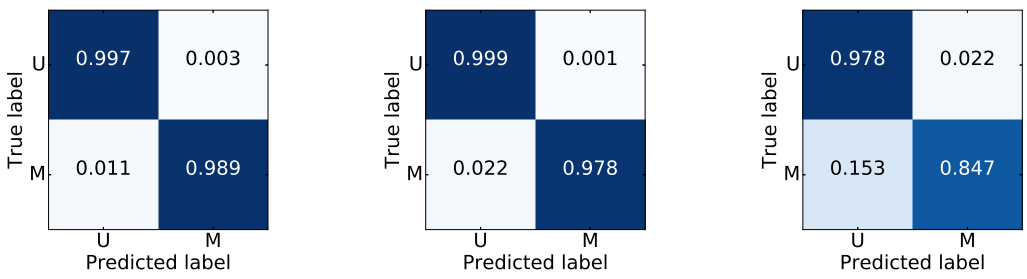
Class	Precision	Recall	F1 score	Quantity
U	0.979	0.999	0.989	9126
M	0.999	0.978	0.988	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.859	0.978	0.915	13455
M	0.976	0.847	0.907	14157

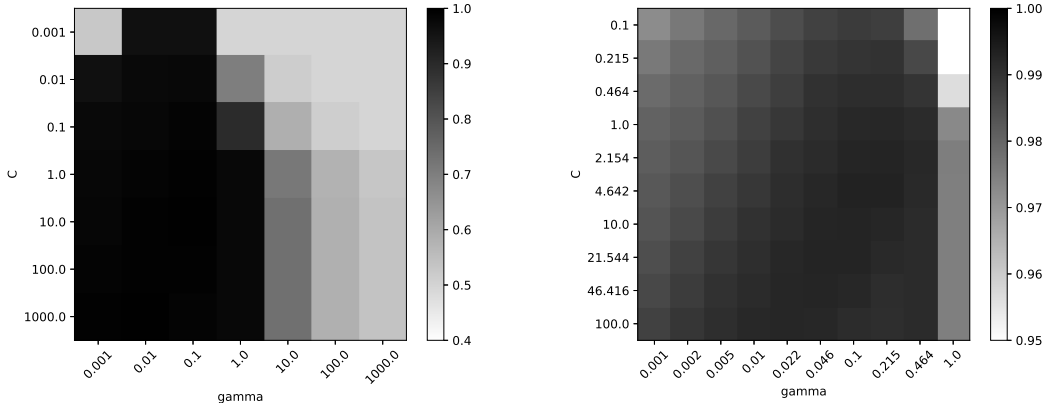
(c) Self-recorded validation data

Table 21 – Precision, Recall and F1 score of the classifier SVMR for the different test data sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.18 – Confusion matrices of the different test data sets with the classifier SVMR.



(a) Grid search for best Recall (rough)

(b) Grid search for best Recall (fine)

Figure 4.19 – Grid search for the best over the two classes averaged Recall with the classifier SVMR. Best parameters are located in the range $\gamma = [0.001, 1]$ and $C = [0.1, 1000]$ (left). The right figure shows a finer grid with $\gamma = [0.001, 1]$ and $C = [0.1, 100]$. The best parameters are chosen with $\gamma = 0.1$ and $C = 3$.

Summary All models perform well on the test data from the training set and on the basic validation data set. The goal of this classification task is an optimized performance with the self-recorded validation data set. Significant classification results for the self-recorded validation set are summarized in Table 22: The Accuracy of the classifier PPN with 71.5% is the worst. All other classifier have a Accuracy greater than 80%. Nevertheless, the performance measures Precision, Recall and F1 score for the class M play an important role when analyzing the results. Especially a higher Recall should be achieved to guarantee, that the model really reacts when class M is present. Best Recall of $\approx 87\%$ is given with NB, but on the other hand Precision and F1 score are lower than 87%. SVMR shows better results for Precision and F1 score and yet has a Recall of $\approx 84.7\%$.

Model	Accuracy	Precision (M)	Recall (M)	F1 score (M)
PPN	0.715	0.93	0.479	0.633
LR	0.822	0.963	0.679	0.796
NB	0.848	0.84	0.869	0.854
KNN	0.854	0.841	**0.881**	0.861
SVMR	0.911	0.976	0.847	0.907

Table 22 – Summary of specific performance measures of the self-recorded validation data set for the applied models in the binary classification task.

In another attempt, the self-recorded validation set was also preprocessed with a noise suppression algorithm and than classified again for a comparison. This attempt did not improve the results.

4.2 Classification of the classes *Martinshorn*, *Environmental Siren* and *Urban Noise*

Results of the first approach show, that the two classifiers PPN and LR did not perform sufficiently well on the self-recorded validation data set. For this reason those classifiers are excluded in the next two machine learning approaches.
During initial trials on this classification task it appeared, that a distinction between the classes M and S was not good enough with the available feature set. For this reason more features are introduced for this classification task to provide a better distinctness between those two classes.

4.2.1 Preprocessing

This approach performs the same preprocessing steps as in the binary classification task (see Section 4.1.1 and Table 10). It is assumed, that the length of a sound snippet of 4s guarantees enough distinctive information when comparing the classes S and M. Also the sampling frequency $f_s = 5000\text{Hz}$ has been retained, because higher frequency areas may not necessarily contain additional information.

4.2.2 Conclusive Audio Features

The feature selection of the first approach showed, that specific features make no determining contribution on classification. This knowledge is used for the new feature space, which leads to the preselection: C, CE, STER and FFR.

However, first results with the existing feature set on the classification task with three classes showed, that other new features could lead to an even better performance. The focus of those features lies on the distinction between the three classes S, M and U. It is assumed, that classes S and M can get confused more easily because of their similarities.

Each new calculated feature of this classification task (see chosen symbols and descriptions in Table 23) is described in the following paragraphs: At first the features MPDR, ZCR and COV are explained, then all chromagram features (CP, CC, CS and CF), which in this case are all derived from the median-filtered chromagram (CM), are described. The calculations of the spectral features (CC, CS and CF) are derived from [48] and they were also applied in [49].

Symbol	Description
MPDR	maximum peak density ratio
ZCR	zero-crossing rate
COV	covariance vector
C	chromagram
CM	chromagram median-filtered
CE	chromatic entropy
CEM	chromatic entropy median-filtered
CP	chroma prominence
CC	chroma centroid
CS	chroma spread
CF	chroma flux
STER	short-time energy ratio
FFR	fundamental frequency ratio

Table 23 – Calculated audio features with corresponding symbol and their descriptions. The chromagram-features C, CM and CP involve the 12 pitch classes.

Maximum Peak Density Ratio The maximum peak density ratio (MPDR) leads to a better distinction between the sounds of sirens (M/S) with high peaks in the frequency domain and the class U. The MPDR is calculated in the logarithmic domain and can be formulated as

$$\begin{aligned} \text{MPDR}_{dB} &= S_{\text{dB,max}}[k] - S_{\text{dB,min}}\left[\boldsymbol{k}_{\text{range}}\right], \\ \text{with} \quad \boldsymbol{k}_{\text{range}} &\in \left[\boldsymbol{p}_{\text{loc}}\left[P/2\right] - a, \boldsymbol{p}_{\text{loc}}\left[P/2\right] + a\right], \end{aligned} \tag{119}$$

where the vector $\boldsymbol{p}_{\text{loc}}$ gives the indices of all detected peak locations in the given logarithmic spectrum $S_{\text{dB}}[k]$ with frequency bins k. The length of the vector is given by the maximum number of detected peaks P. The minimum is searched in the vicinity of the peak location $\boldsymbol{p}_{\text{loc}}[P/2]$. The parameter a should be chosen with respect to the fft-size N_{fft}. It should lead to a suitable frequency range for which the minimum value is searched (here $a = 10$).
Hence, instead of choosing the minimum value of the entire spectrum, a minimum value of a segment with a high peak density of the spectrum is chosen. This ratio lead to good results when comparing the distributions of each class.
Fig. 4.20 shows how the MPDR is calculated with an exemplary audio frame of class M and Fig. 4.21 shows how the MPDR is calculated for an exemplary audio frame of class U.
Fig. 4.22 shows the course of the MPDR of the training set and the corresponding boxplots for each class. It can be seen, that the scattering and the median value of class U is lower in comparison to the siren sounds.

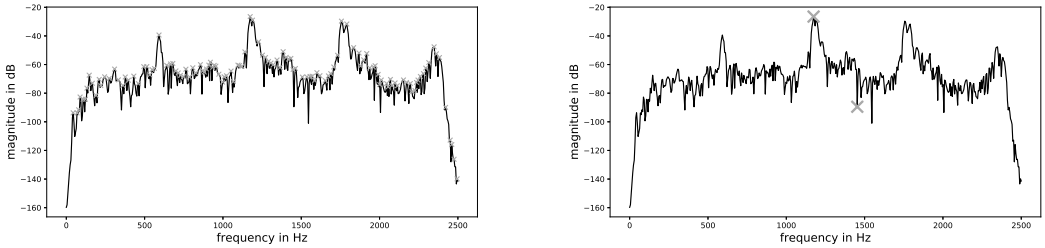


Figure 4.20 – Calculation of the maximum peak density ratio (MPDR) of an exemplary audio file of Class M: Left figure shows the logartihmic short-time fourier transform with 123 detected peaks and right figure shows the two detected peaks for which the MPDR is calculated. Here, the minimum is searched within the region of 1066Hz and 1462Hz. The maximum is the highest detected peak resulting in a maximum peak density ratio of MPDR $\approx 63dB$.

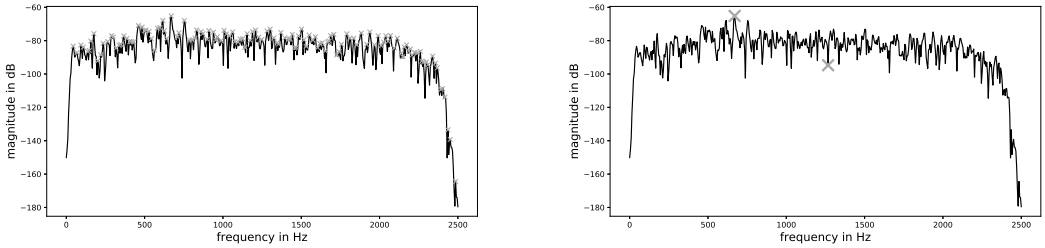


Figure 4.21 – Calculation of the maximum peak density ratio (MPDR) of an exemplary audio file of Class U: Left figure shows the logartihmic short-time fourier transform with 134 detected peaks and right figure shows the two detected peaks for which the MPDR is calculated. Here, the minimum is searched within the region of 1125Hz and 1482Hz. The maximum is the highest detected peak resulting in a maximum peak density ratio of MPDR $\approx 29dB$.

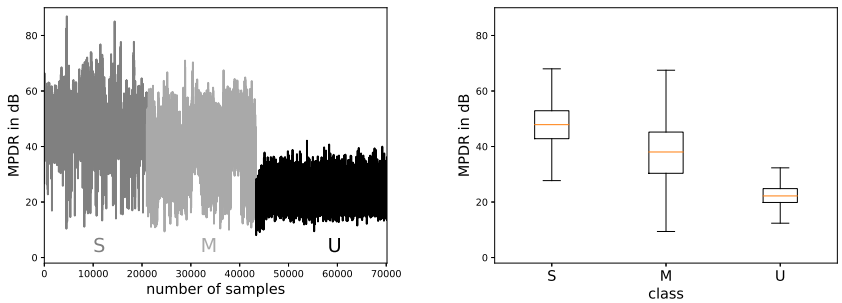
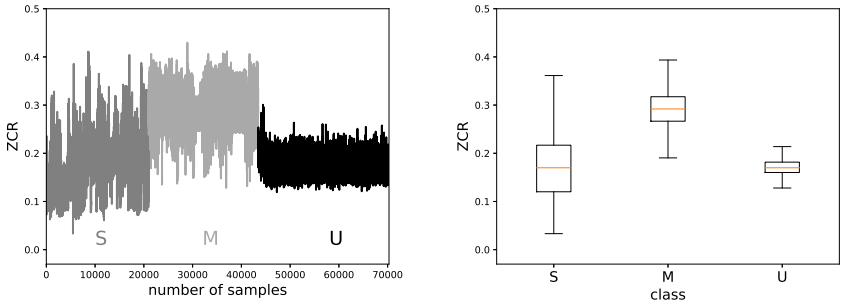


Figure 4.22 – Resulting maximum peak density ratio (MPDR) and respective boxplots of the training set with classes S, M and U.

Zero-crossing Rate Definitions of the zero-crossing rate (ZCR) can be found in [48, 50]. The feature has been calculated with the python toolbox librosa [40]. The Zero-crossing rate describes the rate of sign-changes along time and can be formalized as



(a) ZCR (b) Boxplot of ZCR

Figure 4.23 – Resulting zero-crossing rate (ZCR) and respective boxplots of the training set with classes S, M and U.

$$\text{ZCR} = \frac{1}{N-1}\sum_{n=1}^{N-1} |\text{sign}(x[n]) - \text{sign}(x[n-1])|, \tag{120}$$

with N as the number of samples and the signum function

$$\text{sign}(x[n]) = \begin{cases} 1, & \text{if } x[n] > 0 \\ 0, & \text{if } x[n] = 0 \\ -1, & \text{if } x[n] < 0 \end{cases} \tag{121}$$

There is some sort of relation between the ZCR and the fundamental frequency. Hence, higher values of the ZCR indicate higher estimations of the fundamental frequency ($f = \frac{\text{ZCR} \cdot f_s}{2}$) or indicate more or less stochastic, noisy signal components.

Fig. 4.23 illustrates the ZCR of the training set. Boxplots show, that the ZCR of class U has the lowest scattering, which could lead to better classification results. Apparently the estimated fundamental frequencies of class M (between approximately 0.2 and 0.4 or rather 500Hz and 1000Hz) are mostly higher when compared to class S.

It is assumed, that the estimated fundamental frequencies of class M are not accurate enough, because of the signal's noise and the A-weighting of the preprocessing step (more damped lower fundamental frequency). Nevertheless, the first harmonic of the lower fundamental frequency seems to be well captured and has also a relevant signal information. The higher scattering of class S on the other hand results from the fact, that the sweeps at the start and at the end are measured very well.

Covariance Vector The *Covariance Vector* (COV) combines variances and covariances of specific time frames. Each 4s-snippet is cut into 4 bandpass-filtered signals (here, the signal is filtered within the region of 280Hz and 670Hz), where each of them has a length of 1s (corresponding time ranges: $0\text{s} \ldots 1\text{s}$, $1\text{s} \ldots 2\text{s}$, $2\text{s} \ldots 3\text{s}$ and $3\text{s} \ldots 4\text{s}$). This leads to a vector of the form

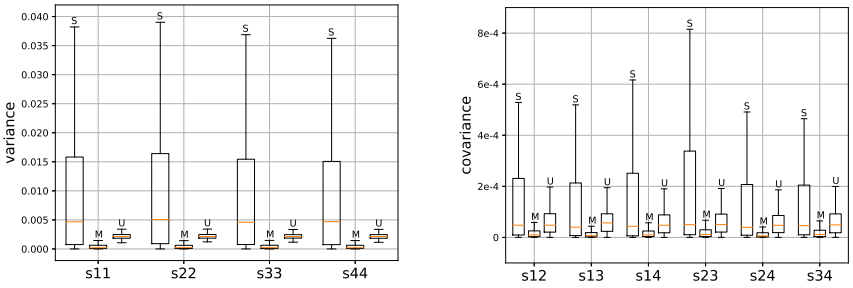
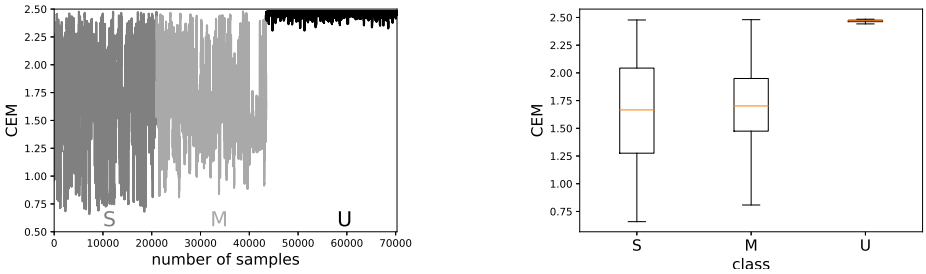


Figure 4.24 – *Covariance Vector* with variances of signals $\boldsymbol{s}_{11}$, $\boldsymbol{s}_{22}$, $\boldsymbol{s}_{33}$ and $\boldsymbol{s}_{44}$ (left) and covariances $s_{i,j} = |\mathrm{COV}(s_i, s_j)|$ related to the examined signal combinations (right).

$$\mathrm{COV} = \begin{bmatrix} \mathrm{Var}(\boldsymbol{s}_1) \\ \mathrm{Var}(\boldsymbol{s}_2) \\ \mathrm{Var}(\boldsymbol{s}_3) \\ \mathrm{Var}(\boldsymbol{s}_4) \\ |\mathrm{COV}(\boldsymbol{s}_1, \boldsymbol{s}_2)| \\ |\mathrm{COV}(\boldsymbol{s}_1, \boldsymbol{s}_3)| \\ |\mathrm{COV}(\boldsymbol{s}_1, \boldsymbol{s}_4)| \\ |\mathrm{COV}(\boldsymbol{s}_2, \boldsymbol{s}_3)| \\ |\mathrm{COV}(\boldsymbol{s}_2, \boldsymbol{s}_4)| \\ |\mathrm{COV}(\boldsymbol{s}_3, \boldsymbol{s}_4)| \end{bmatrix}^T , \tag{122}$$

where $\mathrm{Var}(\boldsymbol{s}_i) = \mathbb{E}[(\boldsymbol{s}_i - \mu_i)^2]$ is the variance of the respective signal and $|\mathrm{Cov}(\boldsymbol{s}_i, \boldsymbol{s}_j)|$ (where $i \neq j$) the absolute value of the covariance of the two compared signals. For each snippet N copies of each entry of the vector are repeated, resulting in a matrix with size $[N \times 10]$, which can be added to the feature space.
The idea behind this feature is the relationship between two estimated time regions in which one of the two fundamental frequencies of the *Martinshorn* could lie. Even if the frequencies are changing within a time region of approximately 1s, the variances and covariances of those regions have characteristic differences in comparison to noise sounds. Fig. 4.24 shows, that in all cases class S has a much higher scattering than classes M and U.

Chromagram Median-filtered The median-filtered chromagram (CM$[k, m]$ with k pitch classes and $m \in [0, N_{buf} - 1]$ frames) is calculated with a spectrogram decomposition algorithm from the audio analysis toolbox librosa [51, 52]. The basis is formed by the normalized chromagram (see section 4.1.2). The idea behind the algorithm is emphasizing repeating elements within a snippet. A *recurrence matrix* with size $[(N_{buf} - 1) \times (N_{buf} - 1)]$ is derived from a sparse *k-nearest neighbor graph*, which shows the connections between neighboring points [17]. Here, for each time frame m the $k_{NN} = 2 \cdot \left\lceil \sqrt{(N_{buf} - 1) - 1} \right\rceil + 2 = 16$ nearest neighbors are calculated with a cosine distance



(a) CEM (b) Boxplot of CEM

Figure 4.25 – Resulting chromatic entropy median-filtered (CEM) and respective boxplots of the training set with classes S, M and U.

metric. The cosine distance can be derived from the cosine similarity. Hence, the distance between two time frames of the chromagram can be written as

$$\mathrm{d}(j_a, j_b) = 1 - \cos(\theta) = 1 - \frac{\sum_{k=1}^{12} \mathrm{C}[k, j_a]\mathrm{C}[k, j_b]}{\sqrt{\sum_{k=1}^{12} \mathrm{C}[k, j_a]^2}\sqrt{\sum_{k=1}^{12} \mathrm{C}[k, j_b]^2}}, \tag{123}$$

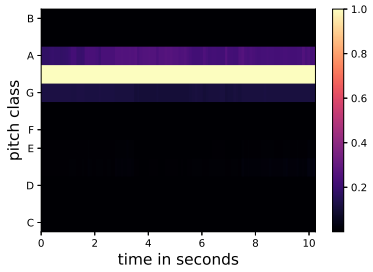
where $\forall j_a, j_b \in [1, m]$, which indicates a brute-force search. The identified repeating nearest-neighbor elements with indices J_j are then median-filtered, what leads to the *denoised* chromagram

$$\mathrm{CM}[k, m] = \underset{l \in [1, k_{NN}]}{median}\{C[k, J_j(l)]\}. \tag{124}$$

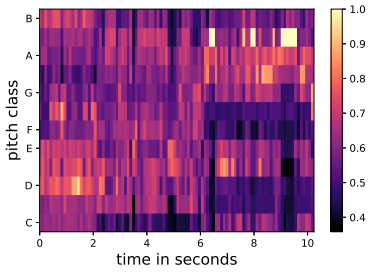
An exemplary time frame of a sirene sound from the training data set shows, that one pitch is strongly emphasized (see Fig. 4.26a). This should lead to a good separation for a model with three classes.

Chromatic Entropy Median-filtered The median-filtered chromatic entropy (CEM) is derived from the median-filtered chromagram. The calculation is analogous to the calculation of the chromatic entropy of the unfiltered chromagram (see section 4.1.2). Fig. 4.25 shows the resulting feature values of each sample of the training set and the respective boxplots. Median values of classes S and M are similar, only the median value of class U is a bit higher.

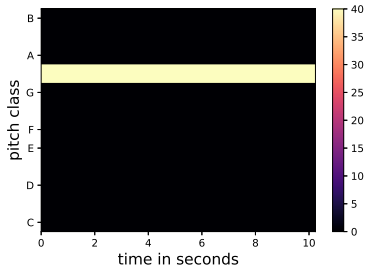
Chroma Prominence Ideally the chromagram of class S has one emphasized pitch (see Fig. 4.26a) and the chromagram of class M two emphasized pitches (see Fig. 4.6a). For this reason the idea to count the quantity of the highest peaks of each pitch class within a snippet came up. The feature with size $[12 \times (N_{buf} - 1)]$ is designated as the *Chroma Prominence*. In each frame m of the median-filtered chromagram the pitch k with the highest magnitude (here, the maximum value max $\{\mathrm{CM}[k, m]\}$ in each frame m is equal to 1) is detected and stored in a counter. This results in a vector with 12 entries,



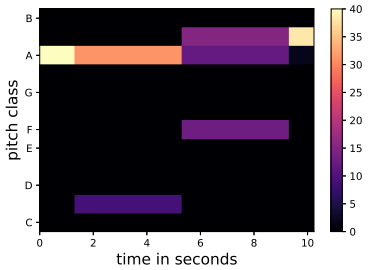
(a) Median-filtered chromagram of class S



(b) Median-filtered chromagram of class U



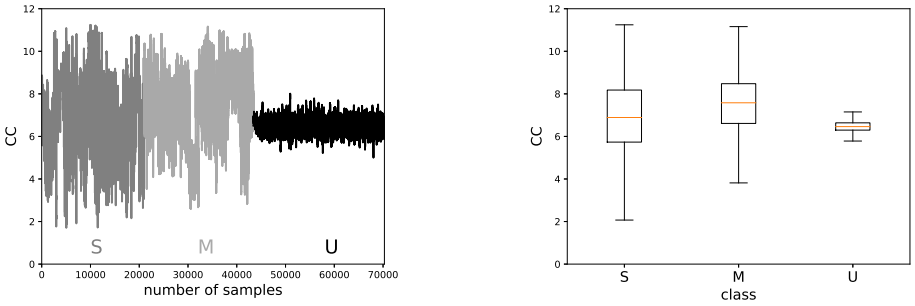
(c) Chroma prominence of class S



(d) Chroma prominence of class M

Figure 4.26 – Exemplary time frames of the training data set showing the median-filtered chromagram (first row) and the respective chroma prominence (second row) of class S (left) and class U (right). One dominant pitch can be recognized in case of class S. In case of class U the median-filtered chromagram is similar distributed. Second row illustrates the chroma prominence feature of class S (c) and class M (d). For class S only the pitch $G\#$ always has the highest amplitude, in case of class M mostly 3 (fundamental frequencies plus dominant harmonics) or 2 (only fundamental frequencies) pitches are emphasized.

each describing the quantity of maximum peaks of each pitch class within a snippet. Next, for each snippet N copies of each entry of the vector are repeated, resulting in the matrix with size $[(N_{buf} - 1) \times 12]$.
Fig. 4.26c shows, that in case of an exemplary time frame of class S only one pitch is emphasized. On the other hand, an exemplary time frame of class M has 2 or 3 dominant pitch classes within a snippet.



(a) CC (b) Boxplot of CC

Figure 4.27 – Resulting chroma centroid (CC) and respective boxplots of the training set with classes S, M and U.

Chroma Centroid The chroma centroid is an *experimental feature*, which describes the emphasis of the median-filtered chromagram. The chromagram is considered as a distribution, where the chroma bins describe the values and the amplitudes describe the probabilities. It can be written as

$$\mathrm{CC}[m] = \frac{\sum_{k=1}^{12} k \cdot \mathrm{CM}[k,m]}{\sum_{k=1}^{12} \mathrm{CM}[k,m]}. \tag{125}$$

Fig. 4.27 illustrates the distributions of the chroma centroid in the training set. It can been seen, that sirene sounds have a much higher scattering compared to urban noise sounds.

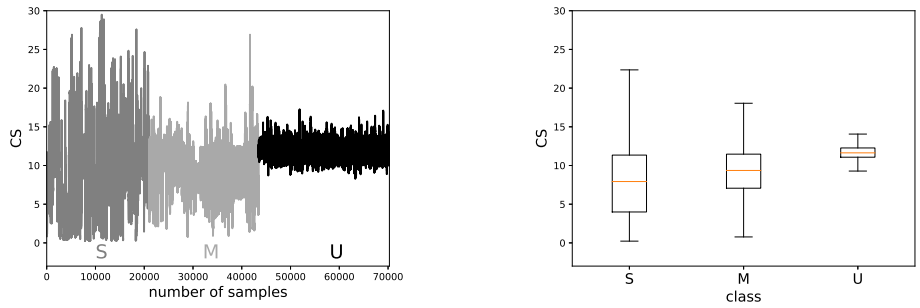
Chroma Spread The chroma spread gives a measure of the spread around the chroma centroid and is tested additionally to the related feature chromatic entropy. It can be considered as the shape of the median-filtered chromagram in each frame and is calculated as

$$\mathrm{CS}[m] = \frac{\sum_{k=1}^{12} (k - \mathrm{CC}[m])^2 \cdot \mathrm{CM}[k,m]}{\sum_{k=1}^{12} \mathrm{CM}[k,m]}. \tag{126}$$

Fig. 4.28 shows the distributions of the chroma spread of the training set. Once more, the scattering of class U is lower in comparison to classes S and M. On the other hand the median value of class U is a bit higher.

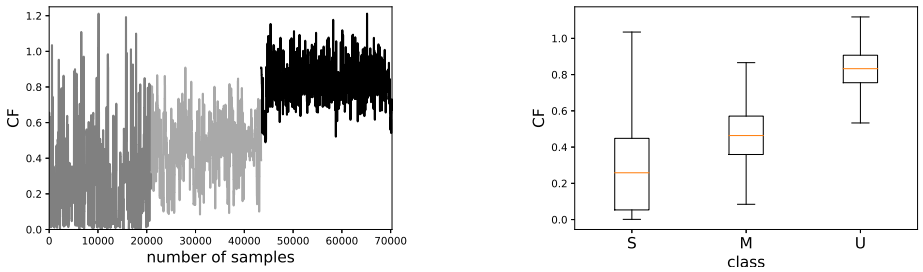
Chroma Flux The magnitudes of each pitch class vary over time. Those differences are measured and summed by the chroma flux

$$\mathrm{CF}[m] = \sum_{k=1}^{12} |\mathrm{CM}[k,m] - \mathrm{CM}[k,m-1]|, \tag{127}$$



(a) CS (b) Boxplot of CS

Figure 4.28 – Resulting chroma spread (CS) and respective boxplots of the training set with classes S, M and U.



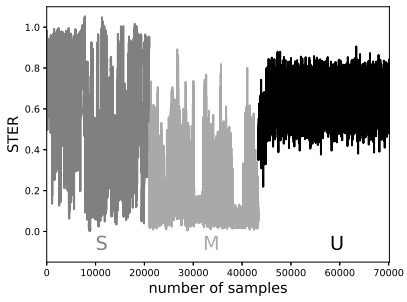
(a) CF (b) Boxplot of CF

Figure 4.29 – Resulting chroma flux (CF) and respective boxplots of the training set with classes S, M and U.

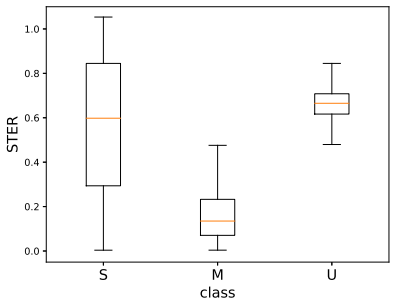
which is summing up all pitch class changes over time. Note, that for this feature the magnitude of the subsequent frame is necessary to calculate the actual value. This means, that only $N_{buf} - 1 = 39$ values can be calculated for each snippet, if it was buffered to $N_{buf} = 40$ frames. Here, for a better statistical representation $N_{buf} - 1 = 39$ copies of the mean value of all frames per snippet are illustrated. In case of class U, the scattering and also the median value of CF is higher when compared to classes S and M (see Fig. 4.29).

Short-Time Energy Ratio The calculation of the STER is analogous to the before described calculation (see 4.1.2). The distributions of the classes M and U stay the same (compare Fig. 4.2 and 4.30). In case of class S the median value amounts approximately 0.6 and the ratio values have a high scattering, which makes it difficult to distinguish between the class S and the classes M and U. The feature distribution shows, that the amplitudes in the training data of class S vary widely.

Fundamental Frequency Ratio The calculation of the FFR is analogous to the before described calculation (see 4.1.2). In Fig. 4.31 the resulting FFR feature of the training

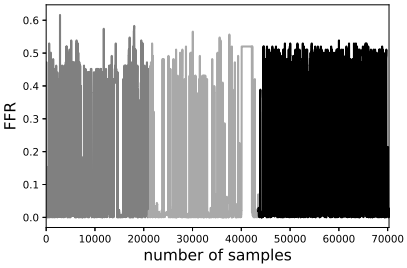


(a) STER.

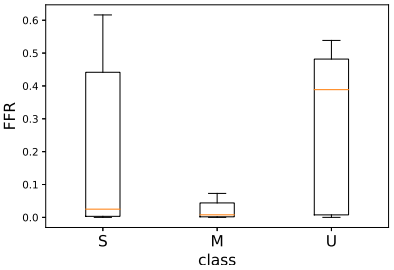


(b) Boxplot of STER.

Figure 4.30 – Short-time energy ratio (STER) over time (a) and boxplot of STER (b) of training data with the three classes S, M and U.



(a) FFR



(b) Boxplot of FFR

Figure 4.31 – Resulting fundamental frequency ratio (FFR) and respective boxplots of the training set with classes S, M and U.

set and the corresponding boxplots of each class are illustrated. Classes S and U are penalized very often in comparison to class M. Nevertheless, when looking at the boxplot of class U the median value is higher than the median values of classes S and M.

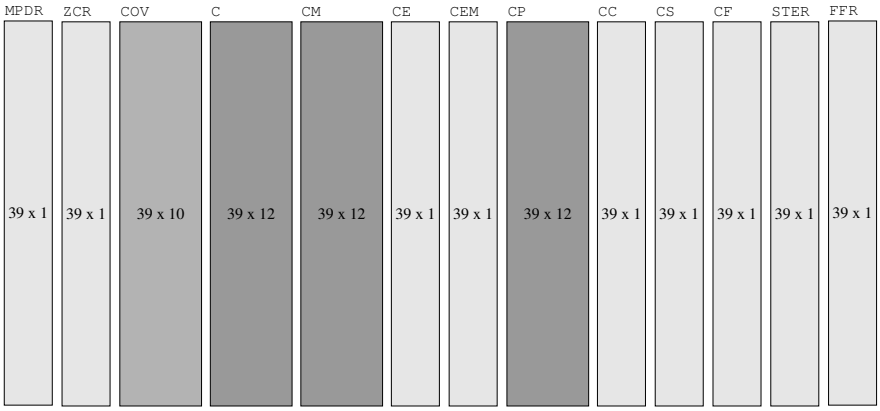


Figure 4.32 – Sketch of the feature set illustrated by blocks with varying sizes. The feature matrices chromagram (C), chromagram median-filtered (CM) and chroma prominence (CP) with its 12 pitch classes are the largest blocks, followed by the feature matrix covariance vector (COV) with its 10 measures of variation. In comparison, the features maximum peak density ratio (MPDR), zero-crossing rate (ZCR), chromatic entropy (CE), chromatic entropy median-filtered (CEM), chroma centroid (CC), chroma spread (CS), chroma flux (CF), short-time energy ratio (STER) and fundamental frequency ratio (FFR) are defined by only one value per observation, where the FFR feature includes copies for one in the 4s-snippet calculated fundamental frequency ratio.

4.2.3 Resulting Feature Space

The basic feature space for this approach consists of 95 features and 70161 observations, which leads to a feature matrix $\boldsymbol{X}$ with size $[70161 \times 95]$. Again, each row of the feature matrix is characterized by a time frame of $N_{fft} = 1024$ samples and the hop-size is $R = 512$ (see Table 10).

The resulting feature set is illustrated in Fig. 4.32. In comparison to the first binary classification task the features LSTFT, MELS and MFCC are removed from the feature set. The remaining features C, CE, FFR and STER were calculated again for this classification task. In addition, the features MPDR, ZCR, COV, CM, CEM, CP, CC, CS, CF, STER and FFR are appended to the feature space.

There are 3 matrices with respect to the 12 chromagram pitch classes appended to the features space (C, CM and CP). The covariance matrix COV has 10 feature values, which are delivered as $N_{buf} - 1 = 39$ copies in the feature matrix of one snippet with a length of 4s.

Only $N_{buf} - 1 = 39$ values can be calculated in case of the feature CF, because the calculation of a value for the actual frame depends on the knowledge of the value of one preceding frame. Hence, it is only possible to transmit $N_{buf} - 1$ samples to the feature space.
In case of the FFR feature again 39 copies of the calculated value are transferred to the feature space. For all remaining features 39 different feature values for each time frame of approximately 0.2s are transmitted to the feature space.

4.2.4 Feature Selection

The features are again selected with the χ^2-statistics (see section 4.1.4). In Table 24 the first 20 ranked features are depicted. The selection shows, that the median-filtered chromagram (CM), the fundamental frequency ratio (FFR) and the short-time energy ratio (STER) give the highest values for the χ^2-statistics. However, when thinking of the *Doppler effect*, a consideration of the entire median-filtered chromagram makes sense. Another feature selection without consideration for the unfiltered chromagram showed, that also the feature ZCR ($\chi^2 = 3085.97$) improved the results in this classification task.

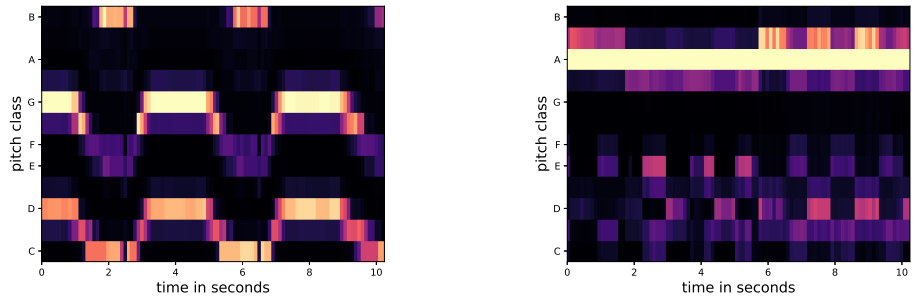
For this reason the chosen feature selection involves the features ZCR (1), CM (12) and FFR (1).

Count	Symbol	Number	χ^2
1	CM_G	32	10839.93
2	CM_C	25	10595.67
3	CM_F#	31	9312.47
4	CM_F	30	8917.08
5	CM_D	27	8769.64
6	CM_H	36	8710.08
7	C_G	20	8585.76
8	C_C	13	8368.51
9	CM_C#	26	8141.59
10	CM_E	29	8011.36
11	CM_D#	28	7631.8
12	FFR	55	7355
13	C_F#	19	6798.59
14	C_H	24	6555.32
15	STER	54	6378.74
16	C_F	18	6352.24
17	C_D	15	6250.57
18	C_C#	14	5913.24
19	C_E	17	5728.6
20	CM_G#	33	5636.58
⋮	⋮	⋮	⋮

Table 24 – Feature Ranking with the 20 highest scores of χ^2 statistics.

4.2.5 Classification Results

There is no self-recorded validation set for class S available. However, when evaluating the classification results of the existing self-recorded validation set (see section 3) with classes M and U a problem occurred with the microphones positioned in the engine compartment. In the recordings of class U the ventilation of the car temporarily produced a sound, which exhibits a similar sound characteristic like recorded sirene-sounds. Fig. 4.33 shows the median-filtered chromagram of the self-recorded validation set of classes M and U. If we compare Fig. 4.33b with Fig. 4.26a it gets understandable that class U could mistakenly be classified as a sirene. This led to a situation in which about 20% of class U is classified as class S.



(a) Median-filtered chromagram of class M

(b) Median-filtered chromagram of class U

Figure 4.33 – Exemplary time frames showing the median-filtered chromagram of class M (left) and class U (right) of the self-recorded validation set. In case of class U the median-filtered chromagram shows a *sirene-like* sound produced by the ventilation of the car, which leads to confusions with class S.

With the unfiltered chromagram this confusion was less pronounced, but on the other hand a distinction between the two sirene sounds (classes S and M) became worse. This classification task intends to demonstrate, that the two classes S and M can be distinguished in a more promising way. For this reason the data of the self-recorded validation set at those microphone positions was removed in the following classification results.

It is suggested for future experiments with car recordings to choose more suitable microphone positions, to prevent *sirene-like* urban noise sounds. The positioning of the microphones has a strong influence on the classification task and should be researched additionally. Hence, the provision of representative microphone recordings is the most decisive factor when developing a sound event detection algorithm for cars, especially if more than two classes are taken into account.

In this approach only the classifiers NB, KNN and SVMR were taken into account, which are described in the following paragraphs. Again all features are transformed to zero mean and unit variance before fitting the classification algorithms and also results of a k-fold cross validation are presented.

Naive Bayes Classifier Results for the NB classifier are listed in Tables 25, 26 and Fig. 4.34. In case of the self-recorded validation set more than 58% of class M is classified incorrectly as class S.

Data	Accuracy
Test data from training set	0.904
Cross validation with training set	0.88 ± 0.08
Basic validation set	0.868
Self-recorded validation set	0.571

Table 25 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier NB.

Class	Precision	Recall	F1 score	Quantity
U	0.977	0.981	0.979	8050
M	0.893	0.833	0.862	6704
S	0.825	0.88	0.852	6295

(a) Test data from training set (30%)

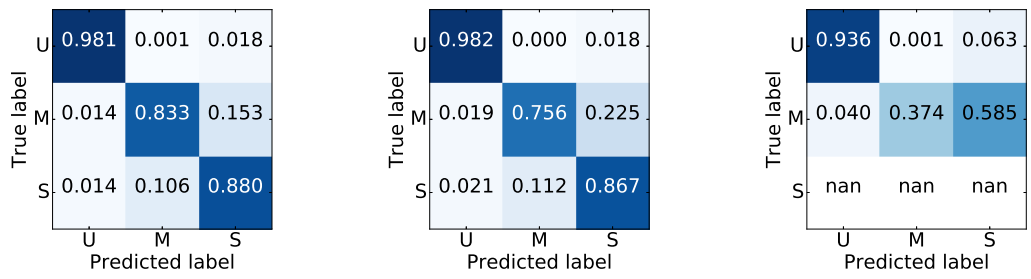
Class	Precision	Recall	F1 score	Quantity
U	0.961	0.982	0.971	9126
M	0.875	0.756	0.811	9165
S	0.775	0.867	0.819	8853

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.929	0.936	0.932	7987
M	0.998	0.374	0.585	14157

(c) Self-recorded validation data

Table 26 – Precision, Recall and F1 score of the classifier NB for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.34 – Confusion matrices of the different test data sets with the classifier NB.

K-Nearest-Neighbor In this case the KNN classifier was chosen with $k = \sqrt{N} \approx 221$. Results are listed in Tables 27, 28 and Fig. 4.35. Here, in case of the self-recorded validation set only 23% of class M is classified incorrectly as class S.

Data	Accuracy
Test data from training set	0.948
Cross validation with training set	0.86 ± 0.1
Basic validation set	0.906
Self-recorded validation set	0.794

Table 27 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier KNN.

Class	Precision	Recall	F1 score	Quantity
U	0.942	1	0.97	8050
M	0.952	0.924	0.937	6704
S	0.951	0.906	0.928	6295

(a) Test data from training set (30%)

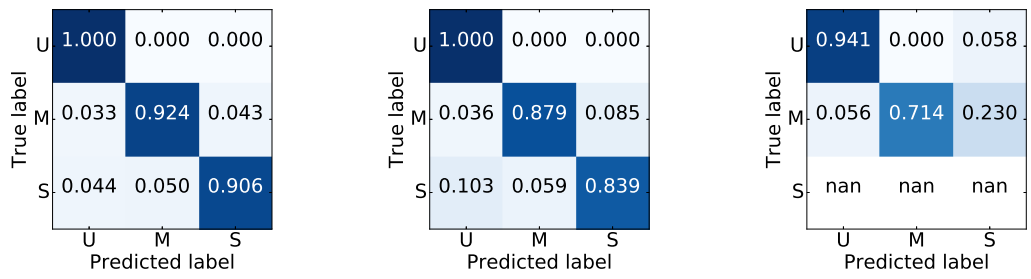
Class	Precision	Recall	F1 score	Quantity
U	0.88	1	0.936	9126
M	0.939	0.879	0.908	9165
S	0.905	0.839	0.871	8853

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.905	0.941	0.923	7987
M	0.9999	0.714	0.831	14157

(c) Self-recorded validation data

Table 28 – Precision, Recall and F1 score of the classifier KNN for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.35 – Confusion matrices of the different test data sets with the classifier KNN.

Support Vector Machine Results for a the classifier SVMR are listed in Table 29, Tables 30 and Fig. 4.36. The parameterization of C and γ is based on a grid search, which is depicted in Fig. 4.37.

Data	Accuracy
Test data from training set	0.996
Cross validation with training set	0.93 ± 0.06
Basic validation set	0.957
Self-recorded validation set	0.87

Table 29 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier SVMR.

Class	Precision	Recall	F1 score	Quantity
U	0.997	0.9999	0.998	8050
M	0.996	0.994	0.995	6704
S	0.995	0.994	0.995	6295

(a) Test data from training set (30%)

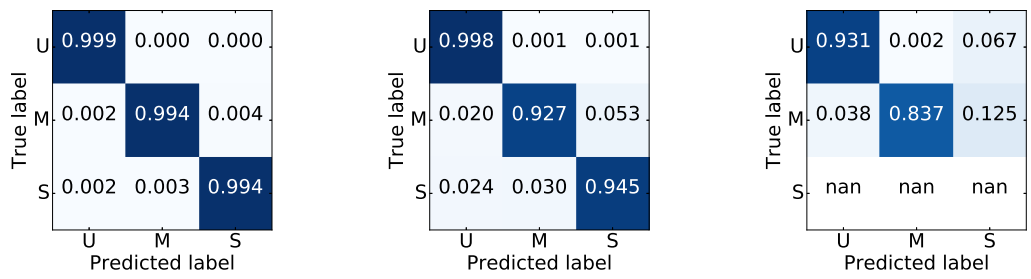
Class	Precision	Recall	F1 score	Quantity
U	0.958	0.998	0.978	9126
M	0.968	0.927	0.947	9165
S	0.945	0.945	0.945	8853

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.932	0.931	0.932	7987
M	0.999	0.837	0.91	14157

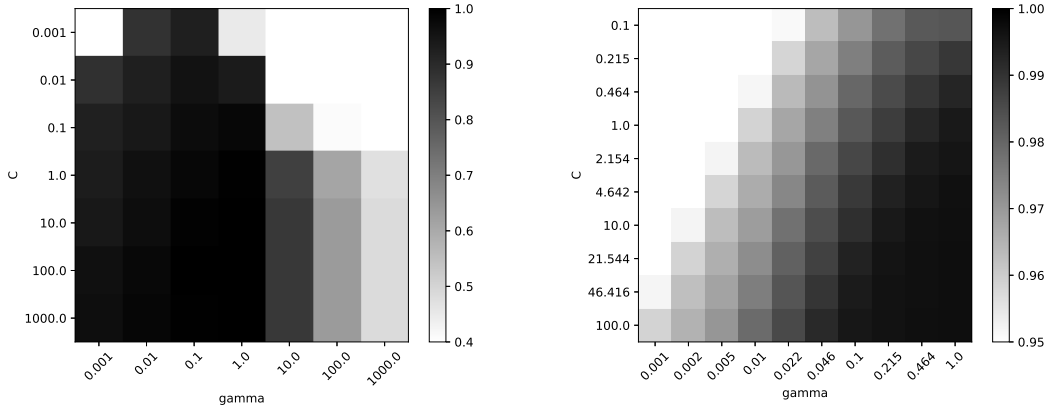
(c) Self-recorded validation data

Table 30 – Precision, Recall and F1 score of the classifier SVMR for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.36 – Confusion matrices of the different test data sets with the classifier SVMR.



(a) Grid search for best Recall (rough)

(b) Grid search for best Recall (fine)

Figure 4.37 – Grid search for the best over the three classes averaged Recall with the classifier SVMR. Best parameters are located in the range $\gamma = [0.001, 1]$ and $C = [0.1, 100]$ (left). Right figure shows a finer grid with $\gamma = [0.001, 1]$ and $C = [0.1, 100]$. The best parameters are chosen with $\gamma = 0.2$ and $C = 10$.

Summary Again, all models perfom very well on the test data from the training set and on the basic validation data set. Table 31 summarizes the most meaningful classification results for the self-recorded validation data set: With the classifier NB the Accuracy is only 57.1% and just 37.4% of the actual class M is predicted as class M. The classifier KNN gives a better distinction between classes M and S: 23% of the sounds of the actual class M are predicted as class S. With the classifier SVMR the results are most promising: 83.5% of the actual class M are predicted as class M and 12.5% of the actual class M are predicted as class S. This confusion is reasonable due to the fact, that the sirene sounds (classes M and S) have similar signal characteristics. Especially the sweeps at the start and at the end of the sound of a *Environmental Sirene* make it difficult to distinguish between the two classes. On the other hand a clear distinction between urban and sirene sounds is guaranteed.

Model	Accuracy	Precision (M)	Recall (M)	F1 score (M)
NB	0.571	0.998	0.374	0.585
KNN	0.794	0.999	0.712	0.831
SVMR	0.87	0.999	0.835	0.91

Table 31 – Summary of specific performance measures of the self-recorded validation data set for the applied models in the classification task with three classes.

4.3 Binary Classification with additional Features

In this approach the new developed features from the classification task with the three classes (see section 4.2.2) are now also used for the binary classification task again with the classes M and U. The preprocessing steps stay the same as in the already described classification tasks. In comparison with the first binary classification task, the feature space is different and also another feature selection algorithm is tested.

4.3.1 Feature Selection

This time, the features are selected by a combination of a *Sequential Forward Selection* (SFS) and a *Sequential Backward Selection* (SBS). The resulting selection algorithm is called *Plus-L Minus-R Selection* (LRS). Information about the calculation of this selection algorithm can be found in [53, 54]. This sequential feature selection algorithm tries to find an optimal subset of features by adding or rather removing features with respect to a relevance criterion (called filter-technique). Here, the objective function (statistical measurement, which describes the properties of a data set) of a feature subset $\boldsymbol{Y}$ is the trace ratio of the *within-class scatter matrix* $\boldsymbol{S}_w$ and the *between-class scatter matrix* $\boldsymbol{S}_b$:

$$J(\boldsymbol{Y}) = \frac{Tr\{\boldsymbol{S}_b\}}{Tr\{\boldsymbol{S}_w\}}. \tag{128}$$

The trace operator $Tr\{\cdot\}$ of a matrix calculates the sum over all diagonal elements.
The matrix $\boldsymbol{S}_w$ gives a measurement for the variances (compactness) within a class and is defined as

$$\boldsymbol{S}_w = \sum_{i=1}^{N_C} \frac{N_i}{N} \cdot \boldsymbol{C}_i, \tag{129}$$

where N is the total amount of observations, N_C the number of classes, N_i the amount of observations in class i and $\boldsymbol{C}_i$ the covariance matrix of class i.

The matrix $\boldsymbol{S}_b$ gives a measurement for the means (separability) between the classes and can be formalized as

$$\boldsymbol{S}_b = \sum_{i=1}^{N_C} \frac{N_i}{N} (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T, \tag{130}$$

where N_C is again the number of classes, $\boldsymbol{\mu}$ is the global mean of all classes and $\boldsymbol{\mu}_i$ the mean of class i.
Hence, the trace ratio gets maximal, if the average distance to the global mean ($Tr\{\boldsymbol{S}_b\}$) is high and the average variance over all classes ($Tr\{\boldsymbol{S}_w\}$) is small.

The LRS for the two classes is performed by executing the following steps:

— initialize an empty feature set $\boldsymbol{Y}_0$ (where the subscript $k = 0$ is a counter for the actual feature subset)
— selection of the first feature: Add the feature with the highest *Fisher's Ratio* [8] score to the feature set
— L times sequential adding of features $\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \boldsymbol{x}_{add}$, which maximizes the objective function $J(\boldsymbol{Y}_k + \boldsymbol{x})$
— R times sequential removing of features $\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k - \boldsymbol{x}_{remove}$, which minimizes the objective function $J(\boldsymbol{Y}_k - \boldsymbol{x})$
— stop when objective function converges

There is no theoretical basis about the choice of the parameters L and R. Here, the first 20 selected features with $L = 10$ forward steps and $R = 5$ backward steps are calculated and shown in Table 32.

Count	Symbol	Number	DP
1	STER	54	1.0728
2	COV_13	8	1.0737
3	COV_24	11	1.0737
4	COV_14	9	1.0737
5	COV_34	12	1.0737
6	COV_23	10	1.0737
7	COV_12	7	1.0737
8	COV_1	3	1.0737
9	ZCR	2	1.0737
10	CM_C	25	1.0731
11	COV_3	5	1.0737
12	COV_2	4	1.0737
13	COV_4	6	1.0737
14	CM_G	32	1.0732
15	CM_E	29	1.0735
16	CM_G#	33	1.0736
17	CM_D	27	1.0737
18	CM_C#	26	1.0738
19	C_C	13	1.0739
20	C_D	20	1.0739
⋮	⋮	⋮	⋮

Table 32 – Feature Ranking of the first 25 features with SFS and SBS feature selection with .

8. Definition of *Fisher's Ratio* for one feature in case of the classes M and U with the means μ_1 and μ_2 and the variances σ_1^2 and σ_2^2: FR $= \frac{(\mu_1-\mu_2)^2}{\sigma_1^2+\sigma_2^2}$

The selected features indicate, that a combination of STER, COV, ZCR, CM and C lead to good results. However, the selected features give an idea for a good feature set.

Experiments with the feature set showed, that a feature vector with MPDR (1), ZCR (1), C (12), CE (1), STER (1) and specific combinations of COV (here: a combination of 2 variances) lead to the most promising results with the self-recorded validation set. Even if the features CE and MPDR are not taken into account with the used feature selection algorithm (see Table 32), it could be established after many attempts, that the results improved with this final feature selection [9]. This leads to a final feature vector with 18 features.

For a binary classification the *de-noised* chromagram (CM) does not perform as well as the basic chromagram (C). This is again explained with the fact, that the self-recorded validation set temporarily shows a *sirene-like* sound (see section 4.2.5), which is supported by the median-filtered chromagram. The median-filtered chromagram enhances recurring frames, which in this case highlights the *sirene-like* behavior and leads to a confusion with class M. Hence, there is a danger, that the median-filtered chromagram performs worse in predicting class U correctly. On the other hand, when looking at the classification task with three classes, the median-filtered chromagram performs better in distinguishing classes M and S. This leads to a *trade-off*, which needs to be considered when developing an embedded system.

The covariance vector compares 4 one-second intervals of a snippet and it turned out, that specific combinations can be more helpful than taking all of the 10 measures of dispersion into account. Variances $\mathrm{Var}(\boldsymbol{s}_2)$ (COV_2) and $\mathrm{Var}(\boldsymbol{s}_3)$ (COV_3) lead to a good statistic representation of a snippet. Hence, those measures outperformed in the classification task with the self-recorded validation set. Therefore, this feature takes each of the *inner* variances of a snippet (variances of $1\mathrm{s}\ldots 2\mathrm{s}$ and $2\mathrm{s}\ldots 3\mathrm{s}$).

4.3.2 Classification Results

This third approach combines the gained knowledge of the first two approaches to introduce better classification results in case of the binary classification task.

Models for with the three classifiers NB, KNN and SVMR are designed and the results are described in the following paragraphs. Again all features are transformed to zero mean and unit variance before fitting the classification algorithms and also results of a k-fold cross validation are presented.

9. Note, that regardless of the ratings of the used feature selection algorithms at all times further attempts were made *by hand*, because it was established, that the results could always be further improved. Hence, the final feature selection always additionally depended on the achieved results (especially with regard to the self-recorded validation set).

Naive Bayes Classifier Results of the NB classifier are listed in Tables 33, 34 and Fig. 4.39. This classifier predicts class M $\approx 2\%$ better than in the first approach.

Data	Accuracy
Test data from training set	0.987
Cross validation with training set	0.98 ± 0.03
Basic validation set	0.99
Self-recorded validation set	0.859

Table 33 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier NB.

Class	Precision	Recall	F1 score	Quantity
U	0.983	0.993	0.988	8050
M	0.991	0.979	0.985	6704

(a) Test data from training set (30%)

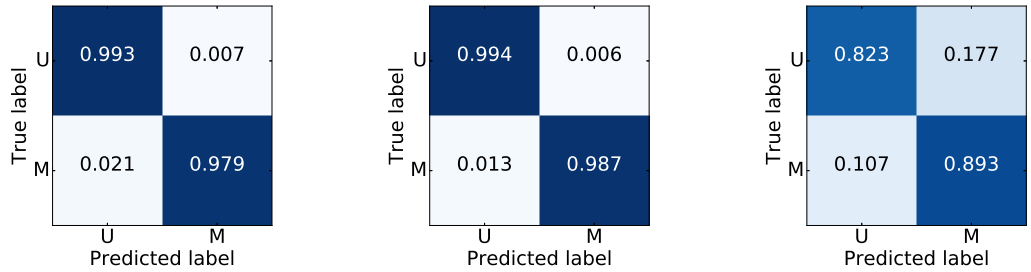
Class	Precision	Recall	F1 score	Quantity
U	0.987	0.994	0.99	9126
M	0.994	0.987	0.99	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.88	0.823	0.85	13455
M	0.841	0.893	0.866	14157

(c) Self-recorded validation data

Table 34 – Precision, Recall and F1 score of the classifier NB for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.38 – Confusion matrices of the different test data sets with the classifier NB.

K-Nearest-Neighbor Results of the KNN classifier with $k = \sqrt{N} = 185$ are listed in Tables 35, 36 and Fig. 4.39. In comparison to the first approach the results in case of the self-recorded validation set are a bit better.

Data	Accuracy
Test data from training set	0.982
Cross validation with training set	0.94 ± 0.02
Basic validation set	0.977
Self-recorded validation set	0.876

Table 35 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier KNN.

Class	Precision	Recall	F1 score	Quantity
U	0.969	0.999	0.984	8050
M	0.998	0.962	0.98	6704

(a) Test data from training set (30%)

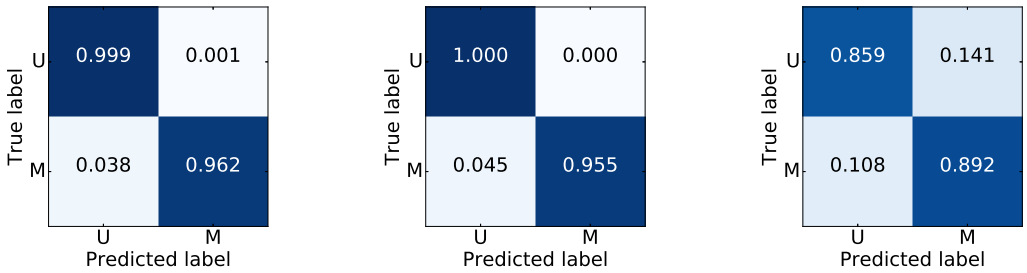
Class	Precision	Recall	F1 score	Quantity
U	0.957	1	0.978	9126
M	1	0.955	0.977	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.883	0.859	0.871	13455
M	0.87	0.892	0.881	14157

(c) Self-recorded validation data

Table 36 – Precision, Recall and F1 score of the classifier KNN for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.39 – Confusion matrices of the different test data sets with the classifier KNN.

Support Vector Machine Results for the SVMR are listed in Tables 37, 38 and Fig. 4.40. Again, the parameterization of C and γ is based on a grid search, which is depicted in Fig. 4.41. This classifier gives the best results, because in case of the self-recorded validation set the Recall of both classes is greater than 91%.

Data	Accuracy
Test data from training set	0.996
Cross validation with training set	0.98 ± 0.02
Basic validation set	0.994
Self-recorded validation set	0.919

Table 37 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier NB.

Class	Precision	Recall	F1 score	Quantity
U	0.993	0.999	0.996	8050
M	0.999	0.991	0.995	6704

(a) Test data from training set (30%)

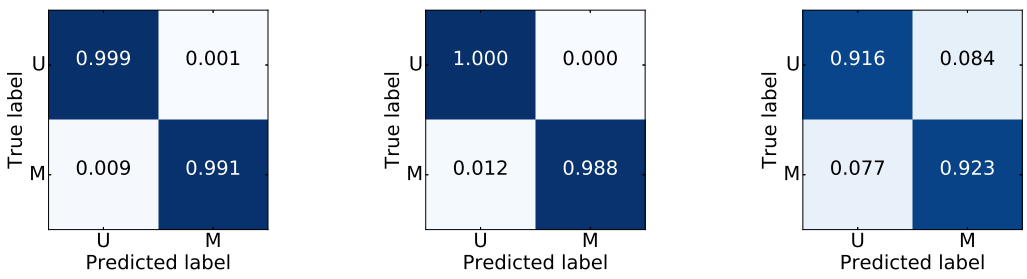
Class	Precision	Recall	F1 score	Quantity
U	0.988	1	0.994	9126
M	1	0.988	0.994	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.918	0.916	0.917	13455
M	0.92	0.923	0.922	14157

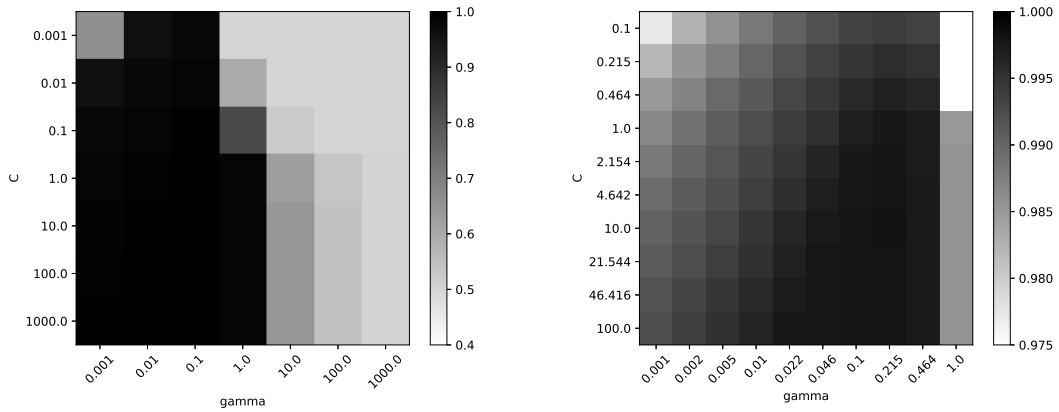
(c) Self-recorded validation data

Table 38 – Precision, Recall and F1 score of the classifier SVMR for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 4.40 – Confusion matrices of the different test data sets with the classifier SVMR.



(a) Grid search for best Recall (rough)

(b) Grid search for best Recall (fine)

Figure 4.41 – Grid search for the best over the two classes averaged Recall with the classifier SVMR. A rough analysis in the range $\gamma = [0.001, 1000]$ and $C = [0.001, 1000]$ shows, that best parameters are located in the range $\gamma = [0.001, 1]$ and $C = [0.1, 1000]$ (left). Right figure shows a finer grid with $\gamma = [0.001, 1]$ and $C = [0.1, 100]$. However with respect to the self-recorded validation set, the best parameters are chosen with $\gamma = 0.1$ and $C = 0.3$ (the parameter C is chosen a little lower, because this lead to better results with the self-recorded validation set and on the other hand this choice prevents the risk of overfitting.).

Summary The results on the test data from the training set and on the basic validation set are again very good. The key results of the self-recorded validation data set are listed in Table 39: The Accuracies of all classifiers are greater than 85%. Also the measures Precision, Recall and F1 score of class M have scores greater than 84% in case of all classifiers. The classifiers NB and KNN perform similar well, but the classifier SVMR outperforms the other results, because the score of each performance measure is greater than 91%. It can be expected, that this developed algorithm is able to provide good results when it is integrated in a car.

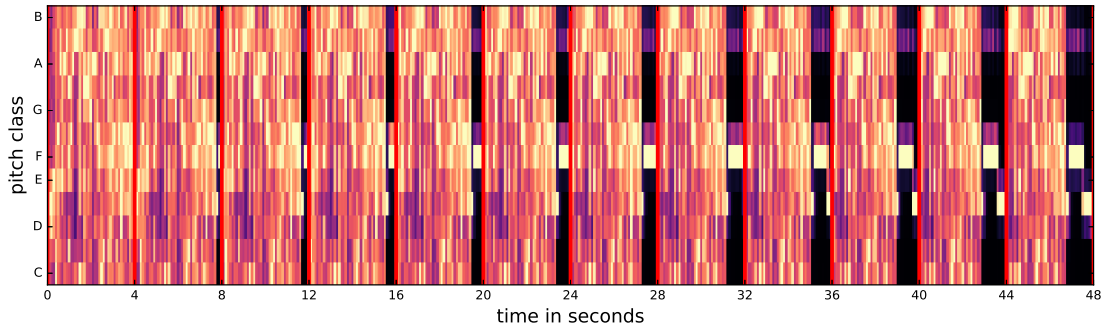
But it is important to consider carefully where and how to place the microphones on or in a car, because too much interfering noise in the input signal (e.g. ventilation of the car, wind noise at high velocities, . . .) makes a distinction more difficult or nearly impossible. One could say, all developed algorithms are *living on* usable microphone recordings.

Model	Accuracy	Precision (M)	Recall (M)	F1 score (M)
NB	0.859	0.841	0.893	0.866
KNN	0.876	0.87	0.892	0.881
SVMR	0.919	0.92	0.923	0.922

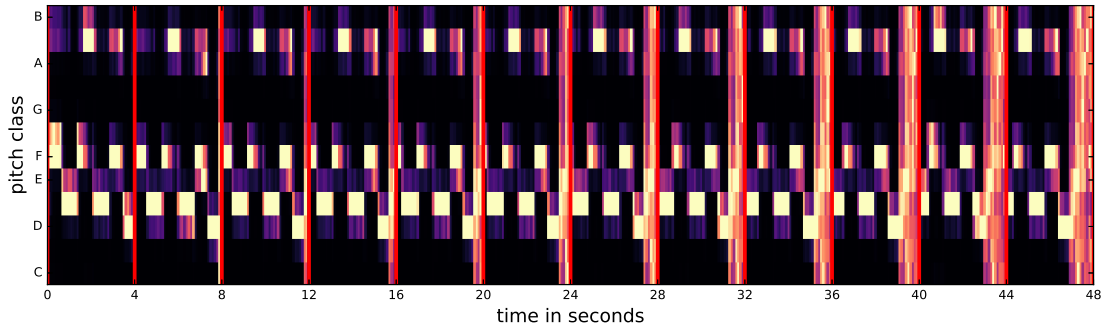
Table 39 – Summary of specific performance measures of the self-recorded validation data set for the applied models in the improved binary classification task with additional features.

4.3.3 Transition results with Support Vector Machine

The binary classification task with additional features and a support vector machine with radial kernel function leads to the most promising results. However, it is interesting to analyze classification results for the best developed classifier with different amounts of class M or class U within an analysis window of 4s (note that for this additional analysis for each snippet now all $N = 40$ feature vectors were transferred to the feature space; see section 4.1.3 for more details). The described algorithms classify 4s-snippets of either class M or class U. It does not deal with transitions between both classes (e.g. classification of a 4s-snippet with 2s amount of class M and 2s amount of class U). This paragraph describes an exemplary transition of two audio files from the validation data set, where either a 4s-snippet of class U passes over to a 4s-snippet of class M or a 4s-snippet of class M passes over to a 4s-snippet of class U (see Fig. 4.42).



(a) Chromagram illustrating transitions from class U to class M



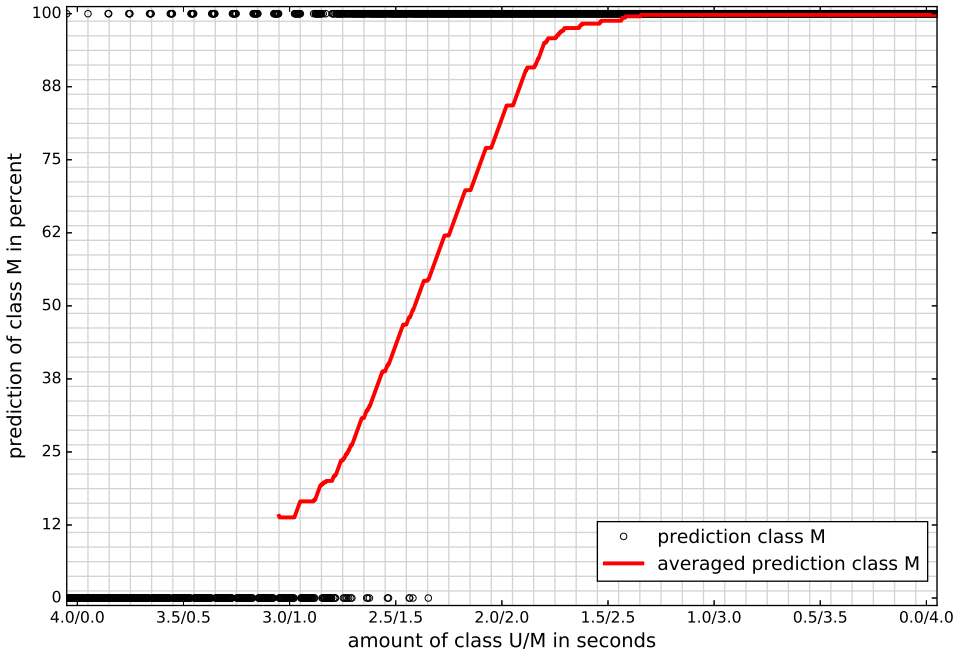
(b) Chromagam illustrating transitions from class M to class U

Figure 4.42 – Chromagrams illustrating the first 12 types of transitions from class U to class M (a) and from class M to class U (b). Two exemplary audio files are taken from the validation data set with each having a length of 4s. Red vertical lines indicate each analysis window with a hop size of 0.1s.

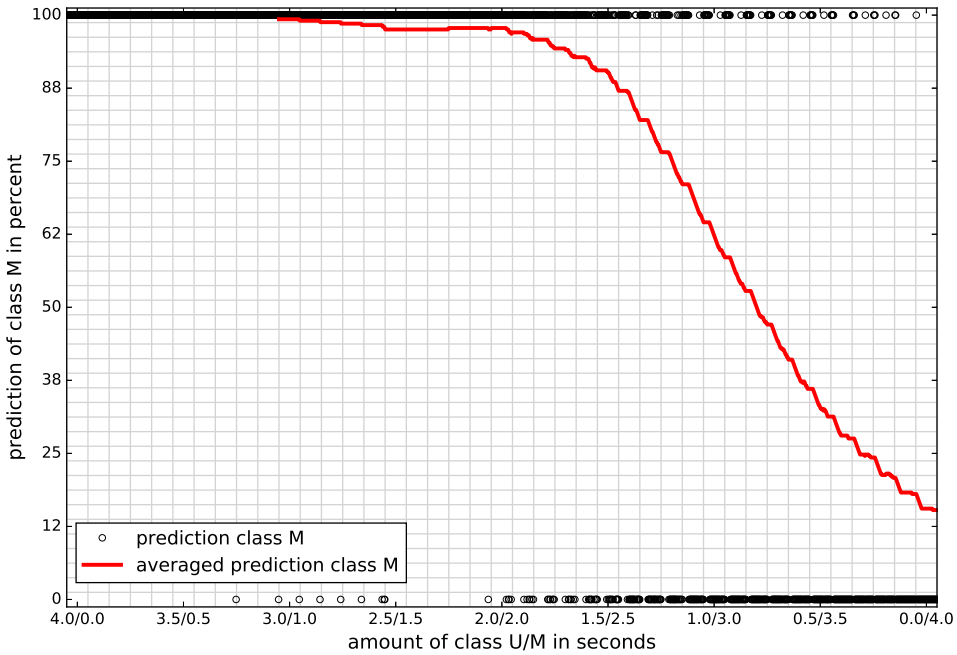
Classification results for the two types of transitions are illustrated in Fig. 4.43:
Fig. 4.43a shows the prediction of class M with a transition from class U to class M. Going through each vertical block (*columns*), the amount of class M within a 4s-snippet increases (per step 0.1s less data of class U and 0.1s more data of class M). After 1s, there are 3s

of class U and $1\mathrm{s}$ of class M within an analysis window. At this point, a moving average filter over $1\mathrm{s}$ indicates, that approximately 12% of the preceding predictions belong to class M. When approximately $1.6\mathrm{s}$ of the audio snippet involves parts of a *Martinshorn*, 50% of the predictions of the preceding second involve class M. In this exemplary case an amount of $1.8\mathrm{s}$ of class M is sufficient that only class M is predicted. With the same amount of class U and class M approximately 80% of the moving averaged preceding predictions indicate that a sirene is present.

Fig. 4.43b shows the prediction of class M with a transition from class M to class U. With an amount of $4\mathrm{s}\ldots2\mathrm{s}$ of class M primary - with a few minor exceptions - a *Martinshorn* is predicted. From then on class U is increasingly predicted.



(a) Predicted class M when passing over from class U to class M



(b) Predicted class M when passing over from class M to class U

Figure 4.43 – Real and averaged predicted class M for a transition from class U to class M (a) and a transition from class M to class U (b). X-axis describes the amount of class M or class U within an analysis window (*grid columns*). Predicted classes are averaged with a moving average filter over $N_{avg} = 400$ training samples ($\hat{=}1\text{s}$).

4.3.4 Computational Costs

In this section the calculation efforts of each used classifier of this approach are compared. In this approach $m = 18$ features are taken into account. Before classification, the same feature set needs to be calculated with new unseen data $\boldsymbol{x}_{new}$. Assuming those new features have been calculated, it is interesting to describe how many calculations in terms of additions, subtractions, multiplications and divisions each classifier requires before a classification result can be presented.

Memory requirements of each classifier and the amount of required instructions is described in the next paragraphs. Within a 4s-snippet $(40 \cdot 18) = 720$ feature values in double-precision floating-point format need to be calculated. First of all, this leads to a memory requirement of $(40 \cdot 18 \cdot 8 \cdot 1024^{-1})\,\mathrm{kB} \approx 5.63\mathrm{kB}$ (when considering 64 Bit double-precision floating-point format).

Naive Bayes Classifier The NB classifier calculates the means and variances of the gaussian likelihood functions

$$p(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot e^{-\frac{(x_j-\mu_y)^2}{2\sigma_y^2}} \tag{131}$$

for each feature of each class. After that, the MAP estimation

$$y_{MAP} = \arg\max_y p(y) \prod_{j=1}^{m} p(x_j|y) \tag{132}$$

gives the estimated label.

The final model from the scikit-learn toolkit provides the prior probabilities of the two classes (`model.class_prior_`), the means for each feature of each class (`model.theta_`) and the corresponding variances (`model.sigma_`).

In this developed final model, for one input vector $\boldsymbol{x}_{new}$ with $m = 18$ features the following calculations need to be done $N_{class} = 2$ (number of classes) times:

- 18 subtractions: $\hat{x}_j = x_j - \mu$
- 18 squares/multiplications and 17 additions: $\tilde{x} = \hat{x}_1^2 + \hat{x}_2^2 + \cdots + \hat{x}_{22}^2$
- 1 multiplication, 1 division and 1 function call: $f(\tilde{x}) = e^{-\frac{\tilde{x}}{2\sigma^2}}$
- 3 multiplications, 1 square root and 1 division: $\frac{1}{\sqrt{2\pi\sigma^2}} \cdot f(\tilde{x})$
- 18 multiplications: $p(y) \prod_{j=1}^{m} p(x_j|y)$

After that, the MAP decision gives the predicted label. Consequently one prediction with the developed NB classifier needs $(2 \cdot 18) = 36$ subtractions, $(2 \cdot 17) = 34$ additions, $[2 \cdot (18 + 3 + 1)] = 44$ multiplications and $[2 \cdot (1 + 1)] = 4$ divisions.

If the exponential function is a 5th-order exponential series

$$e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5, \tag{133}$$

this would imply further 5 additions, 10 multiplications (powers) and 4 multiplications for each input sample $\boldsymbol{x}_{new}$. This results in $(2 \cdot 5) = 10$ additions and $[2 \cdot (10 + 4)] = 28$ multiplications.

The 5th-order Taylor series of a square root is

$$\sqrt{x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 + \frac{7}{256}x^5. \tag{134}$$

Consequently further 3 additions, 2 subtractions, 10 multiplications (powers) and 5 multiplications are required. This results in another $(2{\cdot}3) = 6$ additions, $(2{\cdot}2) = 4$ subtractions and $[2 \cdot (10 + 5)] = 30$ multiplications.

This leads to $(36 + 34 + 44 + 4 + 6 + 4 + 30) = 158$ instructions. If e.g. every 0.1s a new snippet with 40 frames (feature vectors) is read, this would result in $\frac{(158{\cdot}40)}{0.1\text{s}} = 63200$ instructions per second (IPS) or rather 0.0632 millions of instructions per second (MIPS) [10].

If the stored values of the model have a double-precision floating-point format, the memory requirement for the 2 prior probabilities, the $[N_{class} \times m] = [2 \times 18]$ mean values and the $[N_{class} \times m] = [2 \times 18]$ variances is $[(2 + 2 \cdot 18 + 2 \cdot 18) \cdot 8 \cdot 1024^{-1}]\text{kB} \approx 0.58\text{kB}$.

K-Nearest-Neighbor Here, the KNN classifier calculates the *Minkowski distances* with order $p = 2$. The distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{x}_{new}$ can be formalized as:

$$D(x_i, x_{new,i}) = \left(\sum_{i=1}^{m} |x_i - x_{new,i}|^p \right)^{\frac{1}{p}}. \tag{135}$$

For the prediction the $k = 3$ nearest neighbors of a new unseen feature vector $\boldsymbol{x}_{new}$ are detected and a majority decision determines the classification.

For one unseen sample $\boldsymbol{x}_{new}$ with m feature values the following calculations need to be done $N_{train} = 34425$ times:

— 18 subtractions: $\hat{x}_i = x_i - x_{new,i}$
— 18 squares/multiplications and 17 additions: $\tilde{x} = \hat{x}_1^2 + \hat{x}_2^2 + \cdots + \hat{x}_{22}^2$
— 1 square root: $\sqrt{\tilde{x}}$

This leads to $(34425 \cdot 18) = 619650$ subtractions, $(34425 \cdot 18) = 619650$ multiplications and $(34425 \cdot 17) = 585225$ additions.

The square root expressed again as a 5th-order Taylor series needs further 3 additions, 2 subtractions, 10 multiplications (powers) and 5 multiplications (see above). Therefore,

10. Note, that after 0.1s only one new training sample could be added to the 39 preceding frames. Here, this would result in $\frac{158}{0.1\text{s}} = 1580$ instructions per second (the same applies to the other machine learning algorithms).

another $(34425 \cdot 3) = 103275$ additions, $(34425 \cdot 2) = 68850$ subtractions and $[34425 \cdot (10 + 5)] = 516375$ multiplications need to be considered.

This leads to $(619650 + 619650 + 585225 + 103275 + 68850 + 516375) = 2513025$ instructions. With a hop of 0.1s and 40 frames per snippet this results in $\frac{(2513025 \cdot 40)}{0.1\text{s}} = 100521000$ IPS or rather 1005.21 MIPS.

A KNN classifier stores 70% of the training set (the other 30% are the test set), where the training set involves 30% of the basic data set (see section 3). Here, this set has a size of $[N_{train} \times m] = [34425 \times 18]$. If each value is stored in a double-precision floating-point format, the memory requirement is $34425 \cdot 18 \cdot 8 \cdot 1024^{-1}\text{kB} \approx 4841\text{kB}$.

Due to the high computational costs with the KNN classifier, which compares the entire training set to a new unseen data vector another attempt with a slight deterioration of the results (self-recorded validation recall U: 83.8%; self-recorded validation recall M: 80.9%) can be proposed: Per class each feature of the training set with $N_{train} = 34425$ is median-filtered resulting in a modified training feature set with two class-related vectors ($N_{train,new} = 2$). This would lead to $2513025 \cdot \frac{2}{34425} \cdot \frac{40}{0.1\text{s}} = 58400$ IPS and a memory requirement of $2 \cdot 18 \cdot 8 \cdot 1024^{-}1\text{kB} \approx 0.281\text{kB}$. However, this attempt was not further investigated and solely provides another calculation method to save computational costs.

Support Vector Machine A radial kernel is used for the support vector machine (see section 2). The result of each radial kernel function

$$K(\boldsymbol{x}_i, \boldsymbol{x}_{new}) = e^{-\gamma \cdot ||\boldsymbol{x}_i - \boldsymbol{x}_{new}||^2}, \tag{136}$$

with $i = 1, \ldots, N_{SV}$ and N_{SV} as the amount of support vectors needs to be multiplied and summed up to get the decision

$$g(\boldsymbol{x}_{new}) = g\left(b + \sum_{i}^{N_{SV}} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_{new})\right). \tag{137}$$

The amount of support vectors depends on the size of the training set and the parameterization of the SVM (penalty parameter C and kernel coefficient γ).

The final model from the scikit-learn toolkit provides the index of each support vector (`model.support_`), the constant b in the decision function (`model.intercept_`) and the combined coefficients $a = \alpha_i y_i$ (`model.dual_coef_`).

In this developed final mode, for one input sample $\boldsymbol{x}_{new}$ with $m = 18$ features the following calculations need to be done $N_{SV} = 1213$ times:

— 18 subtractions: $\hat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_{new}$
— 18 squares/multiplications and 17 additions: $\tilde{x} = \hat{x}_1^2 + \hat{x}_2^2 + \cdots + \hat{x}_{22}^2$
— 1 multiplication and 1 function call: $f(\tilde{x}) = e^{-\gamma \cdot \tilde{x}}$
— 1 multiplication: $a \cdot f(\tilde{x})$

After that, the results of those $N_{SV} = 1213$ calculations need to be summed up and also the constant b needs to be added. Hence, this leads to $N_{SV} = 1213$ further Additions. In

the end the decision rule gives the predicted label: $g(z) = 1$ for $z \geq 0$ and $g(z) = -1$ otherwise. Consequently one prediction with the developed SVMR needs $(1213 \cdot 18) = 21834$ subtractions, $[1213 \cdot (17+1)+1213] = 23047$ additions and $[1213 \cdot (18+1+1)] = 24260$ multiplications.

If the exponential function is a 5th-order exponential series

$$e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5, \tag{138}$$

this would imply further 5 additions, 10 multiplications (powers) and 4 multiplications for each input sample $\boldsymbol{x}_{new}$. Again, repeating those calculations $N_{SV} = 1213$ times, this results in further $(1213 \cdot 5) = 6065$ additions and $[1213 \cdot (10+4)] = 16982$ multiplications.

Hence, this leads to $(21834 + 23047 + 24260 + 6065 + 16982) = 92188$ instructions. If every 0.1s a new snippet with 40 frames (feature vectors) is read, this would result in $\frac{(92188 \cdot 40)}{0.1\text{s}} = 36875200$ IPS or rather 36.8752 MIPS.

On the other hand the developed SVMR model stores a matrix with $[N_{SV} \times m] = [1213 \times 18]$ values in double-precision floating-point format. Therefore, the memory requirement for the support vectors of the model is $(1213 \cdot 18 \cdot 8 \cdot 1024^{-1})\,\text{kB} \approx 170.58\text{kB}$. Additionally the constant b and the parameter γ are stored in the model.

Summary In Table 40 the memory requirement and the MIPS of each of the three classifiers are summarized. A calculation with the NB classifier needs the least amount of memory and also very few MIPS. The KNN classifier needs the most memory and also has the most MIPS, because an unseen feature vector is compared with each vector of the training set.

Classifier	Memory Requirement [kB]	MIPS
NB	0.58	0.0632
KNN	4841	1005.21
SVMR	170.58	36.8752

Table 40 – Memory requirement and MIPS of each classifier

In any case also the calculation and the memory requirement of the feature space ($\approx 5.63\text{kB}$) of each snippet has to be respected.

5 Deep Learning Approach

The best results of the binary classification task (see section 4.3) show, that suitable solutions are achievable with machine learning classifiers. However, it is interesting to search for a comparable solution when considering the idea of *deep learning*. Hence, this section introduces a neural net model, which keeps up with the best binary classification results.

5.1 Binary Classification with a Neural Net

The binary classification task with additional features provided the best classification results for classes U and M with the classifier SVMR (see section 4.3). Hence, the same features (see section 4.3.1) are used to compare those results with the classification results of a *neural net* (classification with a Multi-Layer Perceptron). It can be shown, that the found results are comparable to this best machine learning approach.

The following paragraph describes a grid search to find a suitable model with a reasonable network structure. By default, in case of binary classification the scikit-learn toolkit uses a logistic sigmoid function (see Eq. 15) at the output layer.

5.1.1 Multi-Layer Perceptron Grid Search

There are many different neural network structures with e.g. different hidden layer sizes, activation functions, penalty values for the regularization term, learning rates or optimization functions. Instead of testing one specific topology of a neural net given in the literature, a grid search with different combinations of the indicated terms was performed.

One found *rule of thumb* recommends the amount of neurons in the hidden layers to be between the input layer size (amount of input features) and the output layer size (one neuron in case of a binary classification task) [55]. This idea was considered in this deep learning approach. The amount of hidden layers depends on the complexity of the classification task. For reproducibility, the following descriptions of the parameters are related to the nomenclature of the scikit-learn toolkit.

To find a suitable model a grid search with the following parameter space was made [11]:

— one hidden layer with size: (i) $\forall i \in 2 \dots 21$
— two hidden layers with sizes: (i, j) $\forall i, j \in 2 \dots 21$
— three hidden layers with sizes: (i, j, k) $\forall i, j, k \in 2 \dots 21$
— three activation functions: *tanh*, *logistic* and *relu*
— 4 penalty values α for the regularization term: $[0.0001, 0.001, 0.1, 1, 10]$

11. Note, that this grid search was based on a slightly different feature selection with 22 input features. The 4 additional features were: $|\mathrm{Cov}(\boldsymbol{s}_1, \boldsymbol{s}_2)|$ (COV_12), $|\mathrm{Cov}(\boldsymbol{s}_2, \boldsymbol{s}_4)|$ (COV_24), $|\mathrm{Cov}(\boldsymbol{s}_1, \boldsymbol{s}_3)|$ (COV_13) and $|\mathrm{Cov}(\boldsymbol{s}_3, \boldsymbol{s}_4)|$ (COV_34). But due to the results of the feature selection (see table 32) it is not absolutely necessary to keep those features in the selection. However, the developed models can be used as an indication for suitable neural net models, because the remaining 18 features were also part of the initial feature set.

— two learning rates: *constant* and *adaptive*
— two *solver* (optimization functions): *sgd* and *adam*

The parameters were updated with mini-batch training ($N_{mb} = 200$) and by default the number of epochs was 200. If the loss function converges earlier the learning process stops automatically. A ranking of the models is provided with the best over the two classes averaged Recall as a performance measure. Each model was evaluated with a k-fold cross validation with $k = 3$. Here, 3 randomized folds involving 30% of the training data as test data are used. This results in $[20 + (20 \cdot 20) + (20 \cdot 20 \cdot 20)] \cdot 3 \cdot 5 \cdot 2 \cdot 2 = 505200$ tested model designs. It turned out, that the Recalls of 147 models with one hidden layer, 5549 models with two hidden layers and 110529 with three hidden layers are greater or equal than 95%. Hence, a suitable solution can also be found with only one hidden layer.

Therefore, the 50 best models with one hidden layer were tested with the self-recorded validation set resulting in a chosen suitable model with the following parameters:

— hidden layer with size: (17)
— activation function: *logistic*
— penalty value α for the regularization term: $[0.0001]$
— learning rate: *constant*
— *solver* (optimization function): *adam*

In the scikit-learn toolkit the *constant* learning rate has the value $\eta = 0.001$ by default. In case of an *adaptive* learning rate this value would be kept constant as long as the loss decreases or it would be divided by 5, if it is not at least decreased by a tolerance value of 0.0001 [56].

The results in Section 5.1.2 show, that all performance measures of the MLP classifier score as well as the performance measures of the SVMR (see section 4.3). Hence, also a suitable solution of the classification task with a neural net can be proposed. For the prediction of one new input sample $\boldsymbol{x}_{new}$ the following calculations (forward propagation) need to be done (see section 2.2.1):

$$\underset{1\times17}{\boldsymbol{a}^{(h)}} = \Phi\left(\underset{1\times18}{\boldsymbol{x}_{new}} \cdot \underset{18\times17}{\boldsymbol{W}^{(h)}} + \underset{1\times17}{\boldsymbol{b}^{(h)}}\right),$$
$$\underset{1\times1}{\boldsymbol{a}^{(out)}} = \Phi\left(\underset{1\times17}{\boldsymbol{a}^{(h)}} \cdot \underset{17\times1}{\boldsymbol{w}^{(out)}} + \underset{1\times1}{\boldsymbol{b}^{(out)}}\right),$$

where $\Phi(z)$ is the logistic sigmoid function, which at the output estimates the probability $a^{(out)}$ for the feature vector $\boldsymbol{x}_{new}$ belonging to class $y = +1$ (M).

5.1.2 Classification Results

Again all features were transformed to zero mean and unit variance before fitting the classification algorithms and also results of a k-fold cross validation are presented.

Multi-Layer Perceptron Classification results with a Multi-Layer Perceptron (MLP) are listed in tables 41, 42 and Fig. 5.1. In case of the self-recorded validation set the measures Precision, Recall and F1 score are all greater than 90% for both classes.

Data	Accuracy
Test data from training set	0.997
Cross-Validation with training set	0.98 ± 0.04
Basic validation set	0.995
Self-recorded validation set	0.913

Table 41 – Accuracies for test data from the training set (30%), a k-fold cross validation with $k = 5$ (mean value and twofold standard deviation), a basic validation set and the self-recorded validation set in case of the classifier MLP.

Class	Precision	Recall	F1 score	Quantity
U	0.996	0.999	0.997	8050
M	0.999	0.995	0.997	6704

(a) Test data from training set (30%)

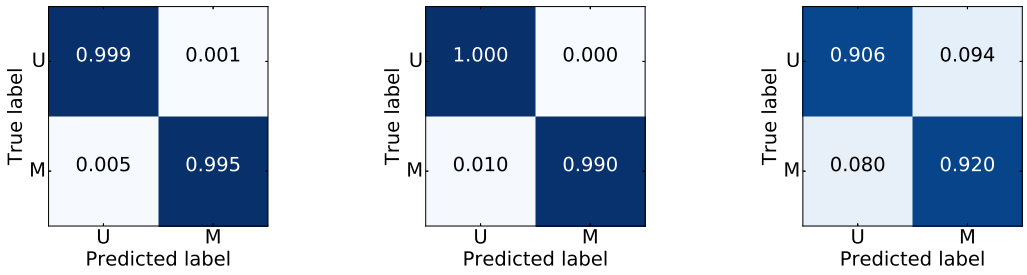
Class	Precision	Recall	F1 score	Quantity
U	0.99	1	0.995	9126
M	1	0.99	0.995	9165

(b) Basic validation data

Class	Precision	Recall	F1 score	Quantity
U	0.915	0.906	0.911	13455
M	0.912	0.92	0.916	14157

(c) Self-recorded validation data

Table 42 – Precision, Recall and F1 score of the classifier MLP for the different test sets with respective amounts of feature vectors.



(a) Test data from training set (b) Basic validation data (c) Self-recorded validation data

Figure 5.1 – Confusion matrices of the different test data sets with the classifier MLP.

6 Summary

Within the scope of this thesis several classification algorithms to distinguish between sirene sounds and urban sounds were developed. Three approaches provide a solution for a binary classification task to predict a *Martinshorn* correctly and one approach gives a solution for a classification task with three classes to predict a *Martinshorn* and *Environmental Sirenes*. The approaches involve different state-of-the-art machine learning algorithms: Perceptron, Logistic Regression, Naive Bayes Classifier, K-Nearest-Neigbor classifier, Support Vector Machine and Multi-Layer-Perceptron.

It turned out, that with the provision of a sufficient feature space good results can be achieved. Especially features, which are strongly related to the specific signal characteristics of sirenes (e.g. the ratio of the fundamental frequencies, the energy distributions in the spectrum or the chromagram) play an important role for the classification task.

The classification task with three classes showed, that it is also possible to distinguish between the two different sirene sounds, but a few confusions must be accepted. This is due to the fact, that there are similarities in the signal characteristics of sirene sounds.

A classification with a Support Vector Machine with a radial kernel lead to the best binary classification result and provides very good performance measures. On the other hand it appeared, that also the classification with the same feature set with a Multi-Layer-Perceptron gives comparable results. This is surely due to the reason, that there are many different possibilities to configure the structure and the parameters of a neural net.

In the end transition results with the best binary classification task are presented. It was shown, that an analysis window of approximately $2\mathrm{s}$ is also enough to predict a *Martinshorn* correctly.

References

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

[2] D. M. W. Powers, "Evaluation: From precision, recall and f-factor to roc, inrofmedness, markedness & correlation," 2007.

[3] "Precision and recall." [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall

[4] "How to construct a confusion matrix in LaTeX?" [Online]. Available: https://tex.stackexchange.com/questions/20267/how-to-construct-a-confusion-matrix-in-latex

[5] "F1 score." [Online]. Available: https://en.wikipedia.org/wiki/F1_score

[6] "Harmonic mean." [Online]. Available: https://en.wikipedia.org/wiki/Harmonic_mean

[7] "Scikit Learn Documentation - Underfitting vs. Overfitting." [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

[8] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[9] S. Raschka and V. Mirjalili, *Python Machine Learning*, 2015.

[10] F. Rosenblatt, "The perceptron: A perceiving and recognizing automaton," 1957.

[11] J. Hao and J. L. Priestley, "A comparison of machine learning techniques and logistic regression method for the prediction of past-due amount," *Grey Literature from PhD Candidates*, no. 1.

[12] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC Medical Genomics*, 2018.

[13] T. Bayes, "An essay towards a problem in the doctrine of chances," *Phil. Trans. of the Royal Soc. of London*, vol. 53, pp. 370–418, 1763.

[14] H. Zhang, "Proceedings of the seventeenth international florida artificial intelligence research society conference, miami beach, florida, usa," in *The Optimality of Naive Bayes*, 2004.

[15] "Scikit Learn Documentation - 1.9 Naive Bayes." [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html

[16] A. B. Hassanat, M. A. Abbadi, G. A. Altarawneh, and A. A. Alhasanat, "Solving the problem of the k parameter in the knn classifier using an ensemble learning approach," vol. 12, 2014.

[17] "Scikit Learn Documentation - 1.6. Nearest Neighbors." [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html

[18] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discover*, vol. 2, pp. 121–167, 1998.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[21] A. L. Mass, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[22] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," 2017.

[23] "Cross entropy." [Online]. Available: https://en.wikipedia.org/wiki/Cross_entropy

[24] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2015.

[25] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," vol. 323, 1986, pp. 533–536.

[26] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, 1998.

[27] J. Frankel and J. Schwartz, "Reaper - digital audio workstation," Cockos Inc. [Online]. Available: https://www.reaper.fm/

[28] Stanza (artist), "Soundcities - online open source database of city sounds." [Online]. Available: http://www.soundcities.com

[29] C. Hurley, S. Chen, and J. Karim, "YouTube - Video-sharing website," Google Inc. [Online]. Available: https://www.youtube.com/

[30] HAZ-Redaktion, "Die Sirenen heulen häufiger - warum?" [Online]. Available: https://www.hildesheimer-allgemeine.de/news/article/die-sirenen-heulen-haeufiger-warum.html

[31] Mardek Sirenen, "Bedeutung der aktuellen Sirenensignale in Deutschland." [Online]. Available: https://www.youtube.com/watch?v=thI5VJrK6ak&t=53s

[32] Bundesrepublik Deutschland, "Straßenverkehrs-Zulassungs-Ordnung (StVZO) Paragraph 55 - Einrichtungen für Schallzeichen." [Online]. Available: https://www.gesetze-im-internet.de/stvzo_2012/BJNR067910012.html

[33] Normenausschuss Feuerwehrwesen (FNFW) im DIN Normenausschuss Akustik, Lärmminderung und Schwingungstechnik (NALS) im DIN und VDI, "DIN 14610 - Akustische Warneinrichtungen für bevorrechtigte Wegebenutzer," 2009.

[34] R. Wagner, "Guide to Test Methods, Performance Requirements, and Installation Practices for Electronic Sirens Used on Law Enforcement Vehicles," 2000.

[35] A. I. Rubin and G. L. Howett, "Emergency Vehicle Warning Systems," 1981.

[36] U. Zölzer, *DAFX: Digital Audio Effects - Second Edition*, 2011.

[37] J. P. Bello, "Towards the automated analysis of simple polyphonic music: A knowledge-based approach," Ph.D. dissertation, 2003.

[38] E. Li and J. P. Bello, "Key-independent classification of harmonic change in musical audio," 2007.

[39] P. Vary and R. Martin, *Digital Speech Transmission: Enhancement, Coding And Error Concealment*, 2006.

[40] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," 2015. [Online]. Available: https://librosa.github.io/librosa/feature.html

[41] M. Slaney, "Auditory Toolbox - Version 2 - Technical Report," 1998. [Online]. Available: https://engineering.purdue.edu/~malcolm/interval/1998-010/

[42] "Is there a Matlab's buffer equivalent in numpy? ." [Online]. Available: https://stackoverflow.com/questions/38453249/is-there-a-matlabs-buffer-equivalent-in-numpy

[43] "Scikit Learn Documentation - 1.13.2. Univariate feature selection." [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

[44] "Chi-Square Statistic: How to Calculate It / Distribution." [Online]. Available: https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/chi-square/

[45] "Scikit Learn Documentation - 3.1. Cross-validation: evaluating estimator performance." [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html

[46] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller, "Fisher discriminant analysis with kernels," *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop*, 1999.

[47] "Scikit Learn Documentation - 5.3 Preprocessing data." [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html#

[48] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the cuidado project," Ircam, Analysis/Synthesis Team, 2004.

[49] D. Raya, V. Patidar, Y. Khatri, and S. B. Devatha, "Music Genre Classification." [Online]. Available: https://www.saibhaskardevatha.co.in/projects/dsp/index.html

[50] A. Schindler, "Music information retrieval: Part 2 - feature extraction." [Online]. Available: http://www.ifs.tuwien.ac.at/~schindler/lectures/MIR_Feature_Extraction.html

[51] "LibROSA Documentation - Spectrogram decomposition." [Online]. Available: https://librosa.github.io/librosa/0.5.1/decompose.html

[52] Z. Rafii and B. Pardo, "Music/voice separation using the similarity matrix," 2012.

[53] M. Verleysen, "Machine learning group - a short tutorial on feature selection," 2011. [Online]. Available: https://mlg.info.ucl.ac.be/Members/MichelVerleysen

[54] M. Zhao, R. H. M. Chan, P. Tang, T. W. S. Chow, and S. W. H. Wong, "Trace Ratio Linear Discriminant Analysis for Medical Diagnosis: A Case Study of Dementia," vol. 20, 2013.

[55] "How to choose the number of hidden layers and nodes in a feedforward neural network?" [Online]. Available: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

[56] "Scikit Learn Documentation - MLPClassifier." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html