Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ivan Turkalj

Entwicklung einer Softwareumgebung zur Optimierung
eines SVM-Klassifikationsverfahrens von Audiosignalen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ivan Turkalj

Entwicklung einer Softwareumgebung zur Optimierung eines SVM-Klassifikationsverfahrens von Audiosignalen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Wolfgang Fohl
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 14. April 2010

Ivan Turkalj

Thema der Bachelorarbeit

Entwicklung einer Softwareumgebung zur Optimierung eines SVM-Klassifikationsverfahrens von Audiosignalen

Stichworte

Skriptsprache GNU Octave, Support Vector Machines, Audioklassifikation, Maschinelles Lernen, Instrumentenklassifikation

Kurzzusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung einer Softwareumgebung zum SVM-Klassifikationsverfahren von Audiosignalen. Die Softwareumgebung wird für die Durchführung der ganzen Reihe von Experimenten zur Klassifikation von Gitarrenklängen eingesetzt. Experimentiert wird mit den extrahierten Merkmalsvektoren, welche die Daten beinhalten, die die Klangeigenschaften der Gitarren beschreiben. Dabei werden beispielhaft die Klänge der Gitarren von drei verschiedenen Gitarrenbauern klassifiziert und anschließend die Ergebnisse der Experimente diskutiert.

Title of the paper

Development of a software environment for the optimization of SVM classification method of audio signals

Keywords

High-level language GNU Octave, Support Vector Machines, Audio-Classification, Machine Learning, Musical Instrument Classification

Abstract

This bachelor thesis deals with the development of a software environment for the SVM classification of audio signals. This software environment will be used in a series of experiments on the classification of guitar sounds. The experiments will be made on vector matrices with extracted guitar tones which describe the features of guitar sounds. As examples will serve three guitars made by three different guitar manufacturers whose tones will be classified according to the results of the previously conducted experiment, as well as their subsequent analysis will be provided.

Danksagung

Ich möchte mich besonders bei Prof. Fohl und auch bei Prof. Meisel für die gute Betreuung dieser Bachelorarbeit bedanken.

Mein besonderer Dank gilt Wolfgang Stolze, der mich als Werkstudent eingestellt hat und immer ein offenes Ohr und Verständnis hatte für die Lage, in der ich mich als Student befand.

Für die Korrektur der Grammatik und Orthographie danke ich Frau Eichstädt-Pajcic.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Die Menschen werden ständig durch äußere Beeinflussung ihrer Sinne zu bewusster oder unbewusster Kommunikation mit ihrem Umfeld bewegt. Diese Bachelorarbeit beschäftigt sich im weitesten Sinne mit dem Hörsinn des Menschen. Es wird angestrebt, durch einen Computer die Fähigkeit des Menschen nachzuahmen, verschiedene Klänge zu unterscheiden und sie einer bestimmten Quelle zuzuordnen. Einen Klang einem Musikinstrument zuordnen zu können oder die Klassifizierung der Musikinstrumente durch einen Computer erfolgreich durchzuführen, erfordert vorab eine umfangreiche Analyse der Audiodaten. In dem Analyseprozess werden die Merkmale extrahiert, die einen Klang charakterisieren, um ihn zwischen vielen anderen Klängen immer eindeutig identifizieren zu können. In dieser Arbeit wird mit Audiodateien dreier verschiedener Akustikgitarren experimentiert.

1.1. Motivation

Die Klassifizierung der Audiosignale und die Erkenntnisse, die daraus folgen, könnten in vielen Gebieten eine praktische Anwendung finden. Sicherlich nicht das kleinste Anwendungsfeld ist die Musik- und Unterhaltungsindustrie. Ich kann mir vorstellen, dass ein Gitarrenbauer großes Interesse an einem Werkzeug haben könnte, das die Ergebnisse seiner Arbeit bzw. die Klangqualität des Instruments zuverlässig messen kann. Es ist bekannt, dass beim Bau akustischer Musikinstrumente nicht jedes Instrument gleich gut gelingt und dass, obwohl die Instrumente aus ähnlichen oder sogar gleichen Materialien aufgebaut werden, sie deutlich unterschiedliche Klangcharaktere haben können. Bei der Beurteilung des Klangs eines Instruments verlässt sich der Instrumentenbauer auf sein Gehör. Problematisch ist dabei, dass die Beurteilung der Qualität sehr von Elementen wie Erfahrung, momentane (seelische/körperliche) Verfassung und Musikgeschmack des Instrumentenbauers abhängt. Zuletzt wirkt sich diese Tätigkeit ermüdend auf den Hörsinn des Menschen aus und somit sind die Ergebnisse, die man am Anfang eines Abhörprozesses gewinnt, sicherlich objektiver als die, die man gewinnt, nachdem man schon längere Zeit mehrere Klangproben gehört hat.

In der 80-er Jahren hat die Ära der digitalen elektronischen Musikinstrumente wie Synthetiser, Sampler, Electronic-Drum u.s.w. angefangen. Bis heute gibt es den Wunsch, den Klang verschiedener Musikinstrumente zu synthetisieren und klanglich so nah wie möglich ans Original anklingen lassen. Die in dieser Arbeit entwickelte Softwareumgebung könnte auch als Hilfe bei der Beurteilung der Qualität eines synthetisierten Klangs eingesetzt werden.

Es gibt eine Reihe verschiedener wissenschaftlicher Arbeiten die sich mit der Klassifizierung der Musikinstrumente beschäftigen. Beispiele:

Markues und Moreno [MM99] haben die Klassifizierung acht unterschiedlicher Instrumente (Klarinette, Dudelsack, Flöte, Posaune, Oboe, Cembalo, Klavier und Orgel) durchgeführt.

Deng [DSC06] und eine Gruppe von Autoren haben die Klassifizierung der Instrumente durchgeführt, die nach vier Instrumententypen (Holzblas-, Blechblas-, Saiteninstrumente und Klavier) aufgeteilt sind.

Steelant [ST04] und eine Gruppe von Autoren haben sich mit der Klassifizierung der Schlaginstrumente (Bass-Drum, Snare-Drum, Tom-Tom, Hihat, und Becken) beschäftigt.

Dosenbach [Dos07] hat die Klänge der drei Akustikgitarren von drei verschiedenen Gitarrenbauern klassifiziert.

In allen vier Arbeiten ist unter anderem auch das Programm Support Vector Machines (SVM) als Klassifikationsalgorithmus eingesetzt worden und hat dabei gute bis sehr gute Ergebnisse geliefert.

Die am “Proc. of the 11th Int. Conference on Digital Audio Effects , Espoo, Finland, September 1-4, 2008“ vorgestellte Arbeit [DFM08] zeigt, dass auch eine Klassifizierung der Musikinstrumente gleicher Kategorie möglich ist. Meine Bachelorarbeit knüpft an Ergebnisse dieser Arbeit an und erweitert, verbessert, und optimiert dort vorgestellte Klassifikationsverfahren. Außerdem liefert meine Arbeit weitere Ergebnisse und Beweise, die die Brauchbarkeit und Richtigkeit des Extraktionsprozesses von Audiodaten untermauern, die Tauglichkeit des SVMs als Klassifikationsalgorithmus zeigen und das ganze Verfahren einen Schritt weiter in die Richtung praktischer Anwendung bringen sollen.

1.2. Herausforderungen und Ziele dieser Bachelorarbeit

Der Extraktionsprozess von Audiodaten ist sehr rechenintensiv und zeitaufwendig. Eines der Ziele dieser Arbeit ist die Entwicklung einer Softwareumgebung, die flexibles Testen und Experimentieren im Klassifikationsprozess der Instrumente erlaubt. Mit dieser Softwareumgebung wird eine Reihe von Experimenten durchgeführt werden. Es wird angestrebt, die Anzahl der Daten in den Extraktionsvektoren und die Anzahl der Audiosignale, die zu Klassifikation verwendet werden, deutlich zu reduzieren und dabei gute Klassifikationsergebnisse (um 80 % Erkennungsrate) beizubehalten.

Ein Merkmalsvektor eines Audiosignals besteht aus drei verschiedenen Gruppen von Merkmalen. Das sind: Zeitlicher Verlauf der Harmonischen (Harmonischen), Daten des nontonalen Spektrums und Mel Frequency Capstral Coeficients (MFCCs). Bei Betrachtung der Daten fällt auf, dass sich die Datengrößen einzelner Harmonischen und MFCCs in ihrer Größe sehr unterscheiden. Deshalb werden gleich im ersten Schritt Skalierungsfaktoren gesucht, die die einzelnen Harmonischen und MFCCs aneinander anpassen. Mit den optimierten Daten einerseits und Originaldaten anderseits wird in dieser Arbeit eine Reihe von Experimenten durchgeführt, die neue Erkenntnisse über die Klassifikation von Audiosignalen bringen sollen.

1.3. Gliederung und Aufbau

Im Kapitel 2 wird ein kurzer Überblick über die Begriffe, Methoden und Werkzeuge geliefert. Im Kapitel 3 werden die Anforderungen dokumentiert, die an die Softwareumgebung gestellt werden. Im Kapitel 4 werden diese Anforderungen umgesetzt. Dieses Kapitel beschäftigt sich mit der Implementierung und Realisation der Softwareumgebung.

In den Kapiteln 5, 6, 7 und 8 werden die Experimente beschrieben und die Ergebnisse diskutiert, die als Ziel haben, die Menge der Daten in den Merkmalsvektoren zu reduzieren. Im Kapitel 9 werden die Ergebnisse diskutiert und die Experimente beschrieben, die als Ziel haben, die Menge von Audiosignalen zu minimieren. Im Kapitel 10 wird mit den verschiedenen SVM-Kernel-Typen und den verschiedenen Konfigurationen des SVM-Kernels experimentiert. In den Anhängen A und B sind zwei Tutorien zu finden. Dort wird die Installation der Software erklärt und der Umgang mit den Konfigurationsdateien anhand eines Beispiel-Projekts verdeutlicht.

2. Grundlagen

In diesem Kapitel wird ein kurzer Überblick gegeben über die Begriffe, Methoden und Werkzeuge, die in dieser Arbeit benutzt werden.

2.1. Klang der Gitarre

Ein Klang kann allgemein definiert werden als ein Schallsignal, dem das Gehör des Menschen eine Tonhöhe zuordnen kann. Ein Klang (vereinfacht, physikalisch gesehen) besteht aus mehreren überlagerten sinusförmigen Schwingungen (Tönen), die sich in ihrer Frequenz und Amplitude (Lautstärke) unterscheiden. Dabei unterscheidet man zwischen dem Grundton (oder Grundfrequenz) und den Obertönen. Die Schwingung mit der niedrigsten Frequenz ist gleichzeitig auch Grundton des Klangs. Die Obertöne sind die Schwingungen, die das ganzzahlige Vielfache der Grundfrequenz haben. In dieser Arbeit werden die Begriffe “Obertöne“ und “Harmonische“ synonym verwendet.

Dagegen ist der Klang einer Gitarre noch viel komplexer aufgebaut. Der Klang der Gitarre besteht nicht nur aus den Harmonischen, sondern es sind auch Schwingungen vorhanden, die sich zwischen den Peaks der Harmonischen befinden. Hier handelt sich um die unharmonischen sowie geräuschhaften Anteile des Klangs. Man spricht über den sogenannten nontonalen Teil des Klangs. Treten diese Schwingungen periodisch auf, spricht man in diesem Zusammenhang von unharmonischen Obertönen [Mey85].

Der Klang einer Gitarre wird auch vom Verhältnis der Stärke der einzelnen Harmonischen zueinander, beeinflusst. Die Stärke der Harmonischen zueinander wird vollständig im Extraktionsmerkmal “Zeitreihe der Harmonischen“ abgespeichert und wird (neben den zwei weiteren Merkmalen, nämlich nontonales Spektrum und MFCCs) in den Merkmalsvektor aufgenommen. Dieses Verhältnis hängt von zwei Faktoren ab. Einerseits ist es die Bauart und das Material, aus dem die Gitarre gebaut wurde, anderseits ist das die Spielweise des Spielers. Abhängig davon, auf welcher Stelle der Spieler die Saite der Gitarre anschlägt, verändert sich auch der Klang. Für die Klassifikationsverfahren ist wichtig, dass der Einfluss des Spielers auf den Klang so klein wie möglich gehalten wird, weil nicht der Einfluss des Spielers auf die Klänge der Gitarren gemessen werden soll,

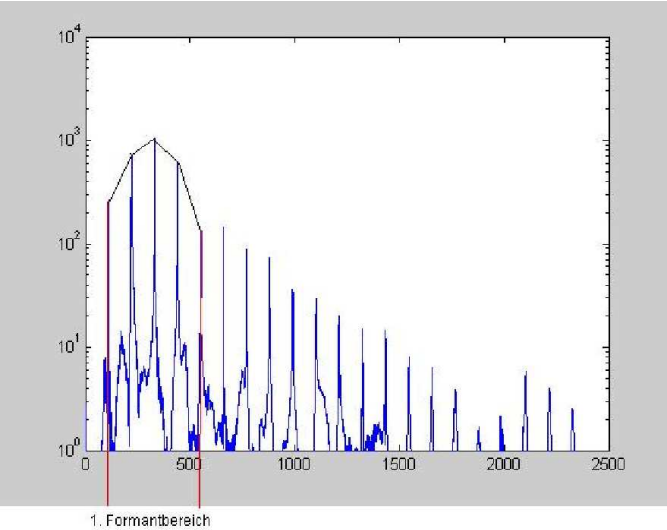
sondern die Gitarenklänge klassifiziert werden sollen. Deshalb sind die Gitarrenklänge, die vorliegen, nur mit bestimmten Klangvorgaben aufgenommen worden. Die Klangvorgaben sind “sonor“, “spitz“ und “warm“. Die Klangfarbe “warm“ haben die Klänge, die in der Spielposition zwischen dem Schallloch und dem Hals der Gitarre erzeugt werden. Die Klänge, die in der Nähe des Gitarrenstegs erzeugt werden, bekommen die Klangfarbe “spitz“. Die Farbe der Klänge, die in der Spielposition zwischen den beiden o.a. Positionen erzeugt werden, wird als “sonor“ bezeichnet.

Eine wichtige Rolle spielt auch das Resonanzverhalten der Gitarre, das von der Hohlraumresonanz, den Eigenresonanzen der Gitarrendecke und des Gitarrenbodens abhängt. Von den Resonanzeigenschaften der Gitarre hängen die Intensität und die zeitlichen Verläufe der Harmonischen ab, deren Ein- und Ausschwingzeiten auch die Klangfarbe der Gitarre beeinflussen.

Außerdem haben Lage und Ausprägung der Formanten einen wesentlichen Einfluß auf die Klangfarbe einer Gitarre. Formanten sind Frequenzbereiche, die besonders ausgeprägt sind und im Wesentlichen von der Bauform bzw. vom Resonanzverhalten der Gitarre abhängen. Gitarren weisen in der Regel mehr als einen Formantbereich auf.

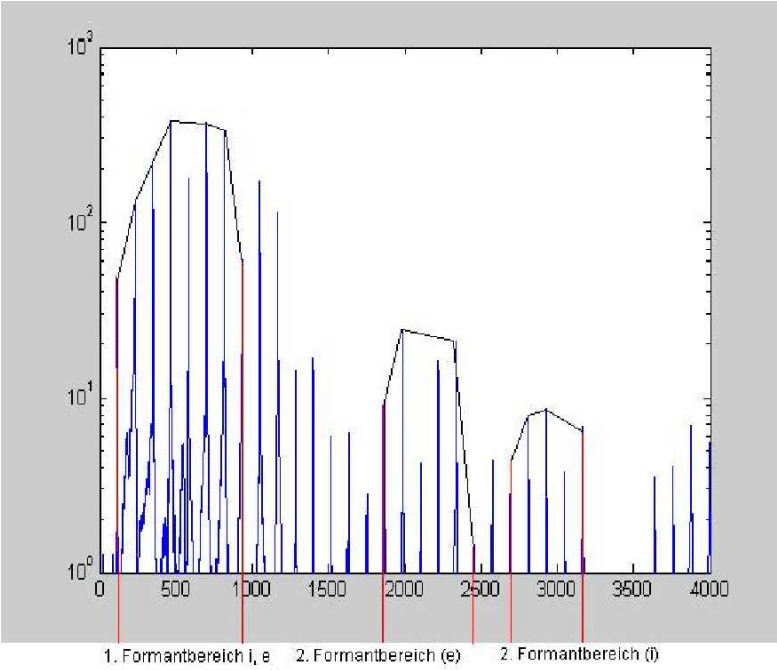
Die Anhebung bestimmter Frequenzbereiche wird im Instrumentenbau durch die Formgebung der Instrumente gezielt eingesetzt, um dem Instrument einen bestimmten Klangcharakter (Klangfarbe) zu verleihen. Fällt eine Oberwelle oder eine Frequenz eines Rauschanteils in einen dieser Frequenzbereiche, so wird dieser Anteil besonders verstärkt.

Abbildung 2.1.: Formantbereich eines Klangs mit der Klangvorgabe “warm“



Die Abbildungen 2.1 und 2.2 (übernommen aus [Voi07]) stellen die Klangspektren zweier Klänge dar. Gezeigt werden die Formantbereiche, die zu den Spielvorgaben “warm“ und “spitz“ gehören. Aus der Abbildung 2.1 kann man erkennen, dass der Formantbereich, der charakteristisch für die Klangfarbe “warm“ ist, zwischen 200 Hz und 600 Hz liegt. Dagegen sind die Formantbereiche der Klänge, die zur Klangfarben-Kategorie “spitz“ gehören, in den Bereichen um 2200 Hz bis 2600 Hz und 3000 Hz bis 3500 Hz zu finden. (Siehe Abbildung 2.2.)

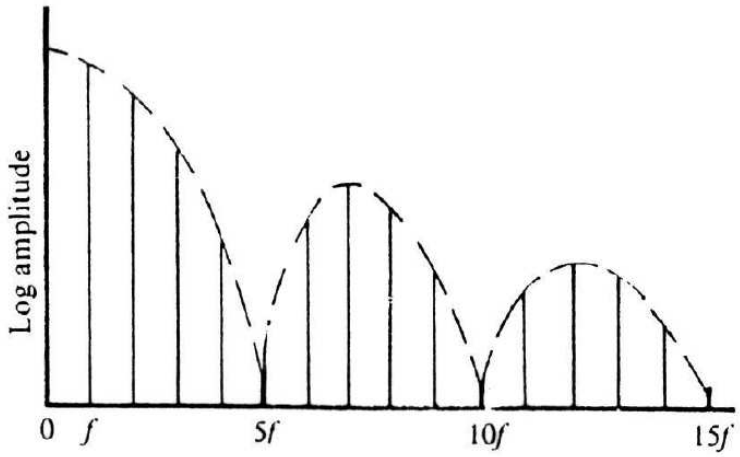
Abbildung 2.2.: Formantbereich eines Klangs mit der Klangvorgabe “spitz“



Die Lage der Formanten und der Reichtum an Obertönen hängen von der Position ab, an der die Saite angezupft wird. Somit beeinflusst der Spieler die Klangfarbe einer Gitarre im Wesentlichen durch die Wahl der Anschlagsposition. Die Gitarrenklänge, die die Klangfarbe “spitz“ haben, sind reicher an Obertönen als diejenigen, die die Klangfarbe “warm“ haben. Als Beispiel kann man zwei Extermfälle nehmen. Das ist der Fall, wenn die Saite über dem 12. Bund, angezupft wird, und auch, wenn das etwa 1 cm vom Steg entfernt geschieht. Im ersten Fall wird jeder zweite Partialton unterdrückt und im zweiten Fall wird (ausgegangen von 64 cm Saitenlänge) jeder 64. Partialton unterdrückt. An der Stelle, an der die Saite angezupft wird, kann kein Schwingungsknoten entstehen. Aus diesem Grund werden die Frequenzen unterdrückt, deren Schwingungen an dieser Stelle einen Knoten haben. Weiteren Details sind im Artikel von Traube und Smith [TS01] zu finden. Aus diesem Artikel wurde die Abbildung 2.3 übernommen, die das Klangspek-

trum einer Saite zeigt, die an der Position 1/5 der Saitenlänge angezupft wurde. Es ist zu erkennen, dass die Frequenzen, die das Vielfache 5 haben, unterdrückt werden.

Abbildung 2.3.: Klangspektrum einer Saite, die an der Position 1/5 der Saitenlänge angezupft wurde

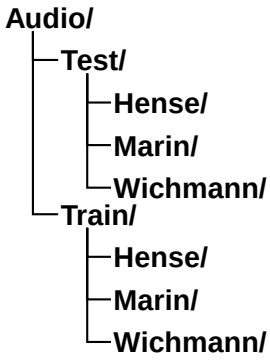


2.2. Verwendete Audiodateien

Auf den Gitarren (Hense, Marin und Wichmann) sind die einzelnen Noten von vier Musikern eingespielt und im Schallmessraum der Hochschule in 24 Bit Auflösung, 44100 Hz Sampling-Rate aufgenommen worden. Dabei haben sich die Musiker an Vorgaben gehalten, und zwar, dass unter der Klangvorgabe “sonor“, “spitz“ und “warm“ nur bestimmte Noten auf vorgegebenen Saiten und Lagen (auf den Bünden 1, 4, 5, 9 und 10) gespielt werden. Jede Note ist viermal aufgenommen worden. So ist gewährleistet, dass ein großes Spektrum an Klängen entstanden ist und dass man eine ausreichend große Trainings- und Testmenge bilden kann.

Die Audiosignale sind mit dem Kondensator-Mikrophon AKG C3000 B aufgenommen und mit MOTU 828mkII Audio-Interface digitalisiert worden. Um negative Einflüsse von “near-field-effects“ zu kompensieren, wurde der im Mikrophon eingebaute Low-Pass-Filter (20dB/Dekade, 500 Hz Cutoff-Frequenz) eingeschaltet. Die aufgenommenen Audiodateien haben eine feste Länge von 2 Sekunden. Der Startpunkt der Audiodateien ist festgelegt worden auf 0,1 Sekunde, bevor die maximale Amplitude des Audiosignals erreicht wird.

Abbildung 2.4.: Verzeichnisstruktur und Aufteilung der Audiodateien



Die Audiodateien sind in einer Verzeichnisstruktur untergebracht worden, die in der Abbildung 2.4 abgebildet ist. Im “Train“ Verzeichnis befinden sich 978 Trainings- und im “Test“ Verzeichnis 972 Testdateien. Das ergibt 1950 Audiodateien, die zur Klassifikation benutzt werden. Dabei wurden die Dateien, die zu einer bestimmten Gitarre gehören, jeweils in einem eigenen Unterverzeichnis abgespeichert.

2.3. Merkmale und Merkmalsextraktion des Klangs

Wie bereits im Abschnitt 1.2 erwähnt, werden die Merkmale der Audiosignale im Extaktionsprozess in Merkmalsvektoren abgespeichert. Mit der Auswahl der Merkmale, die zur Klassifikation der klassischen Musikinstrumente geeignet sind, haben sich die Autoren in verschiedenen Arbeiten beschäftigt. Dabei ging es in vielen dieser Arbeiten um die Aufgabe, ein bestimmtes Instrument aus einer Gruppe von Instrumenten zu identifizieren oder ein Instrument zu einer Instrumenten-Gruppe zuzuordnen.

Im Klassifikationsprozess, der in dieser Arbeit verwendet wird, sind drei verschiedene Merkmale des Klangs zum Einsatz gekommen. Ausgewählt sind die Merkmale, die sich besonders gut zur Klassifizierung der klassischen Instrumente eignen. Das sind: zeitlicher Verlauf der Harmonischen, Daten des nontonalen Spektrums und Mel Frequency Capstral Coeficients (MFCCs).

In den Arbeiten von Marques und Moreno [MM99] und Deng u.a. [DSC06] wurde mit verschiedenen Merkmalen zur Klassifikation der Musikinstrumente experimentiert. In beiden Fällen haben die Mel Frequency Cepstral Coefficients (MFCCs) Merkmale beste Klassifikationsergebnisse geliefert.

Fragoulis u.a. [FDCC06] beschreiben eine weitere Methode, die die Klänge einer Gitarre und eines Pianos nach nontonalen Merkmalen klassifiziert. Die Autoren haben festgestellt, dass die Klänge ihren typischen Charakter verlieren, wenn die nontonalen Frequenzkomponenten aus dem Klangspektrum entfernt werden. Als Folgerung daraus haben sie die These erstellt, dass das nontonale Spektrum ein wichtiger Teil des Klangspektrums ist und einen wesentlichen Einfluss auf den Klang eines Instruments hat.

Das dritte Merkmal, das in den Merkmalsvektor aufgenommen wurde, ist der zeitliche Verlauf der Harmonischen. Das Zusammenspiel zwischen dem Resonanzverhalten der Gitarre und den Formanten hat großen Einfluss auf die zeitlichen Verläufe der Harmonischen, deren Ein- und Ausschwingzeiten die Klangfarbe der Gitarre beeinflussen.

Der von Dossenbach [Dos07] vorgestellte Merkmalsextraktion-Prozess verwendet folgende Algorithmen und Funktionen, die zur Gewinnung der Merkmale eingesetzt wurden.

Die MFCCs wurden berechnet mit der Matlab-Funktion “mfcc.m“, die im Rahmen der Entwicklung der Apple Auditory Toolbox entwickelt wurde. Die Auditory Toolbox wird von Malcolm Slaney [Sla98] weitergeführt. Es wurden erste 10 MFCCs berechnet. Die Funktion ist mit einer Framerate von 25 Hz (40 mS) aufgerufen worden. Extrahiert wurden 50 Werte pro MFCC. Das ergab insgesamt die Größe von 500 Werten, die in den Merkmalsvektor aufgenommen wurden.

Zur Berechnung der Merkmale, die zum nontonalen Spektrum gehören, wurde der in MATLAB entwickelte Algorithmus von Fragoulis u.a. [FDCC06] verwendet. Extrahiert wurden erste 15 Peaks des Spektrums.

Die Berechnung der Merkmale, die zum zeitlichen Verlauf der Harmonischen gehören, wurde mit der Octave-Funktion durchgeführt, die von Julia Voigt [Voi07] entwickelt wurde. Berechnet wurden die ersten 16 Harmonischen. Das Signal wurde dazu in Blöcke von der Länge 4096 Samples unterteilt, was einer Framerate von ungefähr 10 Hz entspricht. Auf diese Weise gewinnt man 40 Werte für alle einzelnen Harmonischen. Insgesamt sind das 640 Werte, die in den Merkmalsvektor aufgenommen wurden.

2.4. Verwendete Software

2.4.1. GNU Octave

Die selbst entwickelte Software wurde komplett mit der Octave Skriptsprache, in der Version 3.0.1 geschrieben. Octave und Matlab sind größtenteils kompatibel, wobei Mat-

lab aber einen größeren Funktionsumfang besitzt. GNU Octave steht unter GPL Lizenz und ist für viele Computersysteme kostenlos erhältlich. Octave ist für die Betriebssysteme Linux, Windows und Mac OS X verfügbar. Außerdem kann man zum Testzwecke, ohne das Programm installieren zu müssen eine Knoppix-Version mit Octave-Inhalt[1] benutzen. Für Octave gibt es viele Tool-Boxen, die Funktionen für spezielle Probleme zur Verfügung stellen. Als Beispiel möchte ich Octave-Forge[2] und Octaviz[3] nennen. Octave-Forge ist eine Sammlung von extra Paketen und Octaviz ist ein Visualisierungs-System für Octave.

2.4.2. Support Vector Machines (SVMs)

Frank Rosenblatt hat die Idee, Objekte durch eine Hyperebene zu trennen, im Jahr 1958 veröffentlicht. Vladimir N. Vapnik und Aleksei Chervonenkis haben die Idee der Support Vector Machines (SVMs) in ihrer Arbeit “Theory of Pattern Recognition“ im Jahr 1974 vorgestellt. Die SVMs sind in den Jahren danach maßgeblich von Vladimir N. Vapnik [Vap98] [Vap00] entwickelt worden.

Die SVMs sind ein statistisches Lernverfahren, das zur Klassifizierung der Daten eingesetzt werden kann. Zu klassifizierende Daten werden auf die Art und Weise in Klassen unterteilt, dass entlang der Klassengrenzen ein möglichst breiter Bereich bleibt, der keine Daten beinhaltet. Oder anders ausgedrückt, SVMs suchen eine optimale Trennebene (die Hyperebene) zwischen den zu trennenden Klassen. Dabei ist die optimale Hyperebene die, die den breitesten Rand besitzt, bzw. die größte Entfernung zwischen den zu trennenden Klassen aufweist. Details über das Klassifikationsverfahren mit SVMs findet man in den Büchern [SS02] und [SC08]. Den SVMs ist in der Mitte der 1990-er Jahre der Durchbruch gelungen. Ab diesem Zeitpunkt wurden viele Modifikationen und Weiterentwicklungen der SVMs veröffentlicht. Eine Übersicht über die SVM-Software und die SVM-Literatur findet man hier[4].

In dieser Arbeit ist die SVM^{light} Implementierung eingesetzt worden, die von Thorsten Joachims [Joa99] entwickelt wurde. SVM^{light} ist eine sehr oft eingesetzte SVM-Implementierung, die zur Klassifizierung und Regression eingesetzt werden kann. SVM^{light} weist einige positive Merkmale und Eigenschaften auf, die den Einsatz dieser Implementierung sehr vorteilhaft machen. Diese Implementierung ist schnell und kann

[1] Download unter: `http://pareto.uab.es/mcreel/Econometrics/`
[2] Download unter: `http://octave.sourceforge.net/`
[3] Download unter: `http://octaviz.sourceforge.net/`
[4] `http://www.support-vector-machines.org/SVM_soft.html`

Berechnungen mit sehr großen Datenvektoren durchführen. Außerdem werden Debugg-Informationen ausgegeben, die Auskunft über der Verlauf der Trainingsphase geben. Unter anderen sind das die Länge der Trainingsphase, ausgedrückt als Anzahl benötigter Iterationen zur Berechnung der optimalen Hyperebene und die Länge der Trainingsphase in Sekunden. Dabei wird auch die Anzahl der berechneten Supportvektoren und die Anzahl der Vektoren angegeben, die sich auf der richtigen bzw. auf der falschen Seite des Margins befinden. Diese Eigenschaften sind nützlich bei der Ermittlung und Optimierung von Feineinstellungen in der Trainingsphase.

Es werden verschiedene Kernel-Typen von SVM^{light} zur Verfügung gestellt.
- Linear-Kernel - $(a * b)$
- Polynomial-Kernel - $(s(a * b) + c)^d$
- Radial-Basis-Function-Kernel - $exp(-gamma||a - b||^2)$
- Sigmoid-Kernel - $tanh(s(a * b) + c)$

Die SVM^{light} Software kann man auf der Webseite des Autors[5] besorgen. Die Software kann als “Source Code“ oder “Binaries“ heruntergeladen werden und ist für die Betriebssysteme Solaris, Windows und Linux verfügbar. Außerdem ist die Software auch unter Cygwin[6] lauffähig. Wenn man sich für die “Source Code“ Version des Programms entscheidet, soll die Software zunächst kompiliert werden. Als Ergebnis des Kompilierungsprozesses werden zwei Programme erzeugt. Das sind `svm_learn` und `svm_classify`, die in der Trainings- bzw. in der Test-Phase verwendet werden. Einzelne Compilierungs-Befehle sind auf der Webseite des Autors beschrieben. Außerdem sind dort die Beispiele und Tutorien zur SVM zu finden.

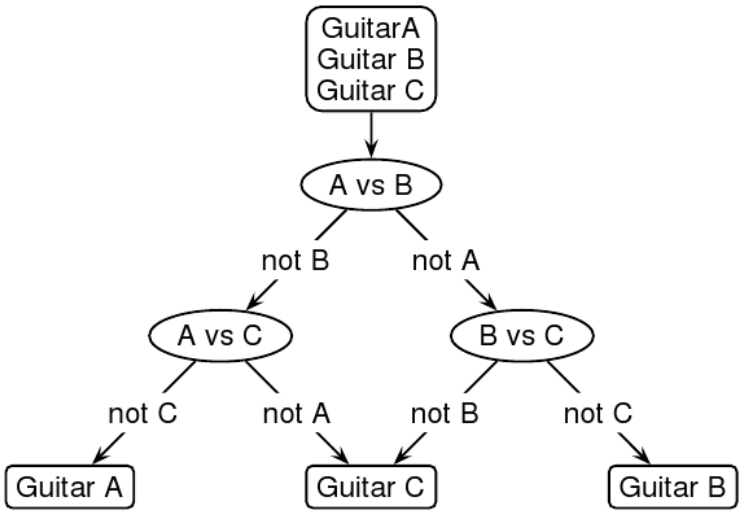
Support Vector Machines können nur zur Lösung von Zweiklassen-Problemen eingesetzt werden. Das Klassifikationsverfahren, das in dieser Arbeit verwendet wurde, soll aber zur Lösung von Mehrklassen-Problemen geeignet sein. Um diesen Anforderungen gerecht zu werden, kann man sich zweier verschiedener Verfahren bedienen, die in solchen Fällen häufig verwendet werden. Das sind One-Versus-One- und One-Versus-Rest-Verfahren.

Das One-Versus-One-Verfahren wird jeweils paarweise für alle Klassen durchgeführt und zwar so, dass jede Klasse gegen jede der anderen Klassen trainiert wird. Für N Klassen, die man klassifizieren möchte, werden N*(N-1)/2 SVMs benötigt, um entscheiden zu können, zur welcher Klasse ein unbekanntes Datum gehört. Bei diesem Verfahren wurde ein azyklischer Graph (Directed Acyclic Graph [DAG]) konstruiert, in dem die Ab-

[5] Download unter: `http://svmlight.joachims.org/`
[6] Das ist eine UNIX-artige Umgebung unter Windows

läufe für die Klassifikation festgelegt wurden. Siehe die Abbildung 2.5, die aus [DFM08] übernommen wurde.

Abbildung 2.5.: DAG für das One-Versus-One-Klassifikationsverfahren



In dem One-Versus-Rest-Verfahren wird jede Klasse gegen den Rest trainiert. Es werden N SVMs für N Klassen benötigt. Obwohl der Rechenaufwand in diesem Verfahren niedriger ist als im One-Versus-One-Verfahren zeigt sich als nachteilig, dass im One-Versus-Rest-Verfahren die Verwendung von modifizierten SVM-Kerneln (nicht-Linear-Kernel) erforderlich ist. Das erhöht den Rechenaufwand und macht die Konfiguration des Verfahrens komplizierter. Aus diesen Gründen ist die Benutzung des One-Versus-One-Verfahrens vorteilhafter als das One-Versus-Rest-Verfahren.

Als SVM-Interface wurde das von Anton Schwaighofer entwickelte Matlab-Interface verwendet (Siehe Abschnitt A.2.3).

3. Software-Anforderungen

In dem Klassifikationsprozess sollen die Audiodaten in mehreren, nacheinander folgenden Phasen verarbeitet werden. Die Audiodaten liegen im wav-Format vor. Die Dateinamen sind nach einer bestimmter Nomenklatur vergeben worden und sind gut geeignet für die Kodierung der Eigenschaften von Audiodateien, weil die Informationen über alle relevanten Eigenschaften von Audiodateien bereits in den Namen enthalten sind.

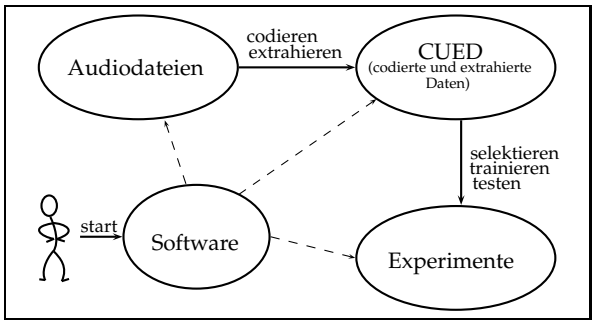
In einer Merkmalsextraktions-Phase sollen wichtige Merkmale extrahiert und in geeigneter Form abgespeichert werden. Außerdem sollen die Audiodaten, anhand Ihrer Dateinamen, einer Kodierung unterzogen werden und ebenfalls in einer Form abgespeichert werden, die problemfrei Zugriff und Nutzung in späteren Phasen des Klassifikationsprozesses ermöglicht.

Eines der Ziele dieser Arbeit ist, die Menge der Daten zu reduzieren, die im Klassifikationsprozess benutzt werden, und zwar bei gleich bleibender Erkennungsrate. Deshalb ist es erforderlich, eine Softwareumgebung zu entwickeln, die eine flexible Selektion der Daten ermöglicht. Die selektierten Daten sollen anschließend in ein Format gebracht werden, das den Anforderungen der SVM Trainings- und Testphase gerecht wird.

Die Softwareumgebung soll eine flexible Konfiguration und Anpassung des Klassifikationsprozesses auf eigene Bedürfnisse ermöglichen. Es gibt verschiedene Szenarien zum Ablauf des Klassifikationsprozesses, der durch die entwickelte Software unterstützen werden soll. So sollen die Audiodaten, die im Rahmen dieser Bachelorarbeit benutzt werden, die Merkmalsextraktion und Kodierung nur einmal durchlaufen, weil alle Experimente nur mit Audiodateien, die bereits vorliegen, durchgeführt werden. Man wird für einzelne Versuche aus diesen Daten-Obermengen einen bestimmten (gewollten) Teil der Daten selektieren und damit weiter experimentieren. Ein weiteres Szenario ist, dass z.B. weitere Audiodateien oder weitere Gitarren-Verzeichnisse zum Audiodaten-Container hinzugefügt werden. Das erfordert dann eine Merkmalsextraktion und Kodierung von neuen Daten.

Die Abbildung 3.1 zeigt den Aufbau des Klassifikationsprozesses. Dabei stellen die Knoten dieses Graphes die Stammverzeichnisse dar, in denen die Daten der einzelnen Verarbeitungsphasen abgespeichert werden. Durch die Kanten des Graphes werden die

Abbildung 3.1.: Einzelne Speicherorte und Phasen des Klassifikationsprozesses



einzelnen Phasen des Klassifikationsprozesses dargestellt. Der genaue Ablauf des Klassifikationsprozesses wird in den Konfigurationsdateien festgehalten, die in dem Stammverzeichnis “Software“ liegen. Die fünf einzelnen Phasen (extrahieren, kodieren, selektieren, trainieren und testen) werden in zwei Gruppen zusammengefasst. Die erste Gruppe bilden die Kodierungs- und Extatktions-Phase, deren Ablauf durch eine Konfigurationsdatei festgelegt wird. Zu der zweiten Gruppe werden die restlichen drei Phasen (selektieren, trainieren und testen) zugeordnet. Diese Aufteilung in zwei Gruppen und die Möglichkeit, beide Gruppen-Phasen getrennt zu konfigurieren und die Software getrennt starten zu können, erhöht die Flexibilität und Leistungsfähigkeit des Klassifikationsprozesses. Die Kodierungs- und Extraktions-Phase werden im weiteren Text CE-Phase genannt. Das Akronym CE steht für “codieren“ und “extrahieren“. Als STT-Phase werden die drei einzelnen Phasen Selektions-, Trainings- und Test-Phase zusammengefasst.

3.1. Kodierungs- und Merkmalsextraktionsphase (CE-Phase)

Die Merkmalsextraktion ist zeitaufwändig und rechenintensiv und soll nur durchgeführt werden, wenn man neue Audiodateien extrahieren möchte oder wenn die bestehenden Dateien mit veränderten Konfigurationsparametern noch einmal extrahiert werden. Die Daten werden im GNU Octave eigenen Matrix-Format abgespeichert. Dieses Matrix-Format lässt sich als Datei abspeichern und bei Bedarf direkt als Octave eigene Matrix wieder laden.

In der Abbildung 3.2 ist die Main-Verzeichnisstruktur des Kassifikationsprozesses dargestellt. Diese Verzeichnisstruktur, bzw. die Namen der Verzeichnisse, soll man in einer Konfigurationsdatei frei vergeben können.

Abbildung 3.2.: Stammverzeichnisse des Kassifikationsprozesses

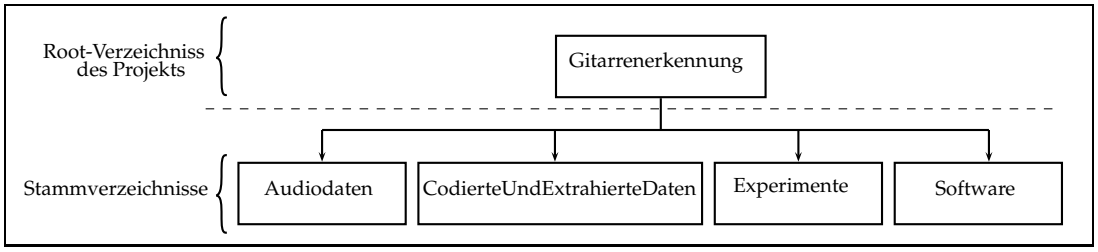


Abbildung 3.3.: Inhalt des Stammverzeichnisses “Audiodaten“

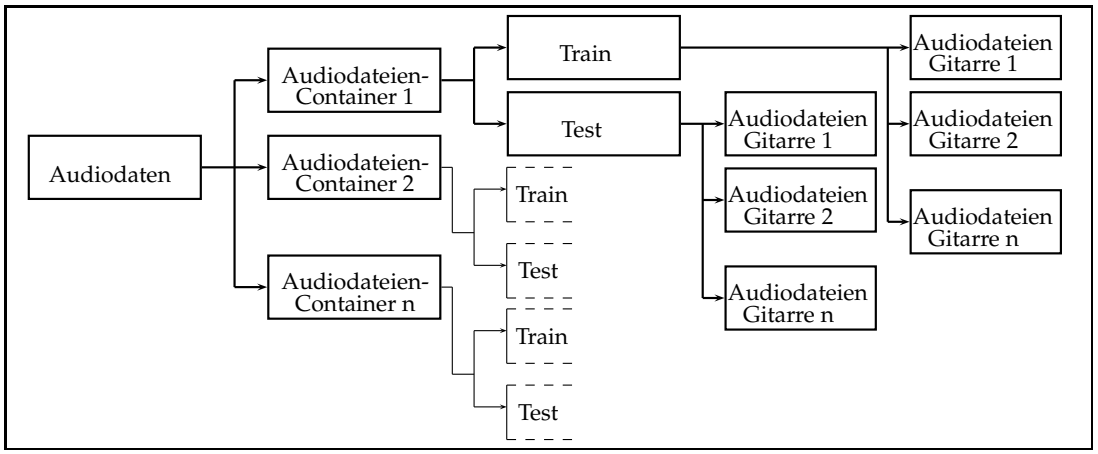


Abbildung 3.4.: Inhalt des Stammverzeichnisses “CodierteUndExtrahierteDaten“

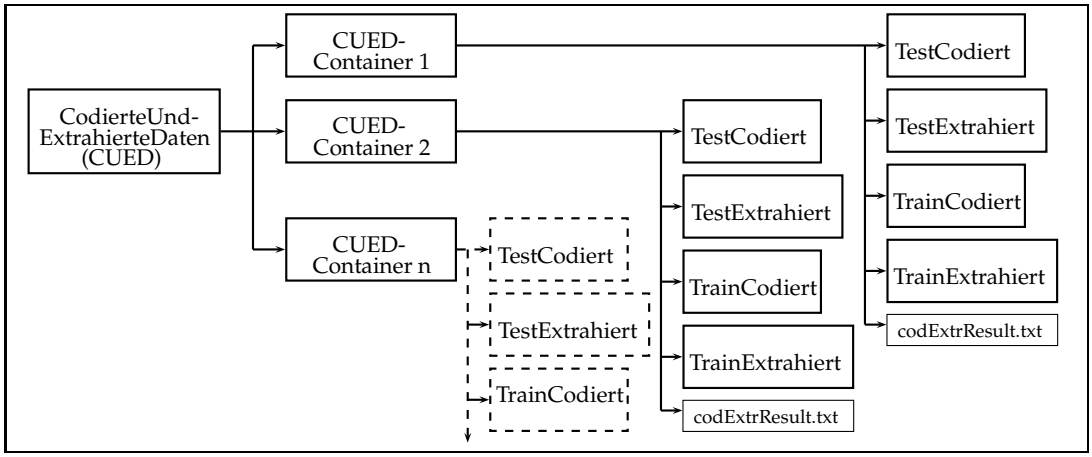
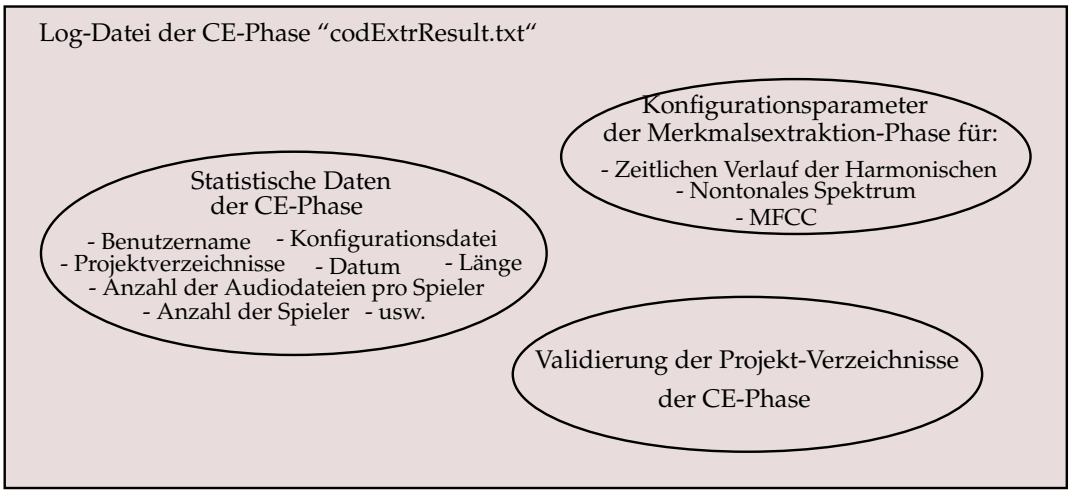


Abbildung 3.5.: Inhalt der Log-Datei “codExtrResult.txt“



Das Stammverzeichnis “Audiodaten“ beinhaltet ein oder mehrere “Audiodateien-Container“ Verzeichnisse, die wiederum mehrere Unterverzeichnisse beinhalten. (Siehe Abbildung 3.3.) Die Namen der Unterverzeichnisse “Test“ und “Train“ sind fest vergeben und sollen nicht geändert werden, weil sie kodiert werden und als Unterscheidungsmerkmal zwischen den Trainings- und Test-Audiodateien dienen. Die beiden Verzeichnisse (“Test“ und “Train“) können zwei oder mehrere Gitarren-Unterverzeichnisse haben. Die Namen der Gitarren-Verzeichnisse werden ebenfalls kodiert, um die Audiodateien gezielt selektieren zu können. Die Namen der Gitarren-Verzeichnisse kann man frei vergeben. Zur Vergebung der Namen können alle alphanumerischen Zeichen, bis auf die Sonderzeichen, verwendet werden. Außerdem werden die Namen der Gitarren-Verzeichnisse eingelesen und zur Bildung der Namen von den Octave-Matrizes-Dateien in den “CUED-Container“ Verzeichnissen verwendet. (Siehe Abbildung 3.4.)

Das Stammverzeichnis “CodierteUndExtrahierteDaten“ beinhaltet die Verzeichnisstruktur wie in der Abbildung 3.4 dargestellt ist. Alle “CUED-Container“ Verzeichnisse beinhalten die kodierten und extrahierten Daten, die als Octave-Matrix (.codiert- und .extrahiert-Dateien) in der abgebildeten Verzeichnisstruktur abgespeichert werden. Außerdem wird in jedem “CUED-Container“ die Log-Datei (“codExtrResult.txt“) erzeugt. In der Log-Datei werden die wichtigen Kenndaten und Ereignisse über den Verlauf der Extraktions- und Kodierungs-Phase eingetragen. (Siehe Abbildung 3.5 und das Listing D.3.)

3.1.1. Matrix für die extrahierten Audiodateien

Diese Matrix hat die Form wie in Abbildung 3.1 dargestellt. Jeweils eine Audiodatei soll eine Reihe der Matrix bilden. Jede Matrixreihe soll als Default-Wert 1155 Spaltenwerte haben. Eine Matrix kann “1“ bis “n“ Audiodateivektoren als Matrixreihen haben. Das ist abhängig davon, wie viele Audiodateien in den Gitarren-Verzeichnissen vorliegen. Die Anzahl von Spalten in der Matrix hängt ab von der Anzahl der MFCCs, Harmonischen und den Daten des nontonalen Spektrums, bzw. von der Anzahl der Werte, die zu der Merkmalsart gehören. Als Default-Werte sind 50 Werte pro MFCC und 40 Werte pro Harmonischen gesetzt. Dazu kommen noch insgesamt 15 Werte des nontonalen Spektrums.

Die Matrizes werden in die Verzeichnisse “TrainExtrahiert“ und “TestExtrahiert“ abgespeichert, die Unterverzeichnisse des “CUED-Container“ sind. (Siehe Abbildung 3.4.) Jedes Gitarren-Verzeichnis, das im “Audiodateien-Container“ eingelesen wird, wird in einer eigenen Matrix abgespeichert. Dabei werden die Namen der Matrix-Dateien nach den folgenden Regeln vergeben:

```
<test oder train>_<Nummer des Gitarren-Verzeichnisses>_
<Name des Gitarren-Verzeichnisses>.extrahiert
```

Dabei ist zu beachten, dass die Gitarren-Verzeichnisse in lexikalischer Reihenfolge (so wie die in den Audiodateien-Containern vorliegenden) eingelesen und extrahiert werden. Nach der lexikalischen Reihenfolge werden die Nummern der Gitarren-Verzeichnisse vergeben. (Siehe Listing 3.1.)

Listing 3.1: Nomenklatur der Matrizes für für die extrahierten Audiodateien

```
test_1_Hense.extrahiert
test_2_Marin.extrahiert
train_1_Hense.extrahiert
train_2_Marin.extrahiert
```

3.1.2. Nomenklatur der Audiodateien

Am Beispiel drei zufällig ausgewählter Dateien kann man aus dem Listing 3.2 und der Tabelle 3.2 Informationen über den Aufbau der Nomenklatur entnehmen, die bei der Vergabe des Namens von Audiodateien benutzt wurde. In der oberen Zeile der Tabelle sind die Kategorien aufgelistet, nach denen die Dateien kodiert werden.

Es soll die Entscheidung getroffen werden, ob man alle Kategorien kodiert oder es schon reicht, eine Teilmenge zu kodieren, um eine Datei vollständig identifizieren zu können. In

Tabelle 3.1.: Aufbau der Matrix mit extrahierten Audiodateien

Merkmalsart	Nontonal	MFCCs	Harmonischen
Spaltenanzahl	15	500	640
Spaltenauswahl	[1, 2 ... 15]	[M1, M2 ... M10]*	[T1, T2 ... T16]**
Audiodatei 1	N1, N2, ... , N15	M1, M2, ... , M10	T1, T2, ... , T16
Audiodatei 2	N1, N2, ... , N15	M1, M2, ... , M10	T1, T2, ... , T16
...	...	...	...
Audiodatei n	N1, N2, ... , N15	M1, M2, ... , M10	T1, T2, ... , T16

* (Es werden jeweils 50 Spaltenwerte pro MFCC ausgewählt (für M1 bis M10))

** (Es werden jeweils 40 Spaltenwerte pro Harmonischen ausgewählt (für T1 bis T16))

diesem Zusammenhang kann man feststellen, dass eine Kodierung der Töne nicht notwendig ist, weil sich diese Information aus der Kombination Saite/Bund ermitteln lässt. Um Redundanzen zu vermeiden, wird die Kategorie “Ton“ nicht kodiert. Die Kategorie “Dateinummer“ wird ebenfalls nicht kodiert, weil sich diese Information für den Klassifikationsprozess als nicht relevant erwiesen hat.

Listing 3.2: Nomenklatur der Audiodateien

```
hoffmannL_hense_sonor_S1-I_F5-1.wav
obaL_marin_spitz_S5-X_G4-1.wav
hoffmanL_wichmann_sonor_S1-VI_Bb5-3.wav
```

Tabelle 3.2.: Nomenklatur der Audiodateien

Beispiel Nr.	Spieler	Kanal	Gitarre	Spielart	Saite	Bund	Ton	Datei Nr.	Endung
Zeile 1	hoffmann	L	hense	sonor	S1	I	F5	1	wav
Zeile 2	oba	L	marin	spitz	S5	X	G4	1	wav
Zeile 3	hoffmann	L	wichmann	sonor	S1	VI	Bb5	3	wav

3.1.3. Matrix für die kodierten Namen der Audiodateien

Aus dem Listing 3.2 und der Tabelle 3.2 ist sichtbar, mit welcher Systematik die Namen der Audiodateien vergeben wurden. Die einzelnen Informationen, die aus den Dateinamen zu entnehmen sind, sollen binär kodiert und in eine Bitmap als Octave eigenes Matrix-Format gespeichert werden.

Diese Matrix hat die Form wie in Abbildung 3.3 dargestellt. In einer 48 Bit langen Bitmap kann man die Werte aller Kodierungskategorien unterbringen, dazu bleiben noch drei Bits als Reserve für eventuelle weitere Anwendungen. Die binäre Kodierung der

Dateinamen soll Vorteile in der Selektionsphase bringen. Auf die Matrix kann man mathematische Operationen anwenden. Das erleichtert die Suche von Dateien und bringt einen Performance-Gewinn bei Berechnungen im Vergleich, wenn man die Selektion durch String-Suche durchführt.

Die Matrizes werden in Verzeichnissen “TrainCodiert“ und “TestCodiert“ abgespeichert, die Unterverzeichnisse des “CUED-Container“ sind. (Siehe Abbildung 3.4.) Jedes Gitarren-Verzeichnis, das im “Audiodateien-Container“ eingelesen wird, wird in einer eigenen Matrix abgespeichert. Dabei werden die Namen der Matrix-Dateien nach den folgenden Regeln vergeben:

```
<test oder train>_<Nummer des Gitarren-Verzeichnisses>_
<Name des Gitarren-Verzeichnisses>.codiert
```

Dabei ist zu beachten, dass die Gitarren-Verzeichnisse in lexikalischer Reihenfolge (so wie die in den Audiodateien-Containern vorliegenden) eingelesen und extrahiert werden. Nach lexikalischer Reihenfolge werden die Nummern der Gitarren-Verzeichnisse vergeben. (Siehe Listing 3.3.)

Listing 3.3: Nomenklatur der Matrizes für für die kodierten Audiodateien

```
test_1_Hense.codiert
test_2_Marin.codiert
train_1_Hense.codiert
train_3_Wichmann.codiert
```

Tabelle 3.3.: Aufbau der Matrix mit kodierten Audiodateien

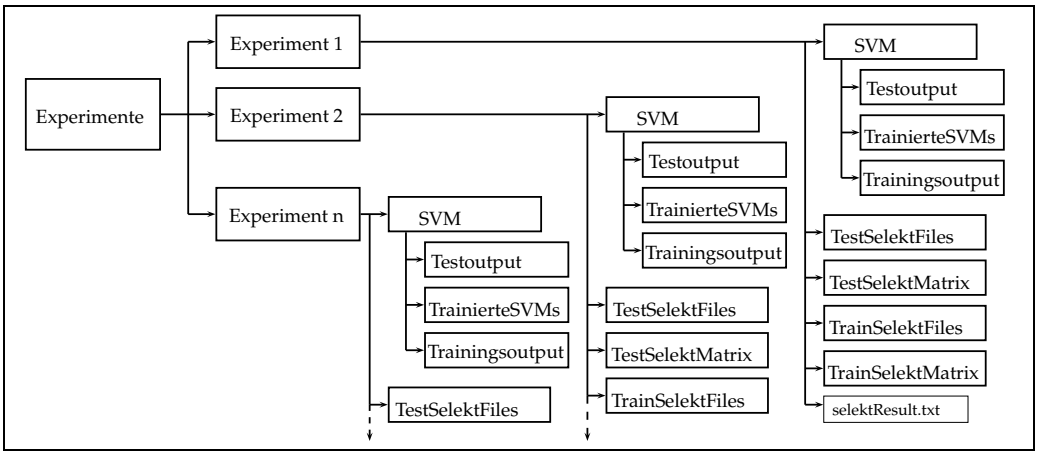
Kategorie Bitsanzahl	Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
Audiodatei 1	x	x	xx	xxxxxxxxxxxxxxxxxxxx	xxx	xxxxxx	xxxxxxxxxxxx
Audiodatei 2	x	x	xx	xxxxxxxxxxxxxxxxxxxx	xxx	xxxxxx	xxxxxxxxxxxx
...	...	...	...	...	...	...	...
Audiodatei n	x	x	xx	xxxxxxxxxxxxxxxxxxxx	xxx	xxxxxx	xxxxxxxxxxxx

(alle “x“ Variablen in der Tabelle können binäre Werte “0“ oder “1“ annehmen)

3.2. Selektions-, Trainings- und Testphase (STT-Phase)

Selektiert werden sollen die Daten, die als Octave Matrizes in den Unterverzeichnissen der “CUED-Container“ vorliegen. (Siehe Abbildung 3.4.) Die Matrizes beinhalten extrahierte und kodierte Daten. In der Selektionsphase sollen die Anforderungen so erfüllt

Abbildung 3.6.: Inhalt des Stammverzeichnisses “Experimente“



werden, dass man bestimmte Reihen einer Matrix selektieren kann, und außerdem sollen auch die Spalten, die man auswählen möchte, frei definierbar sein. Dabei ist zu beachten, dass Spaltenwerte für MFCCs und Harmonische nur blockweise (Default-Werte 50 bzw. 40) selektierbar sind. Bei den 15 Werten, die zu den Daten des nontonalen Spektrums gehören, sollen alle Werte einzeln selektierbar sein. An dieser Stelle sollen die Daten vorbereitet werden, die in der SVM Trainings- und Textphase benutzt werden. Die Trainings- und Testdaten werden an SVM in Form einer Textdatei übergeben, die ein bestimmtes Format haben soll, das von SVM erwartet wird.

3.2.1. Dateien in der Selektionsphase

Die Abbildung 3.6 zeigt die Verzeichnisstruktur, die für das “Experiment 1“ vollständig abgebildet ist und für die weiteren Experimente angedeutet ist. Diese Verzeichnisstruktur ist in jedem Experiment wieder zu finden, das im Rahmen dieser Arbeit durchgeführt wurde.

Die Verzeichnisse “TrainSelektMatrix“ und “TestSelektMatrix“ beinhalten nur die Daten, die für das aktuelle Experiment gebraucht werden. (Siehe Listings 3.4 und 3.5.) Diese Daten haben das gleiche Format (Octave Matrix) wie die in der Tabelle 3.1 vorgestellten extrahierten Audiodateien. Nur handelt es sich jetzt in der Regel um eine Daten-Untermenge der extrahierten Audiodateien. Die Namen der Dateien werden von den Namen der extrahierten Dateien in der CE-Phase übernommen. Geändert wird nur die Endung der Datei in .selektiert, um die Daten unterscheiden zu können.

Listing 3.4: Nomenklatur der Matrizes für die selektierten Train-Audiodateien

```
train_1_Hense.selektiert
train_2_Marin.selektiert
train_3_Wichmann.selektiert
```

Listing 3.5: Nomenklatur der Matrizes für die selektierten Test-Audiodateien

```
test_1_Hense.selektiert
test_2_Marin.selektiert
test_3_Wichmann.selektiert
```

Die selektierten Daten werden für die Benutzung in den SVMs vorbereitet und in den Verzeichnissen “TrainSelektFiles“ und “TestSelektFiles“ abgespeichert. Die SVM Trainings-Dateien werden (nach dem in der Abbildung 2.5 vorgestellten Graph) durch Kombinieren der .selektiert-Train-Dateien erzeugt. Es entstehen drei Dateien, die in der Trainingsphase benutzt werden. (Siehe Listing 3.6.)

Listing 3.6: Nomenklatur für die selektierten Train-Audiodateien

```
1HenseVs2Marin.data
1HenseVs3Wichmann.data
2MarinVs3Wichmann.data
```

Die SVM Test-Daten werden auch aus den .selektiert-Test-Dateien erzeugt. Wobei aus jeder Reihe der Matrix eine .data-Test-Datei erzeugt wird. (Siehe Listing 3.7.) Die Namen der Dateien werden nach den folgenden Regeln vergeben:

```
<Nummer der Gitarre>_<Namme der Gitarre>_
<Nummer des Merkmalsvektors>.data
```

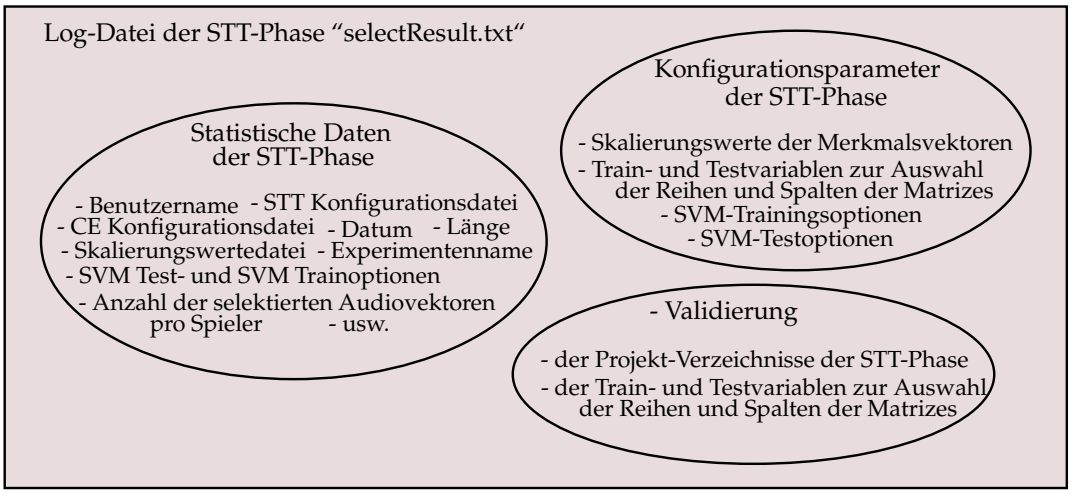
Listing 3.7: Nomenklatur für die selektierten Test-Audiodateien

```
1_Hense_40.data
2_Marin_5.data
3_Wichmann_122.data
```

Mit den so vorbereiteten Daten wird SVM trainiert und getestet. Die Ergebnisse werden im “SVM“ Verzeichnis und dessen Unterverzeichnissen abgespeichert.

Am Anfang der STT-Phase wird eine Log-Datei (“selektResult.txt“, siehe Abbildung 3.6) erzeugt und während dieser Phase mit den verschiedenen Informationen erweitert, die als Hilfe bei der Auswertung der Qualität des Klassifikationsprozesses eingesetzt werden können. Die Informationen, die dort gespeichert werden, kann man in drei Gruppen aufteilen. Das sind statistische Werte über den Verlauf der STT-Phase, die Ergebnisse der Validierung von Variablen und das Abspeichern der Werte der Variablen, die benutzt

Abbildung 3.7.: Inhalt der Log-Datei “selectResult.txt“



wurden. Die Abbildung 3.7 zeigt den Aufbau der Log-Datei und bietet eine Übersicht über die Informationen, die in dieser Datei zu finden sind. Die komplette Log-Datei des Demo-Projekts ist in dem Listing E.7 zu sehen.

3.2.2. Durchführung der Selektion

Die Anforderung, die Audiovektoren (Reihen der Matrix) und extrahierte Merkmale (Spalten der Matrix) flexibel selektieren zu können, wird durch das Setzen der Bits in den vorbereiteten Bitmaps erreicht. Es stehen sieben Kategorien (Bitmaps) zur Verfügung, die zur Selektion der Audiovektoren benutzt werden können (Abbildung 3.4) und drei Kategorien, die zur Selektion der extrahierten Merkmale eingesetzt werden (Abbildung 3.5). Die Bitmaps zur Selektion der Daten sind in der Konfigurationsdatei zu finden. Das Listing E.3 zeigt die Beispiel-Konfigurationsdatei des Demo-Projekts.

Tabelle 3.4.: Bitmaps zur Auswahl von Audiovektoren

Training						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
x	x	xx	xxxxxxxxxxxxxxxxxxxx	xxx	xxxxxx	xxxxxxxxxxxx
Test						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
x	x	xx	xxxxxxxxxxxxxxxxxxxx	xxx	xxxxxx	xxxxxxxxxxxx

(x kann die binären Werte “0“ (gelöscht) oder “1“ (gesetzt) einnehmen)

Tabelle 3.5.: Bitmaps zur Auswahl von extrahierten Merkmalen

Training		
Nontonales Spektrum [Bit 1-15]	MFCCs* [Bit 1-10]	Zeitlicher Verlauf der Harmonischen** [Bit 1-16]
xxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxxxx
Test		
Nontonales Spektrum [Bit 1-15]	MFCCs* [Bit 1-10]	Zeitlicher Verlauf der Harmonischen** [Bit 1-16]
xxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxxxx

(x kann die binären Werte “0“ (gelöscht) oder “1“ (gesetzt) einnehmen)

*(Blockweise selektierbar 50 Werte pro Bit)

**(Blockweise selektierbar 40 Werte pro Bit)

Die Listings 3.8 und 3.9 zeigen die Auszüge aus der Konfigurationsdatei des Demo-Projekts. Die gezeigten Einträge stellen die Variablen und dazugehörigen Werte dar, die für die Selektion der Daten in der STT-Phase relevant sind.

Listing 3.8: Auszug aus der Beispiel-Konfigurationsdatei (Trainingsdaten)

```
Die Variablen, die zur Auswahl der Reihen der Matrix dienen:
trainTrain     1
```

```
testTrain      0
channelTrain   10
playerTrain    1111-0000-0000-0000-0000
timbreTrain    111
stringTrain    1111-11
fretTrain      1000-1100-1100
reserveTrain   000
Die Variablen, die zur Auswahl der Spalten der Matrix dienen:
bm_nonTonalRegionenTrain  0000-0000-0000-000
bm_MFCCAnzahlTrain        0000-0000-00
bm_obertoeneAnzahlTrain   1111-1111-1111-1111
```

Listing 3.9: Auszug aus der Beispiel-Konfigurationsdatei (Testdaten)

```
Die Variablen, die zur Auswahl der Reihen der Matrix dienen:
trainTest     0
testTest      1
channelTest   10
playerTest    1111-0000-0000-0000-0000
timbreTest    111
stringTest    1111-11
fretTest      1000-1100-1100
reserveTest   000
Die Variablen, die zur Auswahl der Spalten der Matrix dienen:
bm_nonTonalRegionenTest  0000-0000-0000-000
bm_MFCCAnzahlTest        0000-0000-00
bm_obertoeneAnzahlTest   1111-1111-1111-1111
```

3.2.3. Trainings- und Testphase

Die SVM erwartet die Training- und Testdaten, die in einem Text-File gespeichert sind. Die ersten Zeilen können Kommentare beinhalten und werden ignoriert, wenn sie mit dem #-Zeichen anfangen. Es wird folgendes Format erwartet:

```
<line> .=. <target> <feature>:<value> ... <feature>:<value> # <info>
```

- zeigt das Format, das eine Zeile des Text-Files haben soll. Wobei eine Zeile des Text-Files einem Merkmalsvektor entspricht.

```
<target> .=. +1 | -1 | 0 | <float>
```

- ist die Klasse und kann in der Trainingsphase Werte +1 oder -1 annehmen. In der Testphase wird der Wert der Klasse auf Null gesetzt.

```
<feature> .=. <integer> | "qid"
```

- ist die Nummer des Merkmals. Merkmale, deren Wert Null ist, können weggelassen werden.

```
<value> .=. <float>
```

- ist der Wert des Merkmals.

```
<info> .=. <string>
```

- ist ein Kommentar.

Der wichtige Punkt ist die Konfiguration der SVM-Trainingsphase und SVM-Testphase. Es sollen die verschiedenen SVM-Kernel und deren Parameter eingestellt werden können, die zur Optimierung der Trainingsphase und der Klassifikationsergebnisse benutzt werden. Zu diesem Zwecke wird das Matlab-Interface zu SVM^{light} von Anton Schwaighofer[7] benutzt, das auch innerhalb von Octave eingesetzt werden kann.

Die Konfiguration der SVM wird durch zwei Octave Funktionen durchgeführt. Der Name der Funktionen kann frei vergeben werden und wird als Parameter in der STT-Phase-Konfigurationsdatei eingegeben. In den Listings E.5 und E.6 ist der Inhalt der “trainOptionsDefault.m“ und “testOptionsDefault.m“ Funktionen zu sehen. Der Überblick über die Parameter, die konfigurierbar sind, ist in dem Listing E.10 zu sehen.

[7] Download unter: `http://ida.first.fraunhofer.de/~anton/software.html`

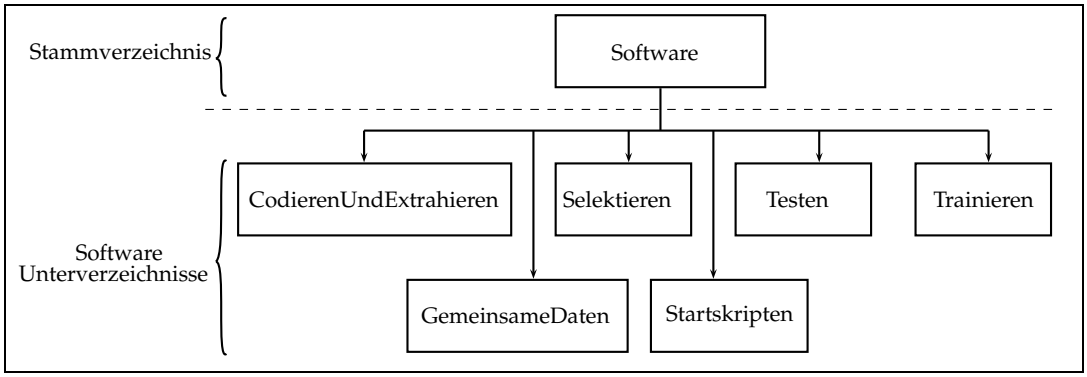
4. Software Implementierung

Zusätzlich zu der in diesem Kapitel beschriebenen Software-Implementierung sind in den Kapiteln A und B ergänzende Informationen zu finden. Dort wird der Inhalt und die Benutzung von wichtigen Dateien des Projekts (Konfigurationsdateien, Log-Dateien und Startskripten) in der CE- und STT-Phase am Beispiel eines Demo-Projekts genauer beschrieben.

4.1. CE-Phase

Im Stammverzeichnis “Software“ sind die Octave Funktionen- und Skript-Dateien in einer Unterverzeichnisstruktur unterbracht worden. (Siehe Abbildung 4.1.) In diesen Unterverzeichnissen sind die Software-Files für einzelne Phasen des Klassifikationsprozesses abgespeichert. Das Verzeichnis “"GemeinsameDaten“ beinhaltet die Software-Files, die von allen fünf einzelnen Phasen des Klassifikationsprozesses benutzt werden. Das Verzeichnis “Startskripten“ beinhaltet sämtliche Startskripten und Konfigurationsdateien für die Experimente, die im Rahmen dieser Arbeit durchgeführt wurden.

Abbildung 4.1.: Stammverzeichnis “Software“



4.1.1. Octave Funktions- und Skript-Dateien

Die Abbildung 4.2 zeigt die Funktionen, Startskripten und die Konfigurationsdatei, die in der CE-Phase verwendet werden. Aus der Abbildung kann man die Abhängigkeiten zwischen den Dateien erkennen. Die Zuordnung der Dateien zu den Verzeichnissen ist zu sehen und die hierarchische Beziehung zwischen den Dateien ist erkennbar. Weiter unten folgt eine Beschreibung der Funktions- und Skript-Dateien, die in der CE-Phase verwendet werden.

4.1.1.1. Dateien im Verzeichnis “Startskripten“

● Skript: *“start_ce_xxx.m“*
In diesem Skript wird Octave Load-Path auf das “Software“ Verzeichnis und alle Software Unterverzeichnisse erweitert. Das heißt, dass man von dieser Skript-Datei alle relevanten Software-Files aufrufen kann. Die Konfigurationsdatei wird geladen, die Variablen der Konfigurationsdatei werden in einem Cell-Array gespeichert. Abschließend wird die “startCodierenUndExtrahieren.m“ Skript-Datei aufgerufen und somit die CE-Phase gestartet.

● Konfigurationsdatei: *“config_ce_xxx.txt“*
Beinhaltet die Konfigurationsparameter (Variablen und dazugehörigen Werte), die den Ablauf der CE-Phase bestimmen.

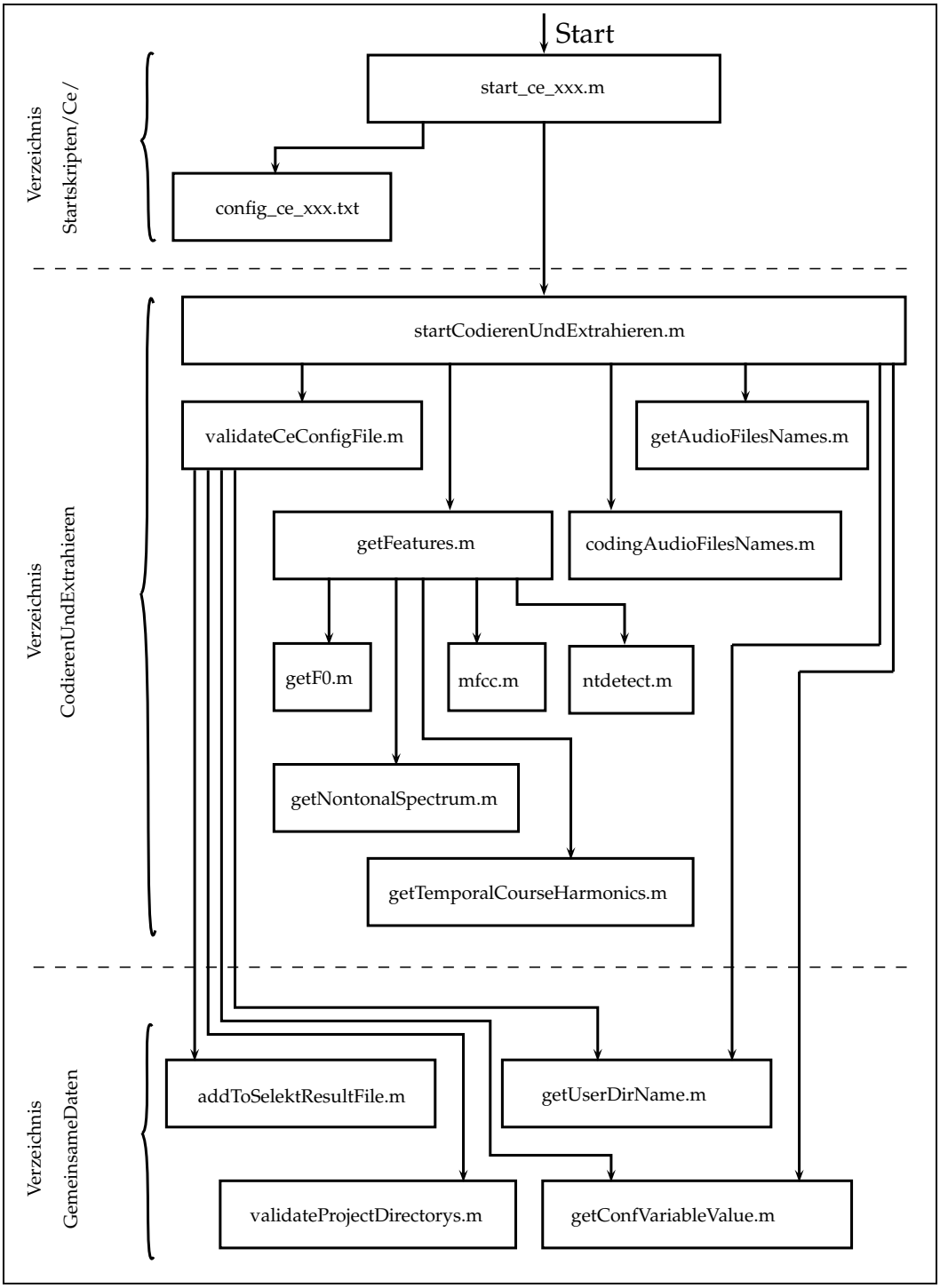
4.1.1.2. Dateien im Verzeichnis “CodierenUndExtrahieren“

● Funktion: *“codingAudioFilesNames.m“*
Diese Funktion liest die übergebene Namensliste, die die Namen der Audiodateien beinhaltet, analysiert die Namen der Audiodateien, kodiert sie und schreibt die Ergebnisse in eine Matrix.

● Funktion: *“getAudioFilesNames.m“*
Liest den Inhalt der Ordner, die Test- und Trainingsdaten enthalten und erzeugt eine Liste der eingelesenen Dateinamen, die als String in ein Cell-Array abgespeichert werden.

● Funktion: *“getF0.m“*
Berechnet die Grundfrequenz nach der Cepstrum-Methode.

Abbildung 4.2.: Implementierung der CE-Phase



● Funktion: *“getFeatures.m“*
Die Funktion berechnet die angeforderten Merkmale, die in einer Merkmalsmatrix abgespeichert werden und als Rückgabeparameter geliefert werden.

● Funktion: *“getNontonalSpektrum.m“*
Berechnet das nontonale Spektrum einer Audiodatei.

● Funktion: *“getTemporalCourseHarmonics.m“*
Berechnet den zeitlichen Verlauf der Harmonischen einer Audiodatei.

● Funktion: *“mfcc.m“*
Berechnet die MFCC-Merkmale einer Audiodatei.

● Funktion: *“ntdetect.m“*
Bestimmt die Grenzen der nontonalen Regionen in 16 logarithmischen Abstand-Schritten und berechnet die gesamte Spektral-Energie des nontonalen Spektrums.

● Skript: *“startCodierenUndExtrahieren.m“*
Ist Startskript der CE-Phase.

● Funktion: *“validateCeConfigFile.m“*
Validiert die Einträge in der Konfigurationsdatei.

4.1.1.3. Dateien im Verzeichnis “GemeinsameDaten“

● Funktion: *“addToSelektResultFile.m“*
Erweitert die Log-Dateien in der CE- und STT-Phase um die Einträge, die den Verlauf dieser Phasen charakterisieren.

● Funktion: *“getUserDirName.m“*
Ermittelt das User-Verzeichnis des Betriebssystems. Falls sich das User-Verzeichnis von der Eingabe in die Konfigurationsdatei unterscheidet, wird das ermittelte User-Verzeichnis verwendet.

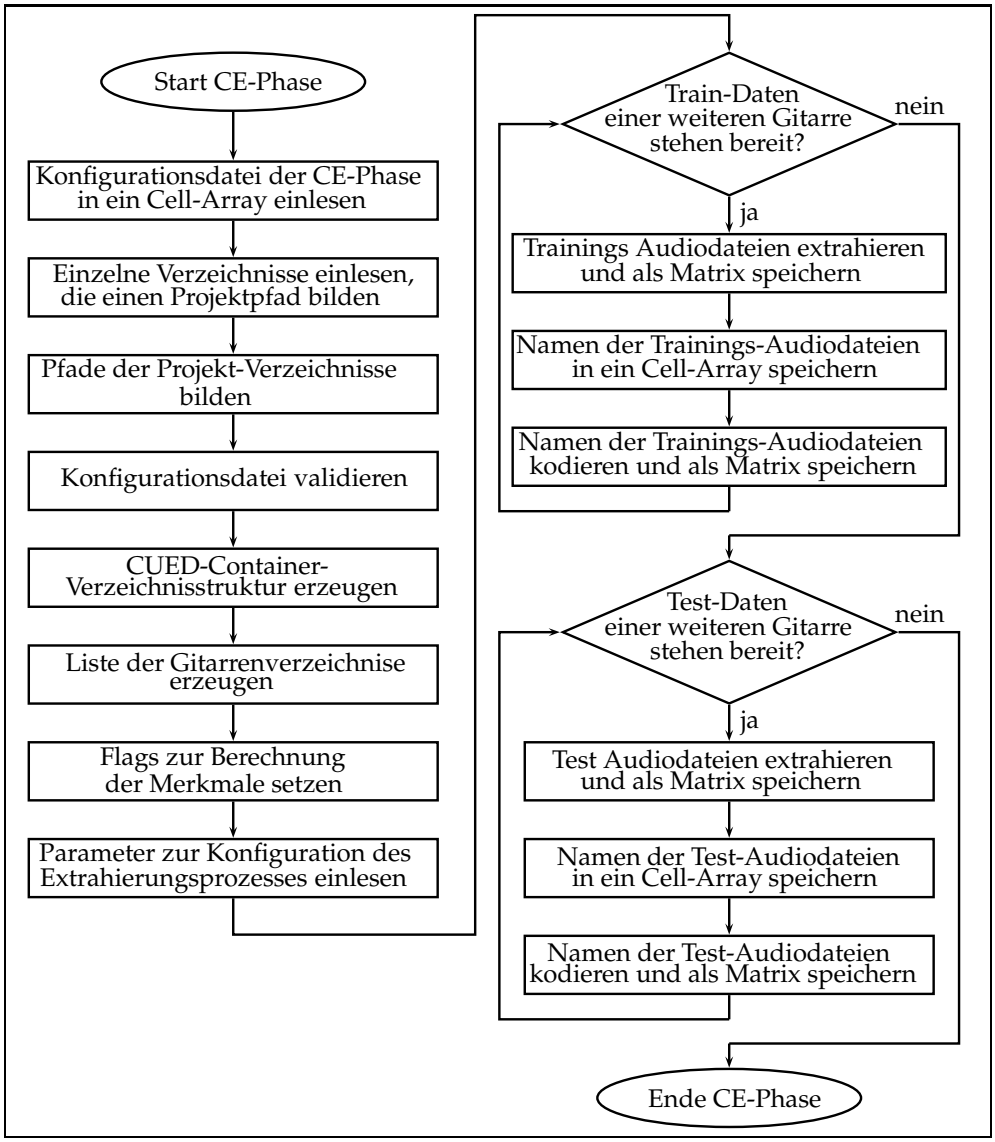
● Funktion: *“getConfVariableValue.m“*
Liefert die Werte aus der Konfigurationsdatei, die zu den gesuchten Konfigurations-Variablen gehören.

● Funktion: *“validateProjectDirectorys.m“*
Prüft die gelieferten Namen der Projekt-Verzeichnisse auf ihre Existenz und Gültigkeit.

4.1.2. Verlauf der CE-Phase

Die Abbildung 4.3 zeigt den Verlauf der CE-Phase im Form eines Flussdiagramms. Nachdem die Validierung der Konfigurationsdatei beendet ist, wird eine Verzeichnisstruktur erzeugt. Zu verdeutlichen ist, dass es sich dabei um die Erzeugung von “CUED-Container“ mit den dazugehörigen Unterverzeichnissen handelt. (Siehe Abbildung 3.4.)

Abbildung 4.3.: Flussdiagramm der CE-Phase



4.2. STT-Phase

4.2.1. Octave Funktions- und Skript-Dateien in der STT-Phase

Die Abbildung 4.4 zeigt die Funktionen, Startskripten und die Konfigurationsdatei, die in der STT-Phase verwendet werden. Aus der Abbildung kann man die Abhängigkeiten zwischen den Dateien erkennen. Die Zuordnung der Dateien zu den Verzeichnissen ist zu sehen und die hierarchische Beziehung zwischen den Dateien ist erkennbar. Weiter unten folgt eine Beschreibung der Funktions- und Skript-Dateien, die in der STT-Phase verwendet werden.

4.2.1.1. Dateien im Verzeichnis “Startskripten“

● Skript: *“start_stt_xxx.m“*
In diesem Skript wird Octave Load-Path auf das Verzeichnis “Software“ und alle Software Unterverzeichnisse erweitert. Das heißt, dass man von dieser Skript-Datei alle relevanten Software-Files aufrufen kann. Die Konfigurationsdatei wird geladen, die Variablen der Konfigurationsdatei werden in ein Cell-Array gespeichert. Abschließend wird die “start-Selektieren.m“ Skript-Datei, aufgerufen und somit die STT-Phase gestartet. Außerdem werden auch Trainings- und Testphase von hier gestartet. Es werden die “learn.m“ und “testdagsvm.m“ Startskripten aufgerufen.

● Konfigurationsdatei: *“config_stt_xxx.txt“*
Beinhaltet die Konfigurationsparameter (Variablen und dazugehörigen Werte), die den Ablauf der STT-Phase bestimmen.

● Konfigurations-Funktion: *“getScalingValueXXX.m“*
Liefert die Skalierungsparameter für die 10 MFCCs und 16 Harmonischen.

● Konfigurations-Funktion: *“trainOptionsXXX.m“*
Ruft die Trainings SVM-Optionen “svmlopt.m“ Funktion auf. Diese Funktion wird mit der Hilfe Octave “eval()“ Funktion aufgerufen. Welche “trainOptionsXXX.m“ Funktion aufgerufen wird, wird in der Konfigurationsdatei festgelegt.

● Konfigurations-Funktion: *“testOptionsXXX.m“*
Ruft die Test SVM-Optionen “svmlopt.m“ Funktion auf. Diese Funktion wird mit der Hilfe Octave “eval()“ Funktion aufgerufen. Welche “testOptionsXXX.m“ Funktion aufgerufen wird, wird in der Konfigurationsdatei festgelegt.

In den Dateien, die oben beschrieben wurden, handelt es sich bei “xxx“ oder “XXX“ um die Teile der Namen, die in der Regel bei verschiedenen Experimenten öfter angepasst werden. Das ist der Fall, wenn die Experimente mit verschiedenen Konfigurationen durchgeführt werden.

4.2.1.2. Dateien im Verzeichnis “Selektieren“

● Skript: *“startSelektieren.m“*
Aus diesem Skript werden die meisten Funktionen aufgerufen, die in der Selektionsphase benutzt werden.

● Funktion: *“validateSttConfigFile.m“*
Validiert die Einträge in der Konfigurationsdatei.

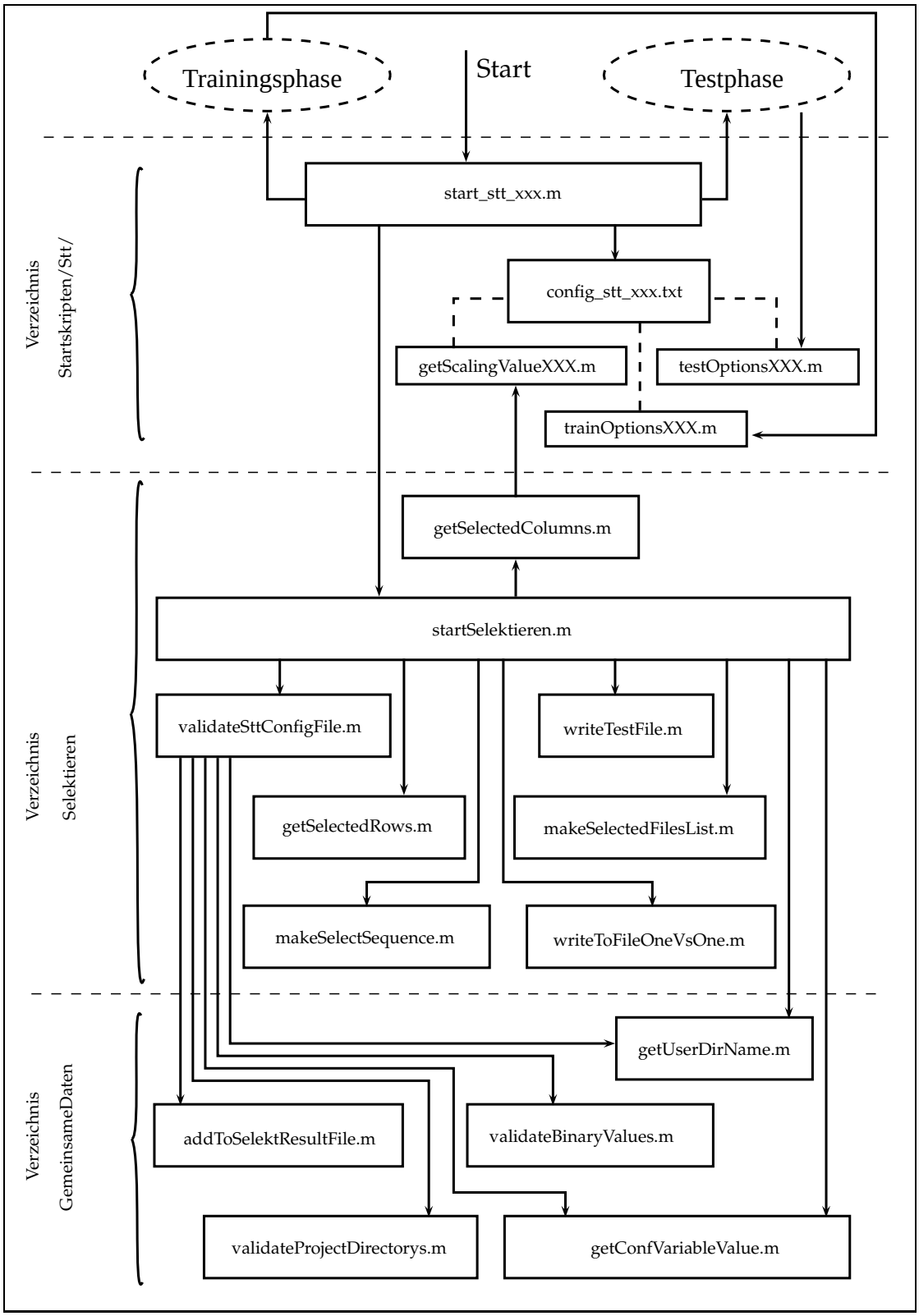
● Funktion: *“makeSelectSequence.m“*
Die Funktion liefert eine 48-Bit lange “Selekt-Sequenz“, die zur Selektion der Reihen der Merkmals-Matrix verwendet wird. Die “Selekt-Sequenz“ wird aus acht Teilen zusammengefügt und in die Form einer Bitmap zurückgeliefert.

● Funktion: *“getSelectedRows.m“*
Als Eingangsgröße bekommt die Funktion eine Matrix mit den extrahierten Merkmalen. Die Funktion liefert eine Matrix zurück, die nur Merkmalsvektoren beinhaltet, die durch die Auswahl-Bitmap (“Selekt-Sequenz“) ausgewählt wurden.

● Funktion: *“getSelectedColumns.m“*
Im zweiten Schritt wird die von “getSelectedRows.m“ gelieferte Matrix nach den ausgewählten Spalten durchsucht. Die markierten Spalten werden in eine Matrix zurückgegeben. Diese Matrix stellt gleichzeitig die gesuchte selektierte Datenmenge dar.

● Funktion: *“makeSelectedFilesList.m“*
Diese Funktion liefert eine Liste der Dateien, die sich in einem Verzeichnis befinden. Es werden nur so viele Dateien gesucht wie im Funktions-Signatur-Parameter “auswahlAnzahl“ angegeben. Die Funktion wird benutzt, um die Anzahl der Gitarren-Matrizes in den “CUED-Containern“ zu ermitteln. Die Dateien können nach bestimmten Kriterien gesucht werden: z.B. alle Dateien auswählen, einen Bereich von Startindex bis Endindex auswählen, nur bestimmten Gitarren auswählen über einen Datei-Index, oder Dateien mit geradem oder ungeradem Index auswählen usw.. Die Funktion wurde so implementiert, dass nur alle Dateien (Gitarren) ausgewählt werden können und die Anzahl der Gitarren auf drei festgesetzt wurde. Das korrespondiert mit dem DAG, der fest für die Klassifikation von drei Gitarren ausgelegt ist.

Abbildung 4.4.: Implementierung der STT-Phase



● Funktion: *“writeToFileOneVsOne.m“*
Schreibt die Trainingsdateien für die drei SVMs, die trainiert werden. Die erste Liste bekommt immer -1 und die zweite immer +1 als Klassenlabel zugewiesen.

● Funktion: *“writeTestFile.m“*
Mit Hilfe dieser Funktion werden die SVM-Testfiles vorbereitet. Für jede Audiodatei, die klassifiziert werden soll, wird eine Datei erzeugt. Die Daten werden mit dem Klassenlabel Null in die Datei geschrieben.

4.2.1.3. Dateien im Verzeichnis “GemeinsameDaten“

● Funktion: *“addToSelektResultFile.m“*
Erweitert die Log-Dateien in der CE- und STT-Phase um Einträge über den Verlauf dieser Phasen.

● Funktion: *“getUserDirName.m“*
Ermittelt das User-Verzeichnis des Betriebssystems. Falls sich das User-Verzeichnis von der Eingabe in die Konfigurationsdatei unterscheidet, wird das ermittelte User-Verzeichnis verwendet.

● Funktion: *“getConfVariableValue.m“*
Liefert die Werte, die zu den gesuchten Konfigurations-Variablen gehören.

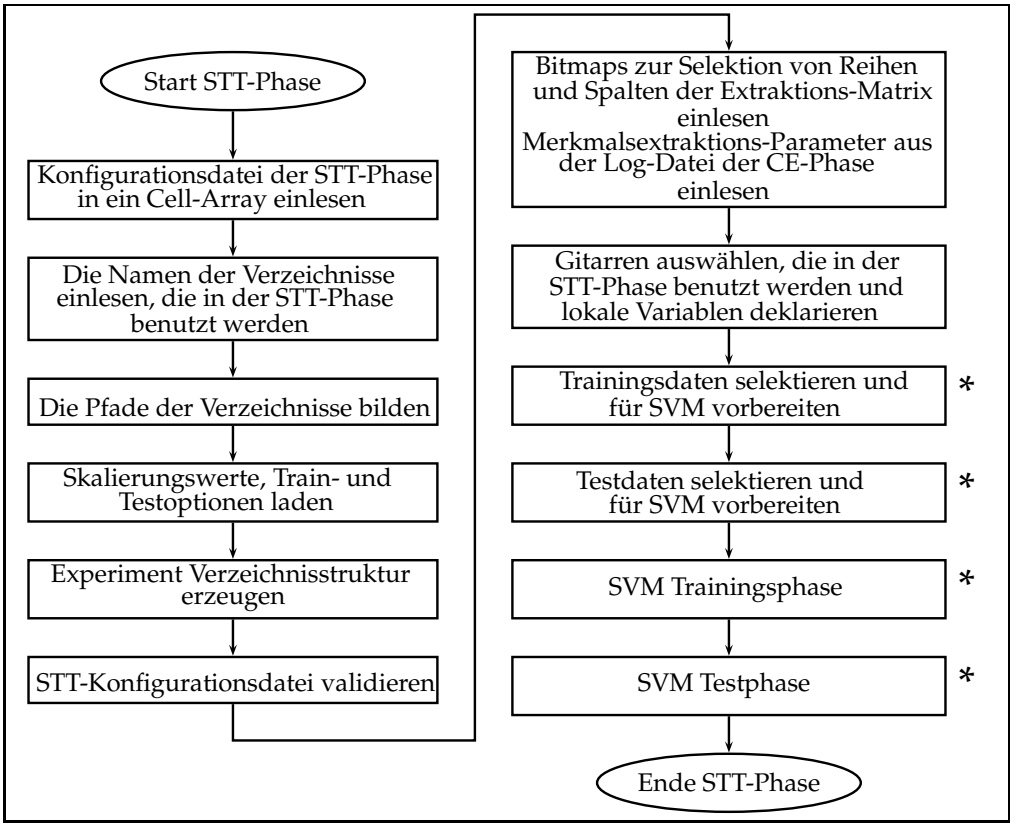
● Funktion: *“validateProjectDirectorys.m“*
Prüft die gelieferten Namen der Projekt-Verzeichnisse auf ihre Existenz und Gültigkeit.

● Skript: *“validateBinaryValues.m“*
Prüft, ob die als Funktions-Signatur-Parameter gelieferte Variable einen gültigen binären Wert hat.

4.2.2. Verlauf der STT-Phase

Das in der Abbildung 4.5 dargestellte Flussdiagramm zeigt den Verlauf der STT-Phase. Die Komponenten des Flussdiagramms, die mit dem *-Zeichen gekennzeichnet sind, werden weiter unten genauer beschrieben.

Abbildung 4.5.: Flussdiagramm der STT-Phase

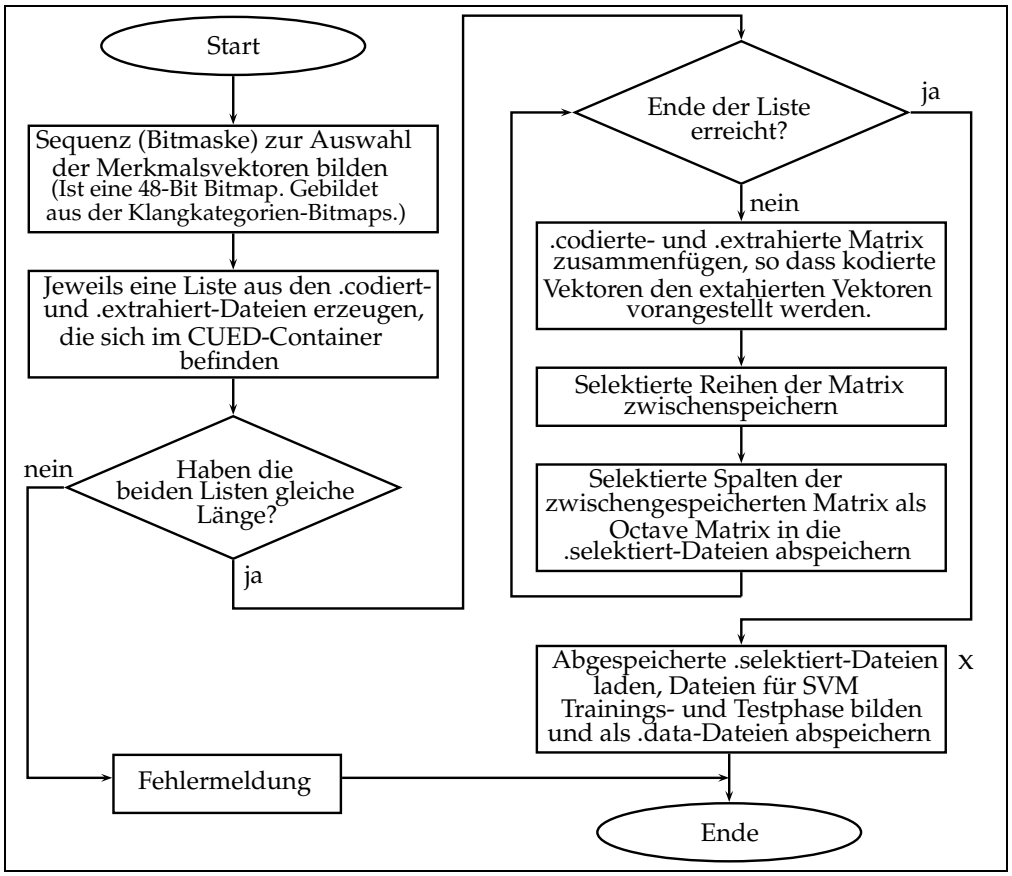


4.2.3. Verlauf der Selektionsphase

Das in der Abbildung 4.6 dargestellte Flussdiagramm zeigt den Verlauf der Selektionsphase. Das Diagramm stellt die zwei mit dem *-Zeichen gekennzeichneten Teile der Abbildung 4.5 dar: das sind “Trainingsdaten selektieren und für SVM vorbereiten“ und “Testdaten selektieren und für SVM vorbereiten“.

Die beide oben genannten Teile sind für Trainings- und Testdaten identisch aufgebaut. Bei Abbildung 4.6 liegt der einzige Unterschied in dem mit X gekennzeichneten Kästchen. Die Dateien für die SVM Trainings- und Testphase werden nach dem One-Versus-One-Verfahren vorbereitet. Für die SVM Trainingsphase werden drei Dateien erzeugt. Dagegen wird für die Testphase pro Audiodatei nur eine Testdatei erzeugt. Die Dateien werden klassifiziert nach dem azyklischen Graph (DAG), der in der Abbildung 2.5 dargestellt wurde.

Abbildung 4.6.: Vorbereitung für die SVM Trainings- und Testdaten



4.2.4. Verlauf der Trainings- und Testphase

Die Abbildung 4.7 zeigt die Funktionen, Startskripten und Konfigurationsdateien, die in der Trainings- und Testphase verwendet werden. Aus der Abbildung sind die Abhängigkeiten zwischen den Dateien und die Reihenfolge, in der die Dateien aufgerufen werden, zu erkennen. Weiter unten folgt eine Beschreibung der Funktions- und der Skript-Dateien, die in der Trainings und Testphase verwendet werden.

4.2.4.1. Dateien im Verzeichnis “Startskripten“

● Skript: *“start_stt_xxx.m“*
In diesem Skript wird Octave Load-Path auf das Verzeichnis “Software“ und alle Software Unterverzeichnisse erweitert. Das heißt, dass man von dieser Skript-Datei alle relevanten Software-Files aufrufen kann. Die Konfigurationsdatei wird geladen, die Variablen der Konfigurationsdatei werden in ein Cell-Array gespeichert. Außerdem werden auch die Trainings- und Testphase von hier gestartet. Es werden die “learn.m“ und “testdagsvm.m“ Startskripten aufgerufen.

● Konfigurationsdatei: *“config_stt_xxx.txt“*
Beinhaltet die Konfigurationsparameter (Variablen und dazugehörigen Werte), die den Ablauf der STT-Phase bestimmen.

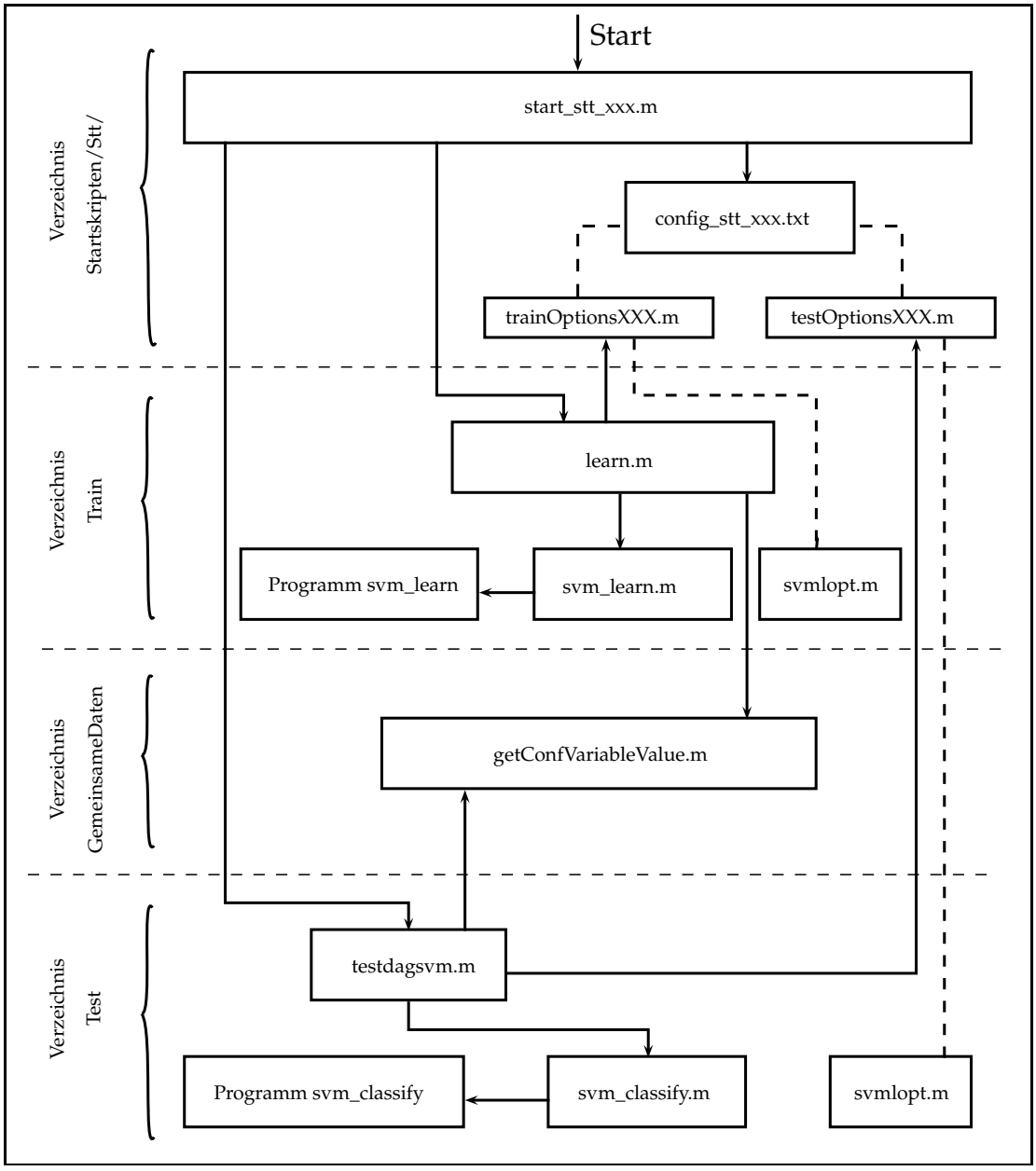
● Konfigurations-Funktion: *“trainOptionsXXX.m“*
Ruft die Trainings SVM-Optionen “svmlopt.m“ Funktion auf. Diese Funktion wird mit Hilfe der Octave “eval()“ Funktion aufgerufen. Welche “trainOptionsXXX.m“ Funktion aufgerufen wird, wird in der Konfigurationsdatei festgelegt.

● Konfigurations-Funktion: *“testOptionsXXX.m“*
Ruft die Test SVM-Optionen “svmlopt.m“ Funktion auf. Diese Funktion wird mit Hilfe der Octave “eval()“ Funktion aufgerufen. Welche “testOptionsXXX.m“ Funktion aufgerufen wird, wird in der Konfigurationsdatei festgelegt.

4.2.4.2. Dateien im Verzeichnis “GemeinsameDaten“

● Funktion: *“getConfVariableValue.m“*
Liefert die Werte, die zu den gesuchten Konfigurations-Variablen gehören.

Abbildung 4.7.: Implementierung der Trainings- und Testphase



4.2.4.3. Dateien im Verzeichnis “Train“

• Skript: *“learn.m“*
Lädt die Konfigurationsvariablen, Pfade und die Verzeichnisse, die in der Trainingsphase gebraucht werden, ruft die Funktion “svm_learn.m“ auf und startet somit die Trainingsphase. Erweitert die Log-Datei der STT-Phase um zusätzliche Einträge.

• Funktionen: *“svm_learn.m“* und *“svmlopt.m“*
Die beide Funktionen gehören zum Anton Schwaighofer SVM^{light} Interface. Durch diese Dateien wird die Trainingsphase der SVM^{light} gestartet und konfiguriert.

• Programm: *“Programm svm_learn“*
Ist das SVM^{light} Trainings-Programm.

4.2.4.4. Dateien im Verzeichnis “Test“

• Skript: *“testdagsvm.m“*
Lädt die Konfigurationsvariablen, Pfade und die Verzeichnisse, die in der Testphase gebraucht werden, ruft die Funktion “svm_classify.m“ auf und startet somit die Testphase. Erweitert die Log-Datei der STT-Phase um zusätzliche Einträge.

• Funktionen: *“svm_classify.m“* und *“svmlopt.m“*
Die beiden Funktionen gehören zum Anton Schwaighofer SVM^{light} Interface. Durch diese Dateien wird die Testphase der SVM^{light} gestartet und konfiguriert.

• Programm: *“Programm svm_classify“*
Ist das SVM^{light} Test-Programm.

4.3. Überblick

Die entwickelte Softwareumgebung kann in der aktuellen Version nur die Audiodateien von drei Gitarren klassifizieren. Die Software ist jedoch für die flexible Klassifikation einer variablen Anzahl von Gitarren weitgehend vorbereitet worden. So ist die gesamte Software, die in der CE-Phase benutzt wird, für den Umgang mit mehreren Gitarren konzipiert worden. Alle Dateien, die in der CE-Phase erzeugt werden, werden durch die Vergabe der eindeutigen Reihennummern kennzeichnet, die in die Namen der Dateien integriert werden. Das ermöglicht eine präzise Selektion der Daten in der STT-Phase.

Die Software, die für die STT-Phase entwickelt wurde, ist auch (bis auf zwei Files) für den Umgang mit mehreren Gitarren vorbereitet worden. Die Funktion “makeSelectedFilesList.m“ und das Skript “startSelektieren.m“, die in dem Verzeichnis “/Software/Selektieren/“zu finden sind, sollen noch erweitert werden. Die Funktion “makeSelectedFilesList.m“ soll ab der Zeile 52 weiter entwickelt werden. Das Skript “startSelektieren.m“ soll um die Vorbereitung der Dateien (nach One-Vs-One-Verfahren) für die Trainingsphase erweitert werden. Der jetzige Source-Code, der ab der Zeile 249 bis 272 zu finden ist, soll umgeschrieben werden.

Um die SVM Testphase flexibel gestalten zu können, soll das Skript “testdagsvm.m“, das sich in dem “/Software/Test/“ Verzeichnis befindet, neu implementiert werden.

5. Experimente zur Skalierung von Merkmalsvektoren

Schon beim Betrachten des Kurvenverlaufs von einzelnen MFCCs (Abbildung 5.1, Abbildung aus [Dos07]) und Harmonischen (Abbildung 5.2, Abbildung aus [Dos07]) am Beispiel von zwei zufällig ausgewählten Audiosignalen sieht man, dass sich die Größen der Werte der einzelnen Kurven voneinander unterscheiden. Das Ziel ist, die Werte der einzelnen MFCCs bzw. Harmonischen, die kleiner sind anzuheben und so genau wie möglich an die Größe der 1. MFCC bzw. 1. Harmonischen anzupassen.

Also soll für jeden einzelnen MFCC, bzw. jede Harmonische der Skalierungsfaktor gefunden werden, der als Multiplikator eingesetzt wird. Diese Vorgehensweise ist zulässig, weil die Daten für die SVM Tainings- und Test-Phase mit gleichen Skalierungsfaktoren multipliziert und somit einheitlich verändert werden.

Abbildung 5.1.: Die ersten 10 MFCCs eines Audiosignals

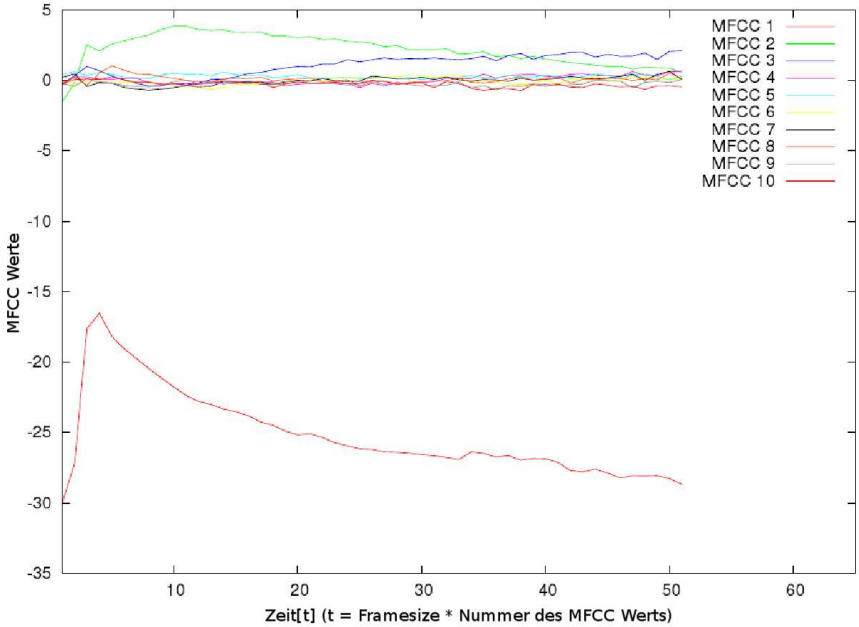
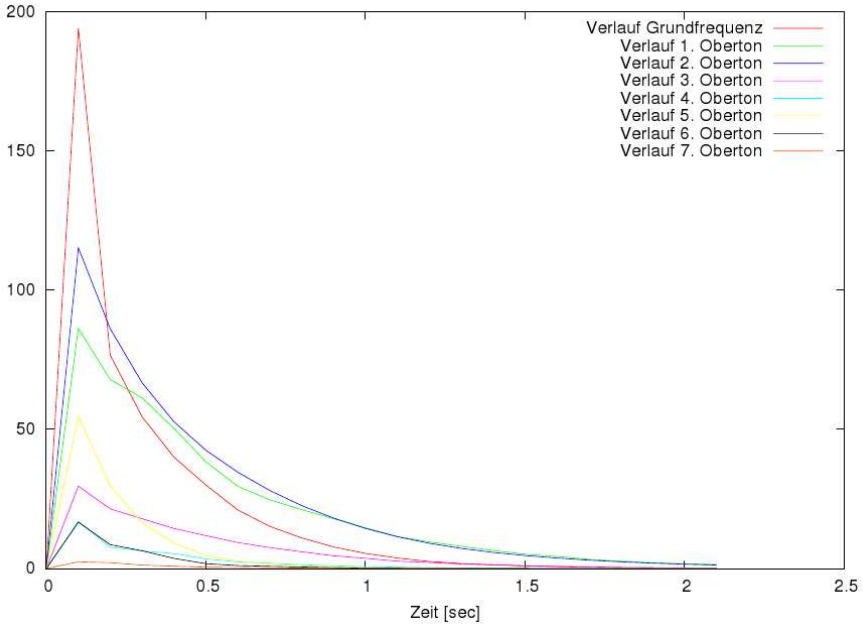


Abbildung 5.2.: Die ersten 7 Harmonischen eines Audiosignals



5.1. Durchführung der Experimente

Als erstes ist eine Auswahl von Audiosignalen getroffen worden, mit denen anschließend die Experimente zur Ermittlung von Skalierungsfaktoren durchgeführt worden sind. Ausgewählt sind insgesamt 108 Audiosignale nach den Kriterien, die aus den Tabellen 5.1 und 5.2 zu entnehmen sind.

Tabelle 5.1.: Bitmaps zur Auswahl von Audiosignalen zur Ermittlung der Skalierungsfaktoren

Training						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
1	0	10	11110000000000000000	010	101010	100000000000
Test						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
0	1	10	11110000000000000000	010	101010	100000000000

Die Werte für die Skalierungsfaktoren sind auf folgende Weise ermittelt worden:

1. Bildung der Summen der Quadrate aller Skalare, die zu einem MFCC, bzw. einer

Tabelle 5.2.: Bitmaps zur Auswahl von extrahierten Merkmalsvektoren zur Ermittlung der Skalierungsfaktoren

Training		
Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Zeitlicher Verlauf der Harmonischen [Bit 1-16]
111111111111111	1111111111	1111111111111111
Test		
Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Zeitlicher Verlauf der Harmonischen [Bit 1-16]
111111111111111	1111111111	1111111111111111

Harmonischen gehören. Für jedes einzelne Audiosignal ist diese Berechnung für 10 MFCCs und 16 Harmonischen durchgeführt worden. Quadrieren ist auf dieser Stelle erforderlich, weil es in den Merkmalsvektoren auch Zahlen gibt, die negative Vorzeichen haben. Durch Quadrieren bekommen alle Zahlen das positive Vorzeichen, und somit wird die Summe für alle MFCCs und Harmonischen, wie angestrebt, nur mit positiven Werten gebildet.

2. Durch Vergleich des ersten MFCC und der ersten Harmonischen ist festgestellt worden, dass die beiden Summen in gleicher Größenordnung liegen. Deshalb konnte man dem ersten MFCC bzw. der ersten Harmonischen den Skalierungsfaktor “1“ zuweisen. Alle weiteren Faktoren sind durch Wiederholen der Multiplikation und Vergleichen der Ergebnisse mit dem ersten MFCC bzw. der ersten Harmonischen ermittelt worden. Siehe Pseudocode Listing 5.1 und Listing 5.2.

5.1.1. Pseudocode zur Ermittlung von Skalierungsfaktoren

In der äußeren While-Schleife werden die MFCCs gezählt. Die innere While-Schleife wird so lange wiederholt, bis die Summe vom ersten MFCC_1 größer ist als die Summe des aktuellen MFCC_X, multipliziert mit dem temporären Skalierungsfaktor. Bei jedem Durchlauf der inneren While-Schleife wird der temporäre Skalierungsfaktor um 10 erhöht. Nach Verlassen der inneren While-Schleife wird der temporäre Wert des Skalierungsfaktors als Ergebnis abgespeichert und der Zähler für MFCCs inkrementiert. Zu beachten ist, dass X in den Variablen “MFCC_X“ und im “skalierungsfaktor_X“ der Platzhalter ist und stellvertretend für den Wert des aktuellen MFCC-Zählers X steht.

Listing 5.1: Pseudocode zur Ermittlung von MFCC Skalierungsfaktoren

```
X = 1;  // Zaehler Variable
skalierungsfaktorTmp = 10;  // Temporaere Variable
```

```
skalierungsfaktor_X = 0;    // Berechneter Skalierungsfaktor

while(X <= 10))
  while ((MFCC_1 * 1) > (MFCC_X * skalierungsfaktorTmp)
  {
    skalierungsfaktorTmp = skalierungsfaktorTmp + 10;
  }
  skalierungsfaktor_X = skalierungsfaktorTmp;
  X = X + 1;
}
```

Die obere Beschreibung des Pseudocodes für die Berechnung von Skalierungsfaktoren für MFCCs trifft auch hier voll zu. Der einzige Unterschied ist, dass hier 16 Skaliereungsfaktoren für 16 Harmonische berechnet werden sollen statt, wie oben beschrieben, 10 Skaliereungsfaktoren für 10 MFCCs.

Listing 5.2: Pseudocode zur Ermittlung der Skalierungsfaktoren von Harmonischen

```
X = 1;  // Zaehler Variable
skalierungsfaktorTmp = 10;      // Temporaere Variable
skalierungsfaktor_X = 0;    // Berechneter Skalierungsfaktor

while(X <= 16))
  while ((Harmonischen_1 * 1) > (Harmonischen_X *
     skalierungsfaktorTmp)
  {
    skalierungsfaktorTmp = skalierungsfaktorTmp + 10;
  }
  skalierungsfaktor_X = skalierungsfaktorTmp;
  X = X + 1;
}
```

5.2. Ergebnisse

In den Tabellen 6.1 und 7.1 sind die ermittelten Skalierungsfaktoren von MFCCs und Harmonischen zu sehen.

Tabelle 5.3.: Überblick über angepasste MFCC Skalierungsfaktoren

MFCCs										
Nummer	1	2	3	4	5	6	7	8	9	10
Faktor	1	10	50	50	100	100	100	100	100	100

Die Experimente, die in den Kapiteln 6 und 7 durchgeführt werden, werden einerseits mit hier ermittelten Skalierungsfaktoren berechnet und anderseits parallel dazu mit unveränderten Merkmalsvektoren. Durch den Vergleich der Ergebnisse beider Experimenten-Reihen wird festgestellt, ob man damit eine Verbesserung der Klassifikationsergebnisse erzielen konnte.

Tabelle 5.4.: Überblick über die angepassten Skalierungsfaktoren von Harmonischen

Zeitlicher Verlauf der Harmonischen (1-8)								
Nummer	1	2	3	4	5	6	7	8
Skalierungsfaktor	1	1	1	10	10	10	10	100
Zeitlicher Verlauf der Harmonischen (9-16)								
Nummer	9	10	11	12	13	14	15	16
Skalierungsfaktor	100	100	100	1000	1000	1000	1000	1000

6. Experimente zu Mel Frequency Cepstral Coefficients (MFCCs)

In diesem Kapitel wird der Einfluss der einzelnen MFCCs auf die Ergebnisse der Klassifizierung der Audiosignale untersucht. Zu erwarten ist, dass die einzelnen MFCCs unterschiedlich stark die Klassifizierungsergebnisse beeinflussen und dass man sogar im günstigsten Fall auf einige der 10 MFCCs im Klassifizierungsprozess verzichten könnte, ohne dabei die Ergebnisse negativ zu beeinflussen.

6.1. Überblick

An dieser Stelle sind folgende zwei Gruppen von Experimenten zu unterscheiden. Für beide Gruppen der Experimente wurde ein zusätzlicher Hauptmultiplikator eingeführt. Diese zusätzliche Skalierung mittels Hauptmultiplikatoren wurde gemacht, um zu prüfen, ob eine solche Anhebung der Werte positiven Einfluss auf die Berechnung der SVM Hyperebene zwischen den einzelnen Klassen bewirken könnte.

6.1.1. Experimente mit angepassten Skalierungsfaktoren

Es wurden insgesamt 15 Experimente “**mit** angepassten Skalierungsfaktoren“ durchgeführt. Die Experimente 1 - 10 haben Hauptmultiplikator “1“. Also blieben die Werte in der Datenmatrix in ihrer Größe unverändert. Die Experimente 11 - 15 bekamen die Hauptmultiplikatoren, die aus der Tabelle 6.5 zu entnehmen sind.

In der Tabelle 6.1 sind Skalierungsfaktoren für MFCC 1 bis MFCC 10 zu sehen. Alle Experimente “**mit** angepassten Skalierungsfaktoren“ wurden mit diesen Einstellungen durchgeführt.

Tabelle 6.1.: Überblick über die angepassten MFCC Skalierungsfaktoren

MFCCs										
Nummer	1	2	3	4	5	6	7	8	9	10
Faktor	1	10	50	50	100	100	100	100	100	100

6.1.2. Experimente ohne angepasste Skalierungsfaktoren

Es wurden auch insgesamt 15 Experimente "**ohne** angepasste Skalierungsfaktoren" durchgeführt. Die Experimente 101 - 110 haben Hauptmultiplikator "1". Also blieben die Werte in der Datenmatrix in ihrer Größe unverändert. Die Experimente 111 - 115 bekamen die Hauptmultiplikatoren, die aus der Tabelle 6.5 zu entnehmen sind.

6.2. Vorbereitung der Experimente

Die folgenden Experimente gehören zu der Gruppe von Experimenten, die zur Reduktion der Menge von Daten der Merkmalsvektoren dienen, bzw. zur Optimierung der Anzahl von Spalten in der Datenmatrix. In der Tabelle 6.2 sind die Bitmaps zu sehen, deren Einstellungen gültig sind für alle Experimente, die in diesem Kapitel beschrieben wurden. Hier sind alle verfügbaren Audiovektoren ausgewählt worden.

Tabelle 6.2.: Bitmaps zur Auswahl von extrahierten Audiodateien

Training						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
1	0	10	11110000000000000000	111	111111	100011001100
Test						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
0	1	10	11110000000000000000	111	111111	100011001100

Der MFCC Merkmalsvektor beinhaltet insgesamt 10 MFCCs. Jeder MFCC hat je 50 Zahlenwerte, das ergibt insgesamt 500 Zahlenwerte pro Audiosignal. Welche MFCCs im Klassifizierungsprozess berücksichtigt werden, wird in der MFCC-Bitmap festgehalten, die zusammen mit anderen konfigurierbaren Parametern in der Konfigurationsdatei für die STT-Phase abgespeichert wurde.

Die MFCC-Bitmap wurde auf zwei verschiedene Arten aufgeteilt, um mit möglichst wenigen Experimenten alle MFCCs im Merkmalsvektor ausreichend getestet zu haben. Die

Tabelle 6.3 zeigt die Aufteilung der MFCC-Bitmap in 2 Teile. Das ergibt $((2^2) - 1)$ Kombinationen, die in 3 Experimenten berechnet wurden.

Tabelle 6.3.: Bitmap Aufteilung in 2 Gruppen

MFCC Bitmap Gruppen Bitmap Aufteilung	MFCCs 1-5 [Bit 1-5]	MFCCs 6-10 [Bit 6-10]	Hauptmultiplikator (Dez.) —
Experiment 1 und 101	00000	11111	1
Experiment 2 und 102	11111	00000	1
Experiment 3 und 103	11111	11111	1

In der Tabelle 6.4 wurde die MFCC-Bitmap in 3 Teile aufgeteilt. Diese Aufteilung ergibt insgesamt $((2^3) - 1)$ Kombinationen. Die Experimente 3 und 10 und Experiment 101 und 110 sind redundant. Das ist so gewollt, weil damit nebenbei noch eine zusätzliche Kontrolle des Klassifikationsverfahrens eingeführt wurde. Durch den Vergleich von beiden Experimenten kann man nachvollziehen, ob der Klassifikationsprozess unter gleichen Bedingungen immer identische Ergebnisse liefert.

Tabelle 6.4.: Bitmap Aufteilung in 3 Gruppen

MFCC Bitmap Gruppen Bitmap Aufteilung	MFCCs 1-3 [Bit 1-3]	MFCCs 4-7 [Bit 4-7]	MFCCs 8-10 [Bit 8-10]	Hauptmultiplikator (Dez.) —
Experiment 4 und 104	000	0000	111	1
Experiment 5 und 105	000	1111	000	1
Experiment 6 und 106	000	1111	111	1
Experiment 7 und 107	111	0000	000	1
Experiment 8 und 108	111	0000	111	1
Experiment 9 und 109	111	1111	000	1
Experiment 10 und 110	111	1111	111	1

Tabelle 6.5.: Überblick über die Anwendung der Hauptmultiplikatoren

MFCC Bitmap Gruppen Bitmap Aufteilung	MFCCs 1-3 [Bit 1-3]	MFCCs 4-7 [Bit 4-7]	MFCCs 8-10 [Bit 8-10]	Hauptmultiplikator (Dez.) —
Experiment 11 und 111	111	1111	111	10
Experiment 12 und 112	111	1111	111	100
Experiment 13 und 113	111	1111	111	1000
Experiment 14 und 114	111	1111	111	10000
Experiment 15 und 115	111	1111	111	100000

6.3. Ergebnisse

Im Vergleich beider Gruppen von Experimenten kann man sagen, dass die Experimente 1-MFCC bis 15-MFCC (Experimente mit angepassten Skalierungsfaktoren) keine Verbesserung von Erkennungsraten gebracht haben. Die Ergebnisse liegen durchgehend für alle 15 Experimente zwischen 3 und 10 Prozentpunkten niedriger als bei den Experimenten 101-MFCC bis 115-MFCC (Experimente ohne Skalierungsfaktoren). Besonders nachteilig hat sich diese Anpassung auf die Experimente 4-MFCC und 7-MFCC ausgewirkt, deren Erkennungsrate um 20 bzw. 17 Prozentpunkte niedriger ausfiel.

An dieser Stelle kann man sagen, dass die Verwendung von Hauptmultiplikatoren keine Verbesserung der Erkennungsrate gebracht hat. Das kann man sehen, indem man die Ergebnisse von Experiment 10-MFCC/110-MFCC mit Ergebnissen von den Experimenten 11-MFCC/111-MFCC bis 15-MFCC/115-MFCC vergleicht. Diese Ergebnisse sind bis auf eine Ausnahme identisch.

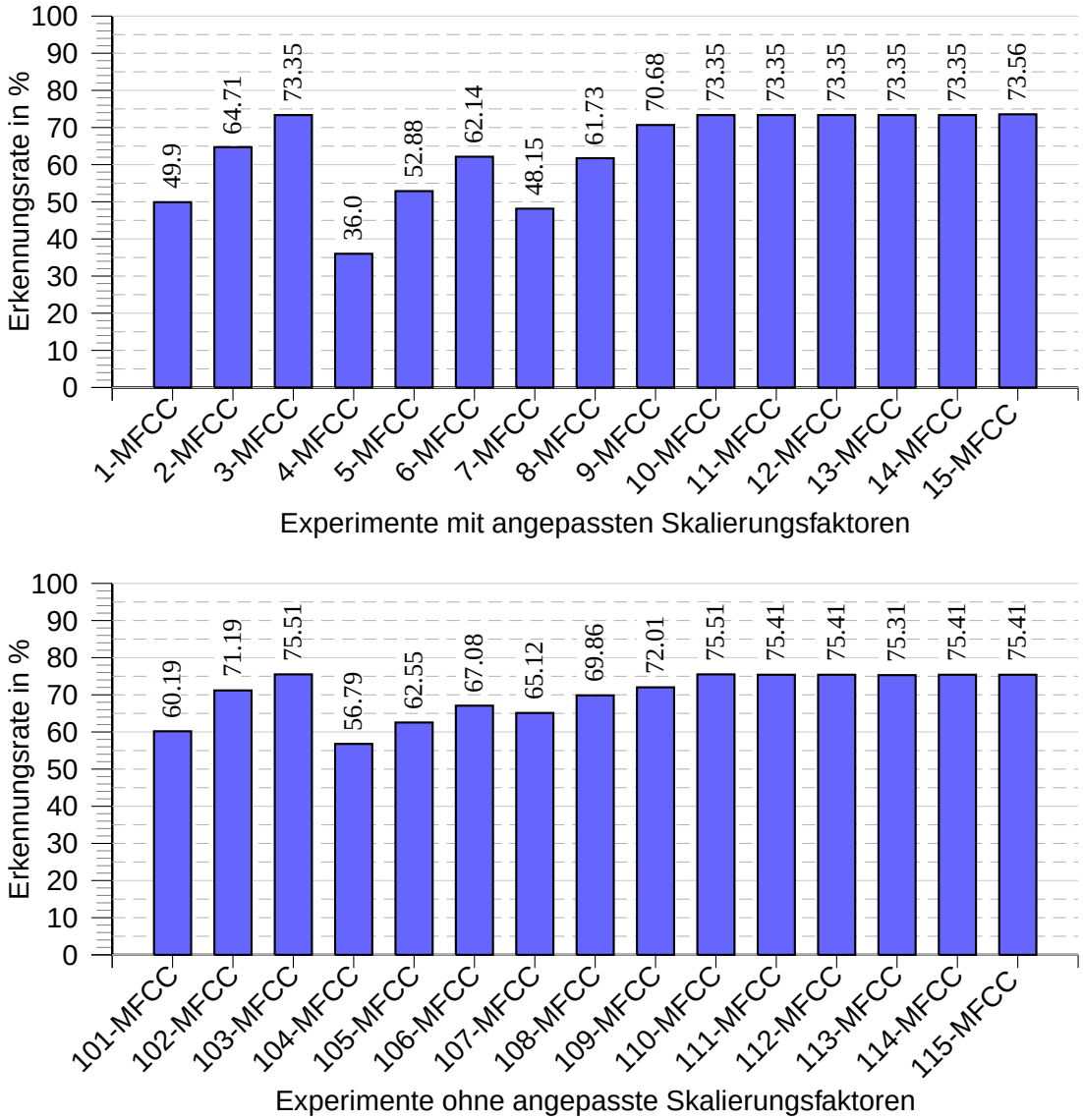
6.3.1. Erkennungsrate mit angepassten Skalierungsfaktoren

Tabelle 6.6.: Einstellungen für die Experimente 1-MFCC bis 15-MFCC

	Training		**Test**	
Experimente Name	MFCC Bitmap [Bit 1-10]	Sounds [Anzahl]	MFCC Bitmap [Bit 1-10]	Sounds [Anzahl]
1-MFCC	0000011111	978	0000011111	972
2-MFCC	1111100000	978	1111100000	972
3-MFCC	1111111111	978	1111111111	972
4-MFCC	0000000111	978	0000000111	972
5-MFCC	0001111000	978	0001111000	972
6-MFCC	0001111111	978	0001111111	972
7-MFCC	1110000000	978	1110000000	972
8-MFCC	1110000111	978	1110000111	972
9-MFCC	1111111000	978	1111111000	972
10-MFCC	1111111111	978	1111111111	972
11-MFCC	1111111111	978	1111111111	972
12-MFCC	1111111111	978	1111111111	972
13-MFCC	1111111111	978	1111111111	972
14-MFCC	1111111111	978	1111111111	972
15-MFCC	1111111111	978	1111111111	972

Abbildung 6.1.: Ergebnisse für die Experimente zu MFCCs

● Ziel: Ermittlung des Einflusses der Skalierungsfaktoren auf die Erkennungsrate. Ermittlung, welche der 10 MFCCs besonders wichtig für den Klassifikationsprozess sind.
● Beschreibung: In dem oberen Teil der Abbildung wurden in den Experimenten angepasste Skalierungsfaktoren benutzt. In dem unteren Teil wurden die angepassten Skalierungsfaktoren ausgestellt (es wurden “Rohdaten“ verwendet).
● Ergebnis: Die Experimente ohne Skalierungsfaktoren haben durchgehend bessere Ergebnisse geliefert. Besonders relevante Experimente sind 102-MFCC und 109-MFCC.



6.3.2. Erkennungsrate ohne angepasste Skalierungsfaktoren

Tabelle 6.7.: Einstellungen für die Experimente 101-MFCC bis 115-MFCC

	Training		**Test**	
Experimente Name	MFCC Bitmap [Bit 1-10]	Sounds [Anzahl]	MFCC Bitmap [Bit 1-10]	Sounds [Anzahl]
101-MFCC	0000011111	978	0000011111	972
102-MFCC	1111100000	978	1111100000	972
103-MFCC	1111111111	978	1111111111	972
104-MFCC	0000000111	978	0000000111	972
105-MFCC	0001111000	978	0001111000	972
106-MFCC	0001111111	978	0001111111	972
107-MFCC	1110000000	978	1110000000	972
108-MFCC	1110000111	978	1110000111	972
109-MFCC	1111111000	978	1111111000	972
110-MFCC	1111111111	978	1111111111	972
111-MFCC	1111111111	978	1111111111	972
112-MFCC	1111111111	978	1111111111	972
113-MFCC	1111111111	978	1111111111	972
114-MFCC	1111111111	978	1111111111	972
115-MFCC	1111111111	978	1111111111	972

Die beste Erkennungsrate von 75,51 % wurde in den Experimenten 103-MFCC bzw. 110-MFCC erzielt. In diesen Experimenten wurden alle 10 MFCCs zur Klassifikation eingesetzt. Eine gute Erkennungsrate wurde in den Experimenten 109-MFCC (72,02 %), 102-MFCC (71,19 %) und 107-MFCC (65,12 %) erzielt. Dabei konnte in Experiment 109-MFCC der Audiovektor um 30 % verkleinert werden, in 102-MFCC um 50 % und in 107-MFCC wurde der Audiovektor sogar um 70 % verkleinert.

Aus den Ergebnissen ist deutlich zu erkennen, dass die MFCCs 1 bis 5 mehr Gewichtung im Klassifizierungsprozess haben, als die mit höheren Nummern (MFCC 6 bis 10).

7. Experimente zum zeitlichen Verlauf der Harmonischen

Alle Experimente, ihre Durchführung, Logik und die Vorgänge, die dahinter stecken, sind ähnlich bzw. weitgehend identisch mit den im Kapitel 6 (Experimente zu Mel Frequency Cepstral Coefficients) beschriebenen Vorgängen. Ich werde mich deshalb auf einiges aus dem Kapitel 6 beziehen und es an dieser Stelle nicht noch einmal komplett beschreiben.

In diesem Kapitel wird der Einfluss der einzelnen Harmonischen auf die Ergebnisse der Klassifizierung der Audiosignale untersucht. Zu erwarten ist, dass auch einzelne Harmonische unterschiedlich stark die Klassifizierungsergebnisse beeinflussen, und dass man somit auf einige der 16 Harmonischen im Klassifizierungsprozess verzichten könnte, ohne dabei die Erkennungsrate zu verschlechtern.

7.1. Überblick

Hier sind folgende zwei Gruppen von Experimenten zu unterscheiden. Schon im Kapitel 6 war klar, dass die Einführung von Hauptmultiplikatoren nicht sehr erfolgversprechend war, trotzdem lasse ich auch hier, konsequenterweise, die Experimente komplett wie im Kapitel 6 laufen.

7.1.1. Experimente mit angepassten Skalierungsfaktoren

Es wurden insgesamt 15 Experimente durchgeführt. Die Experimente 1 - 10 haben Hauptmultiplikator “1“. Die Experimente 11 - 15 bekamen die Hauptmultiplikatoren, die aus der Tabelle 7.5 zu entnehmen sind.

In der Tabelle 7.1 sind Skalierungsfaktoren für für Harmonische 1 bis Harmonische 16 zu sehen. Alle Experimente “**mit** angepassten Skalierungsfaktoren“ wurden mit diesen Einstellungen durchgeführt.

Tabelle 7.1.: Überblick über die angepassten Skalierungsfaktoren der Harmonischen

Zeitlicher Verlauf der Harmonischen (1-8)								
Nummer	1	2	3	4	5	6	7	8
Skalierungsfaktor	1	1	1	10	10	10	10	100
Zeitlicher Verlauf der Harmonischen (9-16)								
Nummer	9	10	11	12	13	14	15	16
Skalierungsfaktor	100	100	100	1000	1000	1000	1000	1000

7.1.2. Experimente ohne angepasste Skalierungsfaktoren

Insgesamt wurden 15 Experimente durchgeführt. Die Experimente 101 - 110 haben Hauptmultiplikator “1“ und die Experimente 111 - 115 bekamen die Hauptmultiplikatoren, die aus der Tabelle 7.5 zu entnehmen sind.

7.2. Vorbereitung der Experimente

In der Tabelle 7.2 sind die Bitmaps zu sehen, deren Einstellungen gültig sind für alle Experimente, die in diesem Kapitel beschrieben werden. Hier werden alle verfügbaren Audiovektoren ausgewählt.

Tabelle 7.2.: Bitmaps zur Auswahl von extrahierten Audiodateien

Training						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
1	0	10	11110000000000000000	111	111111	100011001100
Test						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
0	1	10	11110000000000000000	111	111111	100011001100

Der Harmonischen-Merkmalsvektor beinhaltet insgesamt 16 Harmonische. Jede Harmonische hat je 40 Zahlenwerte, das ergibt insgesamt 640 Zahlenwerte pro Audiosignal. Ob und welche Harmonischen im Klassifizierungsprozess berücksichtigt werden, wird in der Harmonischen-Bitmap festgehalten, die zusammen mit anderen konfigurierbaren Parametern in der Konfigurationsdatei für die STT-Phase abgespeichert wurde.

Die Tabelle 7.3 zeigt die Aufteilung der Harmonischen-Bitmap in 2 Teile. Das ergibt $((2^2)-1)$ Kombinationen, die in 3 Experimenten berechnet wurden.

Tabelle 7.3.: Bitmap Aufteilung in 2 Gruppen

Bitmap Gruppen Bitmap Aufteilung	Harmonische 1-8 [Bit 1-8]	Harmonische 9-16 [Bit 9-16]	Hauptmultiplikator [Dezimal]
Experiment 1 und 101	00000000	11111111	1
Experiment 2 und 102	11111111	00000000	1
Experiment 3 und 103	11111111	11111111	1

In der Tabelle 7.4 wurde die Harmonischen-Bitmap in 3 Teile aufgeteilt. Diese Aufteilung ergibt insgesamt $((2^3) - 1)$ Kombinationen. Die “Experimente 3 und 10“ und “Experiment 101 und 110“ sind mit der gleichen Begründung wie im Kapitel 6 redundant.

Tabelle 7.4.: Bitmap Aufteilung in 3 Gruppen

Bitmap Gruppen Bitmap Aufteilung	Harmonische 1-5 [Bit 1-5]	Harmonische 6-11 [Bit 6-11]	Harmonische 12-16 [Bit 12-16]	Hauptmultiplikator [Dezimal]
Experiment 4 und 104	00000	000000	11111	1
Experiment 5 und 105	00000	111111	00000	1
Experiment 6 und 106	00000	111111	11111	1
Experiment 7 und 107	11111	000000	00000	1
Experiment 8 und 108	11111	000000	11111	1
Experiment 9 und 109	11111	111111	00000	1
Experiment 10 und 110	11111	111111	11111	1

Tabelle 7.5.: Überblick über die Anwendung der Hauptmultiplikatoren

Bitmap Gruppen Bitmap Aufteilung	Harmonische 1-5 [Bit 1-5]	Harmonische 6-11 [Bit 6-11]	Harmonische 12-16 [Bit 12-16]	Hauptmultiplikator [Dezimal]
Experiment 11 und 111	11111	111111	11111	10
Experiment 12 und 112	11111	111111	11111	100
Experiment 13 und 113	11111	111111	11111	1000
Experiment 14 und 114	11111	111111	11111	10000
Experiment 15 und 115	11111	111111	11111	100000

7.3. Ergebnisse

Die Analyse der Ergebnisse hat die Annahme bestätigt, dass der Einsatz von angepassten Skalierungsfaktoren und Hauptmultiplikatoren im Klassifikationsprozess keine Verbesserungen von Erkennungsraten gebracht hat. Die Ergebnisse, die die Experimente

101-Temp bis 110-Temp geliefert haben, sind durchgehend besser als die Ergebnisse (Siehe Abbildung 7.1) mit angepassten Skalierungsfaktoren.

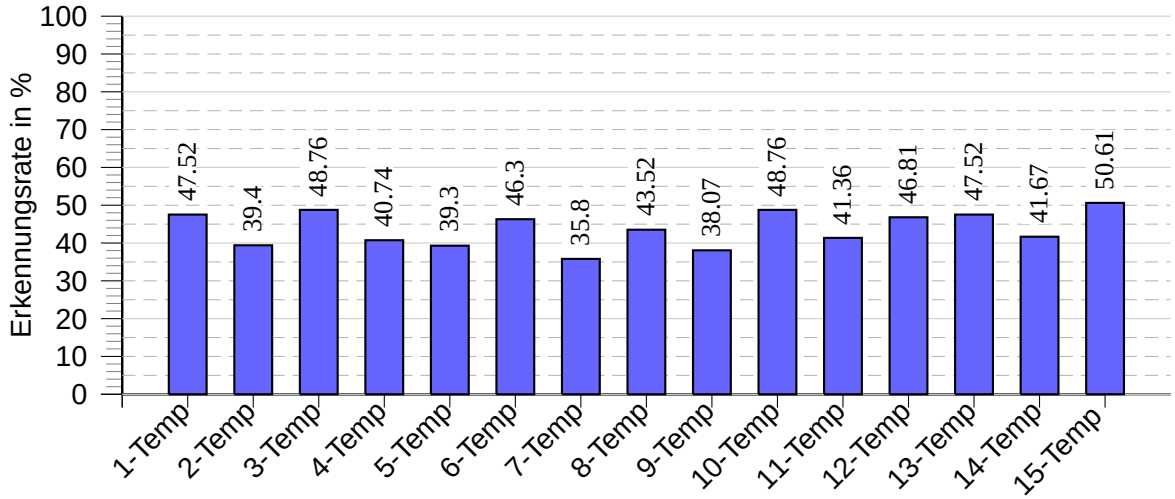
7.3.1. Erkennungsrate mit angepassten Skalierungsfaktoren

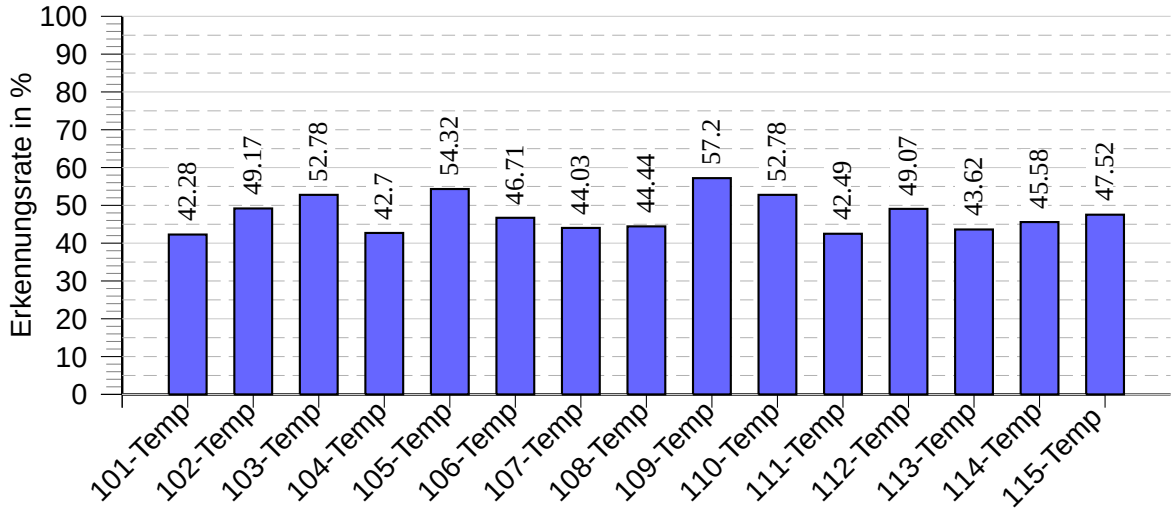
Tabelle 7.6.: Bitmap-Einstellungen für die Experimente 1-Temp bis 15-Temp

	Training		**Test**	
Experimente Name	Harmonischen-Bitmap [Bit 1-16]	Sounds [Anzahl]	Harmonischen-Bitmap [Bit 1-16]	Sounds [Anzahl]
1-Temp	0000000011111111	978	0000000011111111	972
2-Temp	1111111100000000	978	1111111100000000	972
3-Temp	1111111111111111	978	1111111111111111	972
4-Temp	0000000000011111	978	0000000000011111	972
5-Temp	0000011111100000	978	0000011111100000	972
6-Temp	0000011111111111	978	0000011111111111	972
7-Temp	1111100000000000	978	1111100000000000	972
8-Temp	1111100000011111	978	1111100000011111	972
9-Temp	1111111111100000	978	1111111111100000	972
10-Temp	1111111111111111	978	1111111111111111	972
11-Temp	1111111111111111	978	1111111111111111	972
12-Temp	1111111111111111	978	1111111111111111	972
13-Temp	1111111111111111	978	1111111111111111	972
14-Temp	1111111111111111	978	1111111111111111	972
15-Temp	1111111111111111	978	1111111111111111	972

Abbildung 7.1.: Ergebnisse für die Experimente zum zeitlichen Verlauf der Harmonischen

• Ziel: Ermittlung des Einflusses der Skalierungsfaktoren auf die Erkennungsrate. Ermittlung, welche der 16 Harmonischen besonders wichtig für den Klassifikationsprozess sind.
• Beschreibung: In dem oberen Teil der Abbildung wurden in den Experimenten angepasste Skalierungsfaktoren benutzt. In dem unteren Teil wurden die angepassten Skalierungsfaktoren ausgestellt (es wurden “Rohdaten“ verwendet).
• Ergebnis: Die Experimente ohne angepasste Skalierungsfaktoren haben in den relevanten Experimenten bessere Ergebnisse geliefert. Besonders relevante Experimente sind 105-Temp und 109-Temp.





7.3.2. Erkennungsrate ohne angepasste Skalierungsfaktoren

Die beste Erkennungsrate wurde nicht, wie man erwarten würde, in den Experimenten 103-Temp, bzw. 110-Temp erzielt. In diesen Experimenten wurden alle 16 Harmonischen zur Klassifikation eingesetzt, deshalb würde man hier auch beste Ergebnisse vermuten. Beste Ergebnisse haben jedoch die Experimente 109-Temp (57,20 %) und 105-Temp (54,32 %) geliefert, die im Vergleich mit den Experimenten 103-Temp/110-Temp nur 52,78 % erreicht hatten.

Tabelle 7.7.: Einstellungen für die Experimente 101-Temp bis 115-Temp

	Training		**Test**	
Experimente Name	Harmonischen-Bitmap [Bit 1-16]	Sounds [Anzahl]	Harmonischen-Bitmap [Bit 1-16]	Sounds [Anzahl]
101-Temp	0000000011111111	978	0000000011111111	972
102-Temp	1111111100000000	978	1111111100000000	972
103-Temp	1111111111111111	978	1111111111111111	972
104-Temp	0000000000011111	978	0000000000011111	972
105-Temp	0000011111100000	978	0000011111100000	972
106-Temp	0000011111111111	978	0000011111111111	972
107-Temp	1111100000000000	978	1111100000000000	972
108-Temp	1111100000011111	978	1111100000011111	972
109-Temp	1111111111100000	978	1111111111100000	972
110-Temp	1111111111111111	978	1111111111111111	972
111-Temp	1111111111111111	978	1111111111111111	972
112-Temp	1111111111111111	978	1111111111111111	972
113-Temp	1111111111111111	978	1111111111111111	972
114-Temp	1111111111111111	978	1111111111111111	972
115-Temp	1111111111111111	978	1111111111111111	972

Man kann beobachten, dass die Harmonischen 6 - 11 besonders wichtig sind und besonders gute Ergebnisse liefern. Zu sehen ist das besonders gut in den Experimenten 109-Temp (57,20 %), 105-Temp (54,32 %) (Siehe Tabelle 7.7 und Abbildung 7.1.).

Dagegen kann man sagen, dass die Harmonischen 12 - 16 nicht nur unwichtig für die Klassifikation sind, sondern dass sie sogar negativen Einfluss auf die Ergebnisse haben. Zu erkennen ist dieses Verhalten, wenn man die Experimente 105-Temp und 106-Temp vergleicht. In Experiment 105-Temp wurden nur die Harmonischen 6 - 11 benutzt. Das hat das zweitbeste Ergebnis von 54,32 % gebracht. In Experiment 106-Temp sind zusätzlich die Harmonischen 12 - 16 dazu genommen worden. Das Ergebnis hat sich um knapp 10 Prozentpunkte verschlechtert und beträgt nur noch 46,71 %.

Die Harmonischen 1 - 5 beeinflussen die Ergebnisse gering positiv. Um das zu verdeutlichen, nehme ich als Beispiel die Experimente 105-Temp und 109-Temp. In Experiment

105-Temp wurden nur die Harmonischen 6 - 11 benutzt und das hat das zweitbeste Ergebnis von 54,32 % gebracht. In Experiment 109-Temp sind zusätzlich die Harmonischen 1 - 5 dazu genommen worden. Das Ergebnis hat sich um knapp 3 Prozentpunkte verbessert und beträgt 57,20 %.

8. Experimente zur Reduktion der Daten in Merkmalsvektoren

Die in den Kapiteln 6 und 7 gewonnenen Erkenntnisse wurden in diesem Kapitel eingesetzt, um eine Reduktion von Daten in den Merkmalsvektoren zu erreichen. Aus den beiden o.g. Kapiteln wurden jeweils zwei Experimente ausgewählt, die gute Ergebnisse geliefert hatten, bei einer minimalen Größe von Merkmalsvektoren.

8.1. Vorbereitung der Experimente

Ausgewählt wurden die Experimente 105-Temp und 109-Temp, die die besten Klassifikationsergebnisse bei den Harmonischen aufwiesen und entsprechend bei den MFCCs die Experinente 102-MFCC und 109-MFCC, die hier die besten Ergebnisse brachten. Von nun an wurden alle drei Merkmalstypen (Daten des nontonalen Spektrums, MFCCs und Harmonische) in die Merkmalsektoren aufgenommen.

Die Bitmaps-Einstellungen (Siehe Tabelle 8.2.) für die Auswahl der Daten in den Merkmalsvektoren der ausgewählten Experimente wurden miteinander kombiniert und so sind vier weitere Experimente entstanden. Die Aufgabe der Experimente ist zu prüfen, ob die Erkennungsrate unter diesen Bedingungen noch immer zufriedenstellend ist.

Alle Experimente, die in diesem Kapitel durchgeführt wurden, sind mit den Einstellungen zur Auswahl der Audiosignale durchgeführt worden, die in der Tabelle 8.1 zu finden sind. Wie man aus der Tabelle entnehmen kann, werden an dieser Stelle alle verfügbaren Audiosignale verwendet.

8.2. Ergebnisse

Aus der Abbildung 8.1 und der Tabelle 8.2 kann man erkennen, dass sich die Erkennungsrate mit der Verkleinerung des Merkmalsvektors verschlechtert. Das Experiment

Tabelle 8.1.: Bitmaps zur Auswahl von Audiosignalen für die TNM-Experimente

Training								
Experiment Name	Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler* [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]	Sounds [Anzahl]
01-TNM	1	0	10	111100	111	111111	100011001100	978
02-TNM	1	0	10	111100	111	111111	100011001100	978
03-TNM	1	0	10	111100	111	111111	100011001100	978
04-TNM	1	0	10	111100	111	111111	100011001100	978
Test								
Experiment Name	Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler* [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]	Sounds [Anzahl]
01-TNM	0	1	10	111100	111	111111	100011001100	972
02-TNM	0	1	10	111100	111	111111	100011001100	972
03-TNM	0	1	10	111100	111	111111	100011001100	972
04-TNM	0	1	10	111100	111	111111	100011001100	972

*(Nur die Bits 1-6 sind angezeigt worden. Die restlichen 14 Bits haben Wert “0“.)

05-TNM ist mit der vollen Länge des Merkmalsvektors durchgefügrt worden und dient nur als eine Referenz, um die Ergebnisse der Experimente miteinander vergleichen zu können.

Die Referenz-Erkennungsrate (81,58 %) ist im Experiment 01-TNM [8] um nur knapp 4 Prozentpunkte unterschritten worden und beträgt 77,67 %. Gleichzeitig ist der Merkmalsvektor deutlich kleiner geworden und seine Größe beträgt nur noch 43,72 % der Größe des Referenz-Merkmalsvektors. Der Verlauf der beiden Wertereihen (Erkennungsrate und die Größe der Merkmalsvektoren) ist relativ linear. Würde man diese Wertereihen wie eine Kurve darstellen, so würde die Kurve, die die Erkennungsrate darstellt, eine leicht fallende Tendenz zeigen. Dagegen würde die Kurve, die die Größe der Merkmalsvektoren darstellt, einen stark fallenden Verlauf annehmen.

Interessante Ergebnisse haben die Experimente 01-TNM und 02-TNM geliefert. Die beiden Experimente haben eine identische Erkennungsrate (77,67 %) gebracht, obwohl die Merkmalsvektoren eine verschiedene Größe (01-TNM (505 / 43,72 %) bzw. 02-TNM (605 / 52,38 %)) aufweisen. Diese Ergebnisse bestätigen die Richtigkeit der Annahme, dass die MFCCs 1 - 5 wichtiger für den Klassifikationsprozess sind als die MFCCs 6 - 10. Deshalb haben die MFCCs 6 - 7, die ich im Experiment 02-TNM zusätzlich benutzt habe, keinen Einfluss auf die Erkennungsrate gehabt.

[8] TNM ist ein Akronym und steht für die Begriffe Temporal (zeitlicher Verlauf der Harmonischen), Nontonal und MFCC. Das Akronym soll assoziieren, dass alle drei Merkmalstypen in diesen Experimenten verwendet werden.

Abbildung 8.1.: Ergebnisse für die Experimente 1-TNM bis 5-TNM

● Ziel: Durchführung der Experimente mit reduzierter Anzahl der Werte in den Merkmalsvektoren.
● Beschreibung: Die MFCC- (Abbildung 6.1) und Temp-Experimente (Abbildung 7.1), die als besonders relevant eingestuft wurden, werden kombiniert, um gute Erkennungsraten bei minimaler Größe der Merkmalsvektoren zu erreichen.
● Ergebnis: Experiment 01-TNM (gute Erkennungsrate bei kleiner Merkmalsvektor-Größe) und 05-TNM (wird als Referenz-Experiment benutzt) sind besonders relevant.

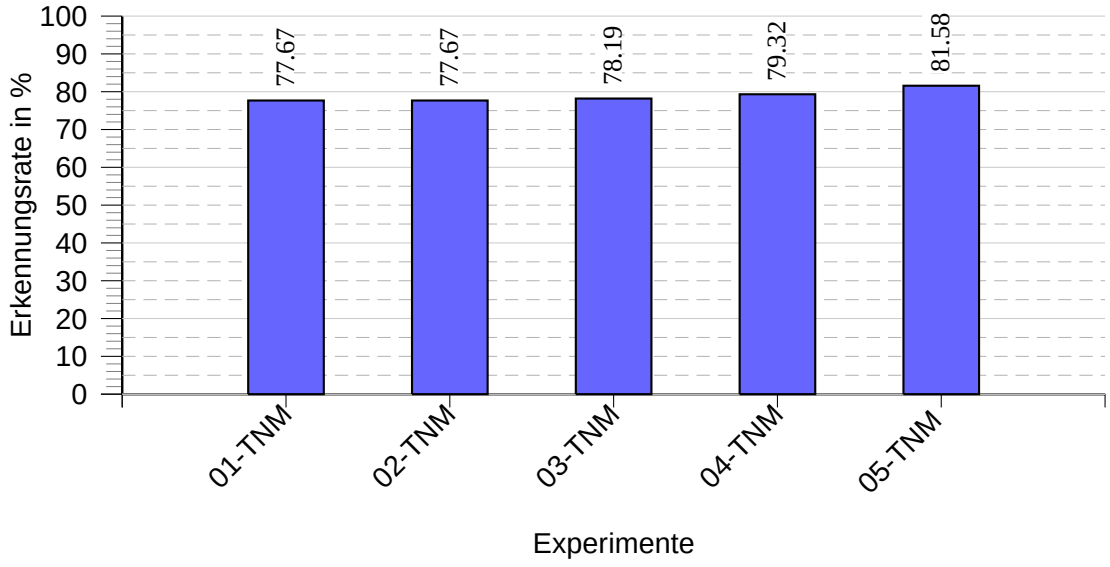


Tabelle 8.2.: Bitmaps zur Auswahl von Daten in den Merkmalsvektoren

Training					
Experimente Name	Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Harmonische [Bit 1-16]	Vektorgröße [Länge]*	[%]**
01-TNM	111111111111111	1111100000	0000011111100000	505	43.72
02-TNM	111111111111111	1111111000	0000011111100000	605	52.38
03-TNM	111111111111111	1111100000	1111111111100000	705	61.03
04-TNM	111111111111111	1111111000	1111111111100000	805	69.69
05-TNM	111111111111111	1111111111	1111111111111111	1155	100.00
Test					
Experimente Name	Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Harmonische [Bit 1-16]	Vektorgröße [Länge]*	[%]**
01-TNM	111111111111111	1111100000	0000011111100000	505	43.72
02-TNM	111111111111111	1111111000	0000011111100000	605	52.38
03-TNM	111111111111111	1111100000	1111111111100000	705	61.03
04-TNM	111111111111111	1111111000	1111111111100000	805	69.69
05-TNM	111111111111111	1111111111	1111111111111111	1155	100.00

*(Die Anzahl der Werte im Merkmalsvektor)

**(Die Größe des Merkmalsvektors im % bezogen auf die Größe des vollen Merkmalsvektors)

9. Experimente zur Reduktion der Audiosignale

Die Experimente, die in den Kapiteln 6, 7 und 8 beschrieben wurden, hatten als Ziel, die Daten in den Merkmalsvektoren zu reduzieren und dabei die gute Erkennungsrate beizubehalten. In diesem Kapitel geht es dagegen um die Reduktion der Anzahl von Audiosignalen bei gleich bleibender Größe der Merkmalsvektoren. Es sind Experimente zu allen Kodierungs-Kategorien der Audiosignale (Spieler, Spielart, Saite und Bund) durchgeführt worden. Das Ziel war, die Audiosignale nach Untermengen der einzelnen Kodierungs-Kategorien zu selektieren und den Einfluss auf die Erkennungsrate zu ermitteln.

9.1. Vorbereitung der Experimente

Die in dem Kapitel 8 ermittelten Bitmaps zur Auswahl der Daten in den Merkmalsvektoren wurden an dieser Stelle verwendet. Siehe die Bitmaps-Einstellungen in der Tabelle 9.1. In der Trainings- und Test-Phase wurden identische Einstellungen der Bitmaps verwendet. Diese Einstellungen sind gültig für alle Experimente in diesem Kapitel. Bei den Ausnahmen wurden die abweichenden Einstellungen explizit angegeben.

Tabelle 9.1.: Bitmaps zur Auswahl der Daten in den extrahierten Merkmalsvektoren

Training		
Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Zeitlicher Verlauf der Harmonischen [Bit 1-16]
111111111111111	1111100000	0000011111100000
Test		
Nontonales Spektrum [Bit 1-15]	MFCCs [Bit 1-10]	Zeitlicher Verlauf der Harmonischen [Bit 1-16]
111111111111111	1111100000	0000011111100000

In der Tabelle 9.2 sind die Kodierungs-Kategorien aufgelistet worden. Für dieses Kapitel galt, dass nur die Bitmap angepasst wurde, mit deren Kodierungs-Kategorie aktuell

experimentiert wurde. Alle anderen Bitmaps blieben unverändert. In der Tabelle wurde beispielsweise die Bitmap der Kodierungs-Kategorie “Saite“ mit “xxxxxx“ markiert. Das heißt, dass bei den Experimenten zu den Saiten (Siehe Abschnitt 9.4.) nur diese Bitmap angepasst wurde. Bei den Ausnahmen wurde auf die abweichende Einstellungen explizit hingewiesen.

Tabelle 9.2.: Bitmaps zur Auswahl von extrahierten Audiodateien

Training						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
1	0	10	11110000000000000000	111	xxxxxx	100011001100
Test						
Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]
0	1	10	11110000000000000000	111	xxxxxx	100011001100

9.2. Ergebnisse der Experimente zu den Spielern

Es sind insgesamt 28 Experimente zu den Spielern durchgeführt worden. Die Ergebnisse der Experimente sind in der Abbildung 9.1 dargestellt worden. Die Bitmaps zur Auswahl der Spieler sind in der Tabelle 9.3 zu sehen.

- 1. Gruppe (01-Spieler bis 04-Spieler)
  In den Experimenten 01-Spieler bis 04-Spieler wurde jeweils mit dem gleichen Spieler trainiert und getestet. Die Erkennungsrate variierte zwischen 74,07 % im Experiment 04-Spieler bis 88,58 % im Experiment 03-Spieler. Durchschnittlich ergibt das eine Erkennungsrate von 79,79 %.

- 2. Gruppe (05-Spieler bis 16-Spieler)
  In der zweiten Gruppe von Experimenten wurde jeweils mit einem Spieler trainiert und einem anderen getestet. Durch konsequentes Fortführen dieser Vorgehensweise sind 12 Experimente entstanden. Das sind die Experimente 05-Spieler bis 16-Spieler. Die Erkennungsrate variierte zwischen 45,06 % im Experiment 10-Spieler bis 62,35 % in den Experimenten 05-Spieler und 12-Spieler. Durchschnittlich ergibt das eine Erkennungsrate von 52,18 %.

- 3. Gruppe (21-Spieler bis 24-Spieler)
  In den Experimenten 21-Spieler bis 24-Spieler ist mit drei Spielern trainiert worden und mit dem vierten Spieler wurde getestet. Durch diese Vorgehensweise sind die vier Experimente entstanden. Die Erkennungsrate in diesen Experimenten liegt zwischen 59,57 % im Experiment 21-Spieler und 69,14 % im Experiment 22-Spieler. Durchschnittlich ergibt das eine Erkennungsrate von 64,58 %.

- 4. Gruppe (25-Spieler bis 28-Spieler)
  Die Experimente 25-Spieler bis 28-Spieler sind mit den identischen Einstellungen der Auswahl-Bitmaps durchgeführt worden. Einziger Unterschied ist, dass diese Experimente mit der vollen Anzahl der Daten in den Merkmalsvektoren (1155 Werte) durchgeführt wurden. Die Erkennungsrate in diesen Experimenten ist ähnlich wie die Ergebnisse aus der Gruppe 3, sie reicht von 48,15 % im Experiment 28-Spieler bis 70,99 % im Experiment 26-Spieler. Durchschnittlich beträgt die Erkennungsrate 60,42 %.

  Wenn man die Ergebnisse aus der Gruppe 3 und Gruppe 4 vergleicht, kann man die folgende Schlüsse ziehen. Diese Ergebnisse bestätigen die Richtigkeit der Erkenntnisse[9] und Annahmen, dass die einzelnen MFCCs und Harmonischen un-

[9] Siehe in den Kapiteln 6, 7 und 8.

terschiedlich großen Einfluss auf die Erkennungsrate haben und dass sich die Nutzung mancher Daten negativ auf die Ergebnisse auswirken kann.

- 5. Gruppe (33-Spieler bis 36-Spieler)
  In den Experimenten 33-Spieler bis 36-Spieler ist mit allen vier Spielern trainiert worden. Dagegen wurde mit jeweils einem Spieler getestet. Auf diese Art sind vier Experimente zu Stande gekommen. Die Erkennungsrate liegt zwischen 71,60 % im Experiment 34-Spieler und 83,64 % im Experiment 35-Spieler. Durchschnittlich beträgt die Erkennungsrate 77,00 %.

Die Analyse der Ergebnisse zeigt, dass auch die Spieler Einfluss auf die Erkennungsrate haben. Der Einfluss der Spieler ist aber nicht sehr groß. Das erkennt man, wenn man die Experimente aus der Gruppe 1 und Gruppe 5 vergleicht. Die durchschnittliche Erkennungsrate in der Gruppe 1 betrug 79,79 %. In der Gruppe 1 wurde immer mit gleichen Gitarren trainiert und getestet. Dagegen wurde in der Gruppe 5 mit allen vier Gitarren trainiert und mit eine Gitarre getestet. Hier hat sich die durchschnittliche Erkennungsrate um knapp 3 Prozentpunkte verringert und betrug 77,00 %.

Die Ergebnisse der Experimente aus den Gruppen 2, 3 und 4 sind dagegen schlechter ausgefallen. Das liegt daran, dass in den Gruppen 3 und 4 die Trainingsmenge der Audiosignale keine Merkmalsvektoren der Gitarren beinhaltete, mit denen anschließend getestet wurde. Die schlechte durchschnittliche Erkennungsrate in der Gruppe 2 ist auch damit zu erklären, dass die Anzahl der Merkmalsvektoren in der Trainings-Phase kleiner ist als die Anzahl der Merkmalsvektoren in der Test-Phase. Das ist ungünstig, weil die Daten-Trainingsmenge[10] bei der SVM-Klassifikation größer als die Daten-Testmenge sein sollte. Das ist in der Gruppe 2 bei manchen Experimenten genau umgekehrt. Siehe die Experimente 08-Spieler, 09-Spieler, 14-Spieler und 16-Spieler in der Tabelle 9.3.

[10] Die Daten-Trainingsmenge soll um das Vielfache der Daten-Testmenge größer sein. Der Faktor zwischen 5 und 10 ist zu empfehlen.

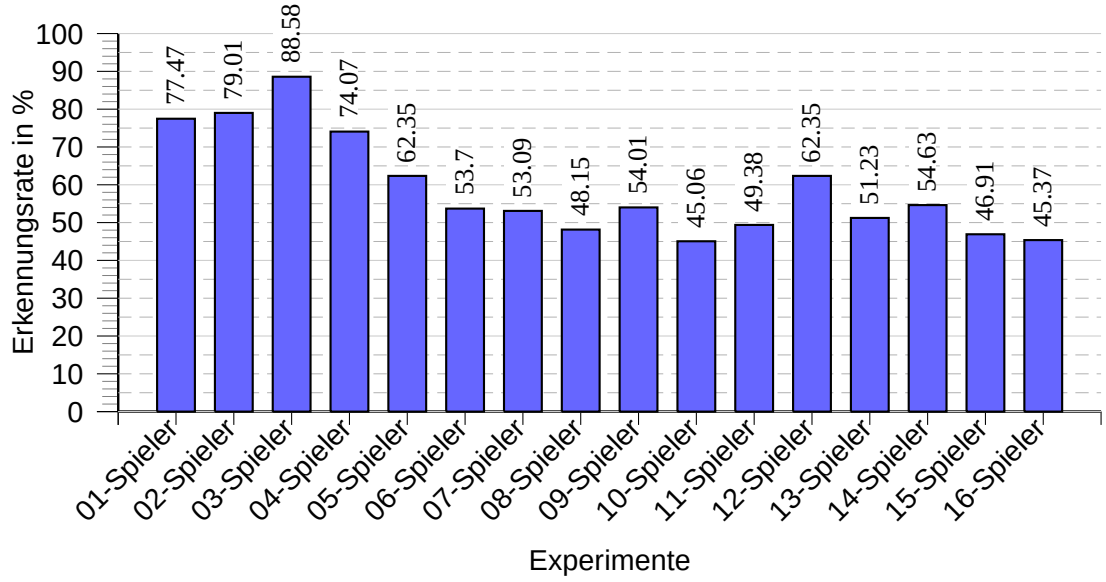
Tabelle 9.3.: Bitmaps für die Experimente zu den Spielern

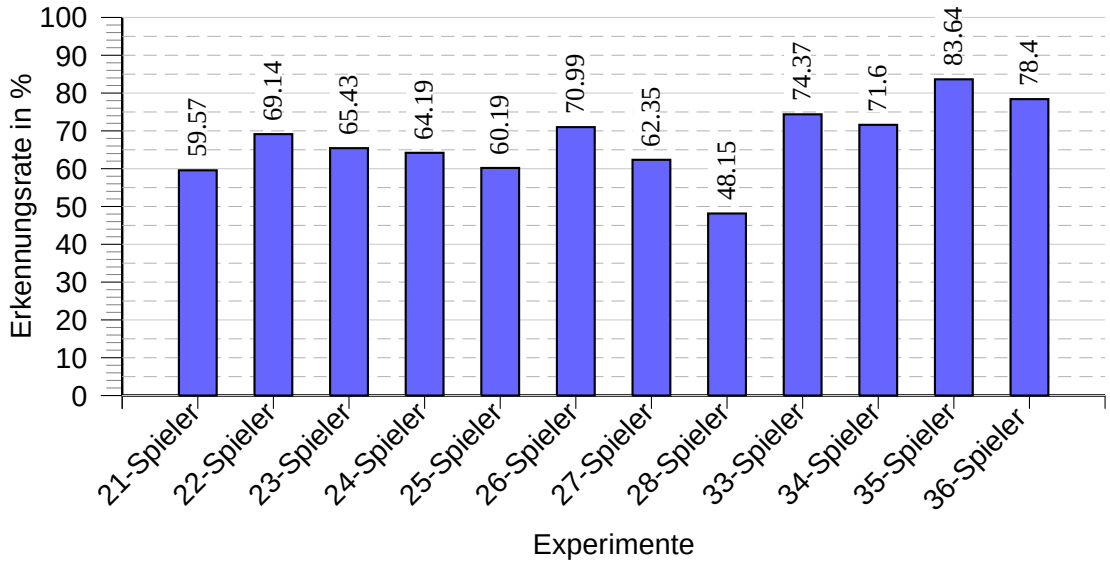
Experimente	**Training**		**Test**	
Name	Bitmap [Bit 1-20]	Sounds [Anzahl]	Bitmap [Bit 1-20]	Sounds [Anzahl]
01-Spieler	10000000000000000000	324	10000000000000000000	324
02-Spieler	01000000000000000000	162	01000000000000000000	162
03-Spieler	00100000000000000000	330	00100000000000000000	324
04-Spieler	00010000000000000000	162	00010000000000000000	162
05-Spieler	10000000000000000000	324	01000000000000000000	162
06-Spieler	10000000000000000000	324	00100000000000000000	324
07-Spieler	10000000000000000000	324	00010000000000000000	162
08-Spieler	01000000000000000000	162	10000000000000000000	324
09-Spieler	01000000000000000000	162	00100000000000000000	324
10-Spieler	01000000000000000000	162	00010000000000000000	162
11-Spieler	00100000000000000000	330	10000000000000000000	324
12-Spieler	00100000000000000000	330	01000000000000000000	162
13-Spieler	00100000000000000000	330	00010000000000000000	162
14-Spieler	00010000000000000000	162	10000000000000000000	324
15-Spieler	00010000000000000000	162	01000000000000000000	162
16-Spieler	00010000000000000000	162	00100000000000000000	324
21-Spieler	01110000000000000000	654	10000000000000000000	324
22-Spieler	10110000000000000000	816	01000000000000000000	162
23-Spieler	11010000000000000000	648	00100000000000000000	324
24-Spieler	11100000000000000000	816	00010000000000000000	162
25-Spieler*	01110000000000000000	654	10000000000000000000	324
26-Spieler*	10110000000000000000	816	01000000000000000000	162
27-Spieler*	11010000000000000000	648	00100000000000000000	324
28-Spieler*	11100000000000000000	816	00010000000000000000	162
33-Spieler	11110000000000000000	978	10000000000000000000	324
34-Spieler	11110000000000000000	978	01000000000000000000	162
35-Spieler	11110000000000000000	978	00100000000000000000	324
36-Spieler	11110000000000000000	978	00010000000000000000	162

*(Diese Experimente wurden mit der vollen Anzahl der Daten in den Merkmalsvektoren (1155 Werte) durchgeführt)

Abbildung 9.1.: Ergebnisse für die Experimente zu den Spielern

● Ziel: Einfluss der Spieler auf den Klassifikationsprozess ermitteln.
● Beschreibung: In der Tabelle 9.3 sind die Bitmaps zur Auswahl der Spieler zu sehen. Die Spieler wurden ausführlicher getestet, weil es wichtig ist zu ermitteln, ob die Spieler die Ergebnisse stark beeinflussen.
● Ergebnis: Das Klassifikationsverfahren wird nur unwesentlich von den Spielern beeinflusst.





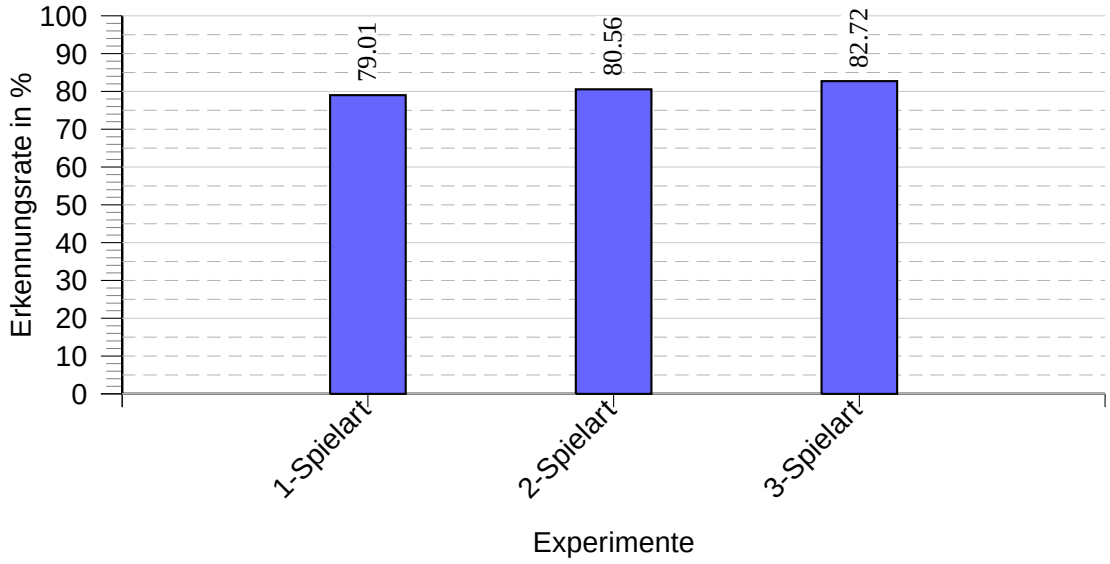
9.3. Ergebnisse der Experimente zu den Spielarten

Es sind insgesamt drei Experimente zu den Spielarten [11] durchgeführt worden. Es ist in jedem Experiment mit einer Spielart in SVM trainiert und anschließend mit der gleichen Spielart getestet worden. Die Einstellungen für die Spielart-Bitmap sind aus der Tabelle 9.4 zu entnehmen.

Die Ergebnisse (Siehe Abbildung 9.2.) für alle drei Spielarten sind gut und bewegen sich zwischen der Erkennungsrate 79.01 % (im Experiment 1-Spielart (sonor)) bis 82,72 % (im Experiment 3-Spielart (warm)). Die Töne, die mit mit der Klangvorgabe “warm“ gespielt wurden, eignen sich somit am besten für die Verwendung in dem Klassifikationsprozess.

Abbildung 9.2.: Ergebnisse für die Experimente 1-Spielart bis 3-Spielart

● Ziel: Beeinflussung des Klassifikationsprozess durch die Kategorie “Spielart“ (Klangvorgabe) ermitteln.
● Beschreibung: Drei Experimente wurden durchgeführt. Es wurde nacheinander mit derselben Spielart trainiert und getestet.
● Ergebnis: Beste Ergebnisse liefert die Spielart “warm“.



[11] Der Begriff Spielart wird synonym zum Begriff Klangvorgabe (“sonor“, “spitz“ und “warm“) verwendet.

Tabelle 9.4.: Bitmaps der Experimente zu den Spielarten

Experimente	Training		Test	
Name	Bitmap [Bit 1-3]	Sounds [Anzahl]	Bitmap [Bit 1-3]	Sounds [Anzahl]
1-Spielart	100	326	100	324
2-Spielart	010	327	010	324
3-Spielart	001	325	001	324

9.4. Ergebnisse der Experimente zu den Saiten

Es sind sechs Experimente zu den Saiten durchgeführt worden. Dabei wurde zu jeder Saite ein Experiment gemacht. Mit denselben Saiten wurde jeweils in SVM trainiert und getestet. Siehe Tabelle 9.5.

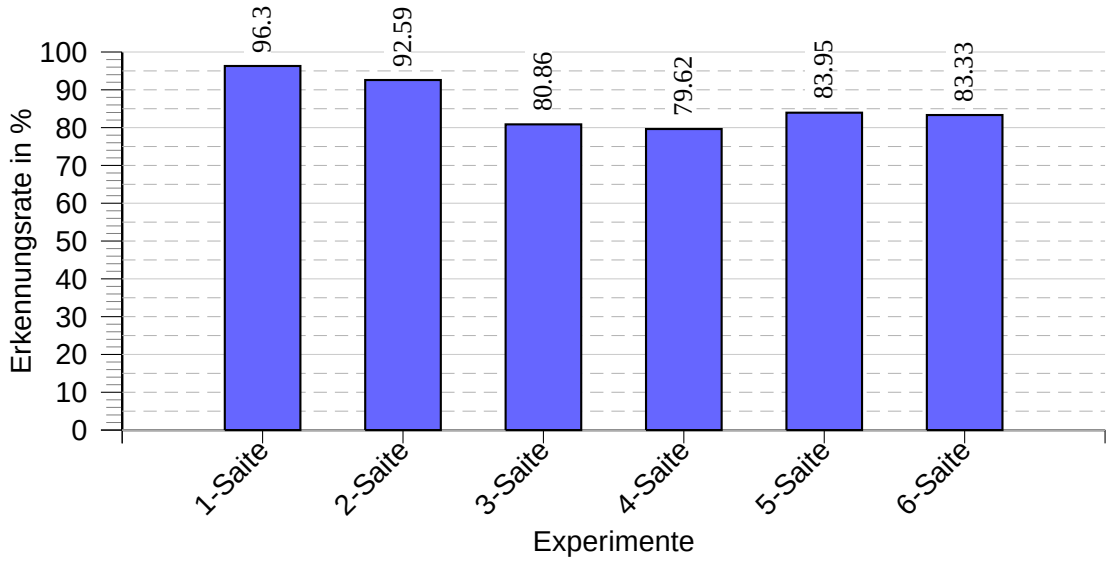
Die Ergebnisse der Klassifikation sind in der Abbildung 9.3 dargestellt worden. Aus dem Diagramm kann man entnehmen, dass die Experimente 1-Saite (96.30 %) und 2-Saite (92.59 %) sehr gute Ergebnisse geliefert haben. Die Töne, die auf den Saiten 1 und 2 (E und H) gespielt wurden, wurden vom Klassifikationsalgorithmus besonders gut bewertet. Weniger gut fallen die Wertungen zu den Tönen aus, die zur Saiten 5 und 6 (A und E) gehören (83.95 % bzw. 83.33 %) Die Ergebnisse zu den Saiten 3 und 4 sind noch etwas schlechter als die, die zu den Saiten 5 und 6 gehören sie betragen 80.86 % bzw. 79.63 %.

Tabelle 9.5.: Bitmaps für die Experimente zu den Saiten

Experimente	Training		Test	
Name	Bitmap [Bit 1-6]	Sounds [Anzahl]	Bitmap [Bit 1-6]	Sounds [Anzahl]
1-Saite	100000	165	100000	162
2-Saite	010000	163	010000	162
3-Saite	001000	162	001000	162
4-Saite	000100	163	000100	162
5-Saite	000010	163	000010	162
6-Saite	000001	162	000001	162

Abbildung 9.3.: Ergebnisse für die Experimente zu den Saiten

● Ziel: Beeinflussung des Klassifikationsprozesses durch die Kategorie “Saite“ ermitteln.
● Beschreibung: Sechs Experimente wurden durchgeführt. Es wurde nacheinander mit derselben Saite trainiert und getestet.
● Ergebnis: Beste Ergebnisse lieferten die Saiten 1 (E) und 2 (H).



9.5. Ergebnisse der Experimente zu den Bünden

Es sind insgesamt 15 Experimente zu den Bünden durchgeführt worden. Die Experimente sind in drei Gruppen aufgeteilt worden. Die Ergebnisse der Experimente und die Einstellungen der Auswahl-Bitmaps sind in der Abbildung 9.4 und in der Tabelle 9.6 zu sehen.

- 1. Gruppe (01-Bund bis 05-Bund)
  In den Experimenten 01-Bund bis 05-Bund ist mit den Tönen, die zu einem Bund gehören, trainiert worden und mit den Tönen, die zu dem gleichen Bund gehören, wurde anschließend getestet. Die Ergebnisse des Experiments 04-Bund wurden an dieser Stelle verworfen, weil sich herausgestellt hat, dass nur 12 Audiosignale zu Verfügung stehen, die zum Bund 9 gehören. Das würde nur die Ergebnisse verfälschen. Diese 12 Audiosignale sind deshalb in den weiteren Experimenten zu den Bünden nicht mehr benutzt worden. Die Erkennungsraten in diesen Experimenten sind zwischen 78,96 % im Experiment 03-Bund bis 84,88 % im Experiment 01-Bund angesiedelt. Die durchschnittliche Erkennungsrate beträgt somit 81,23 %.

- 2. Gruppe (10-Bund bis 15-Bund)
  Die Experimente 10-Bund bis 15-Bund sind entstanden durch die Auswahl von Tönen, die jeweils zu den zwei Bünden gehören, mit denen trainiert wurde. Mit den Tönen, die zu den restlichen Bünden gehören, wurde anschließend getestet. Auf diese Weise sind 6 Experimente entstanden, deren Ergebnisse zwischen 54,01 % im Experiment 14-Bund und 66,98 % im Experiment 12-Bund liegen. Die durchschnittliche Erkennungsrate beträgt 59,91 %.

- 3. Gruppe (16-Bund bis 19-Bund)
  In den letzten vier Experimenten ist jeweils mit den Tönen, die zu den drei Bünden gehören, trainiert worden und mit den Tönen, die zu dem restlichen Bund gehören, wurde getestet. Diese Experimente haben eine durchschnittliche Erkennungsrate von 62,15 % gebracht.

Bei der näheren Betrachtung der Ergebnisse dieser Experimenten-Reihe ist eindeutig, dass die Töne, die zum Bund 1 gehören, am bestens erkannt werden konnten (84,88 %), gefolgt von den Tönen, die zum Bund 10 gehören (81,73 %). Die Experimente, die in den Gruppen 2 und 3 beschrieben wurden, zeigen dass es wichtig ist, dass die Daten-Trainingsmenge mehr Merkmalsvektoren haben soll als die Daten-Testmenge. Im Vergleich der Gruppe 1 gegen die Gruppen 2 und 3 kann man feststellen, dass es sehr wichtig ist, dass in der Daten-Trainingsmenge immer die Töne vertreten sein sollen, mit

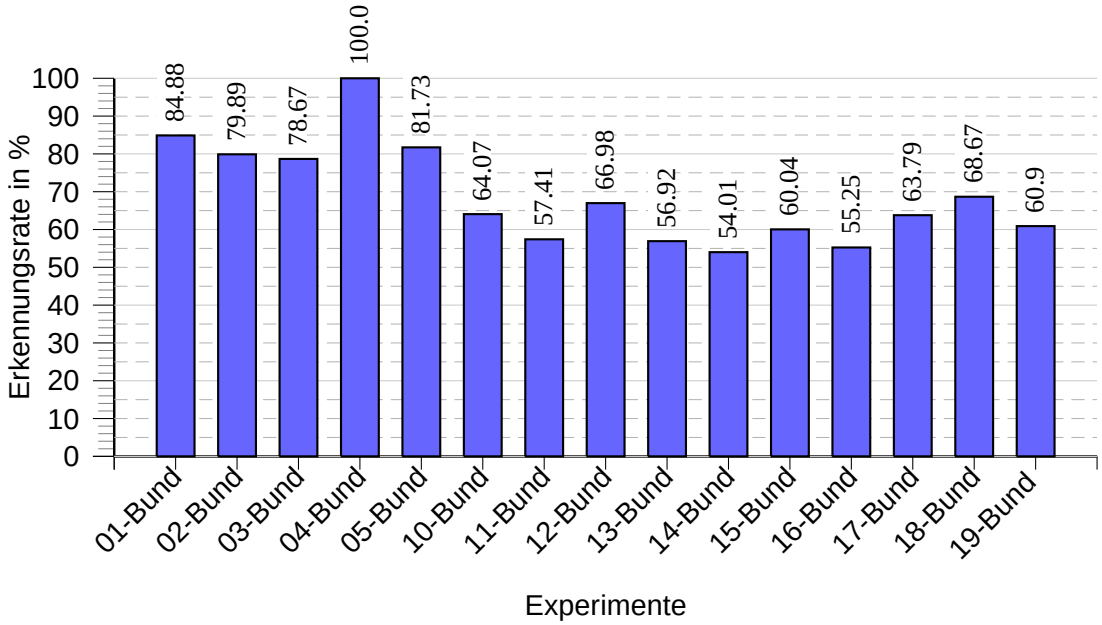
denen nacher getestet wird. Oder noch besser, man lässt immer die Töne klassifizieren wie in den Experimenten 01-Bund bis 05-Bund beschrieben wurde.

Tabelle 9.6.: Bitmaps für die Experimente zu den Bünden

Experimente	**Training**		**Test**	
Name	Bitmap [Bit 1-12]	Sounds [Anzahl]	Bitmap [Bit 1-12]	Sounds [Anzahl]
01-Bund	100000000000	326	100000000000	324
02-Bund	000010000000	176	000010000000	174
03-Bund	000001000000	150	000001000000	150
04-Bund	000000001000	12	000000001000	12
05-Bund	000000000100	314	000000000100	312
10-Bund	100010000000	502	000001000100	462
11-Bund	100001000000	476	000010000100	486
12-Bund	100000000100	640	000011000000	324
13-Bund	000011000000	326	100000000100	636
14-Bund	000010000100	490	100001000000	474
15-Bund	000001000100	464	100010000000	498
16-Bund	000011000100	640	100000000000	324
17-Bund	100001000100	790	000010000000	174
18-Bund	100010000100	816	000001000000	150
19-Bund	100011000000	652	000000000100	312

Abbildung 9.4.: Bitmaps für die Experimente zu den Bünden

● Ziel: Beeinflussung des Klassifikationsprozess durch die Kategorie “Bund“ ermitteln.
● Beschreibung: Die Tabelle 9.6 beinhaltet die Bitmaps, die in den durchgeführten Experimenten verwendet wurden.
● Ergebnis: Beste Ergebnisse lieferten das Experiment 01-Bund. Das Experiment 04-Bund kann man ignorieren. Für den Bund 9 stehen nur 12 Audiosignale zur Verfügung.



9.6. Ergebnisse für die WAV-Opt Experimente

In diesem Abschnitt sind vier Experimente durchgeführt worden, die die bisher gewonnenen Erkenntnisse überprüfen und noch einmal beweisen. Ausgewählt sind die zwei Einstellungen, die besonders gute Ergebnisse liefern sollten und als Gegenprobe zwei weitere Einstellungen, die zwei besonders schlechten Erkennungsraten liefern müssten. Besonders gut werden die Töne erkannt, die auf den Saiten 1 und 2 gespielt wurden, die mit der Klangvorgabe “warm“ eingespielt wurden und die, die zu den Bünden 1 und 10 gehören.

Die Erwartungen, die an diese Experimente gestellt wurden, sind bestätigt worden. Die Experimente 1-WAVOpt[12] und 2-WAVOpt haben sehr gute Erkennungsraten von 86,11 % bzw. 92,59 % geliefert. Die Bitmaps-Einstellungen kann man aus der Tabelle 9.7 entnehmen. Die Experimente 3-WAVOpt und 4-WAVOpt haben wie erwartet sehr schlechte Erkennungsraten geliefert und auf diese Weise die Tauglichkeit des Verfahrens noch einmal bestätigt.

Tabelle 9.7.: Bitmaps für die WAV-Opt Experimente

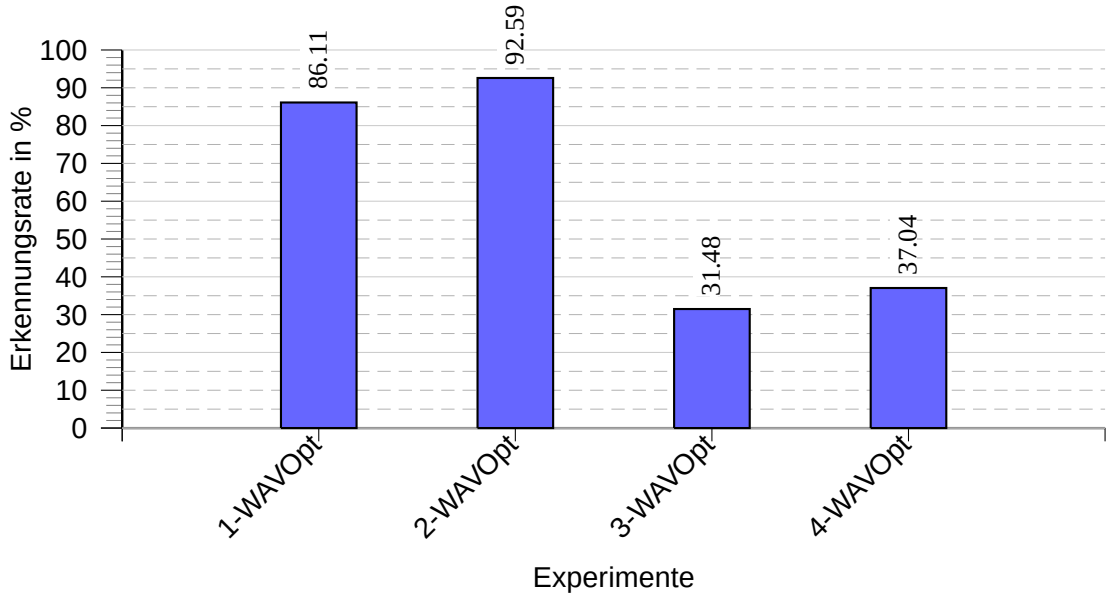
Training								
Experiment Name	Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler* [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]	Sounds [Anzahl]
1-WAVOpt	1	0	10	111100	111	110011	100000000000	217
2-WAVOpt	1	0	10	111100	111	110000	100011001100	328
3-WAVOpt	1	0	10	111100	111	110000	100000000000	108
4-WAVOpt	1	0	10	111100	111	110000	100000000000	108
Test								
Experiment Name	Train [Bit 1]	Test [Bit 1]	Kanal [Bit 1-2]	Spieler* [Bit 1-20]	Spielart [Bit 1-3]	Saite [Bit 1-6]	Bund [Bit 1-12]	Sounds [Anzahl]
1-WAVOpt	0	1	10	111100	111	110011	100000000000	216
2-WAVOpt	0	1	10	111100	111	110000	100011001100	324
3-WAVOpt	0	1	10	111100	111	000011	100000000000	108
4-WAVOpt	0	1	10	111100	111	001100	100000000000	108

*(Nur die Bits 1-6 sind angezeigt worden. Die restlichen 14 Bits haben Wert “0“.)

[12] Der Name “WAVOpt“ in dieser Experimenten-Reihe ist durch das Kürzel WAV (für die wav-Dateien) und Opt (für die Optimierung) gebildet worden, weil es in diesen Experimenten um Optimierung bzw. Reduktion der Anzahl von Audiosignale geht.

Abbildung 9.5.: Ergebnisse für die Experimente zur WAV Optimierung

● Ziel: Den Klassifikationprozess noch einer Probe unterziehen durch die Experimente mit reduzierter Anzahl der Audiosignale.
● Beschreibung: Durchgeführt wurden zwei Experimente, die besonders gute Ergebnisse liefern sollten (1-WAVOpt und 2-WAVOpt), und anschließend zwei Experimente, die besonders schlechte Erkennungsraten erreichen sollten (3-WAVOpt und 4-WAVOpt).
● Ergebnis: Beide Experimenten-Paare haben die Ergebnisse geliefert, die erwartet wurden.



10. Experimente mit den verschiedenen SVM-Kerneln

Alle Experimente, die in den Kapiteln 6, 7, 8 und 9 beschrieben wurden, sind mit den gleichen Einstellungen (Linear-Kernel $(a * b)$) der SVM^{light} Implementierung durchgeführt worden. In diesem Kapitel wird untersucht, ob sich durch die Benutzung eines anderen Kernel-Typs die Erkennungsrate noch verbessern lässt. Bei der Lösung dieser Aufgabe sollte man vor Augen haben, dass eine Lösung nur empirisch zu finden ist. Hier kommt einer der Nachteile von SVM zur Geltung, dass nämlich eine Ermittlung der optimalen Einstellungen, die im Klassifikationsprozess eingesetzt werden, nur durch Ausprobieren möglich ist.

Um überhaupt das Gefühl zu bekommen, auf welche Art und Weise der SVM funktioniert und wie die Veränderung der einzelnen Parameter auf die Ermittlung der Hyperebene wirkt, kann man von einem Visualisierungsprogramm Gebrauch machen. So ein Graphik-Interface[13] ist auf der Internetseite der LIBSVM-Implementierung vorgestellt worden. Dieses Grafik-Interface kann man online benutzen. Das Programm kann man nach dem Download auch lokal als ein Applet im Web-Browser verwenden.

Die Abbildungen 10.1 und 10.2 zeigen jeweils zwei Beispiele, die den Umgang mit dem LIBSVM-Grafik-Interface verdeutlichen.

In der Abbildung 10.1 wird die Ermittlung der Hyperebene unter der Verwendung des Polynomial-Kernels ($(sa * b + c)^d$) gezeigt. Die Abbildung zeigt Werte, die zu zwei verschiedenen Klassen gehören. Diese Werte sind durch den Linear-Kernel nicht vollständig trennbar. Würde man die einzelnen Werte der Klassen mit einer Linie verbinden, so würde eine Parabel-ähnliche Kurve entstehen. Der linke Teil der Abbildung zeigt, dass sich solche Aufgaben mit dem Polynomial-Kernel zufriedenstellend lösen lassen. Hier wurde der Exponent des Polynoms auf Wert 2 gesetzt (Kernel-Parameter -d 2). In dem rechten Teil der Abbildung wurde Kernel-Parameter -d 1 verwendet. Diese Einstellung liefert identische Ergebnisse wie ein Linear-Kernel.

[13] Download unter: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Die Abbildung 10.2 zeigt die Werte von zwei Klassen, die stärker ineinander verschränkt sind. Hier wird die Trennung der Klassen unter Verwendung eines Radial-Basis-Function-Kernels ($exp(-gamma||a-b||^2)$) und des dazugehörigen Kernel-Parameters gamma -g 1 (links) und -g 3 (rechts) gezeigt.

Abbildung 10.1.: Ermittlung der Hyperebene mit dem Polynomial-Kernel

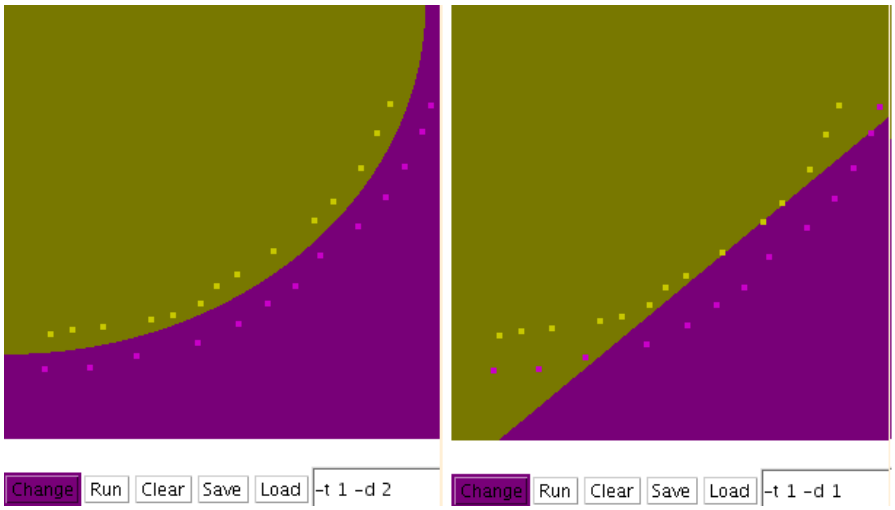
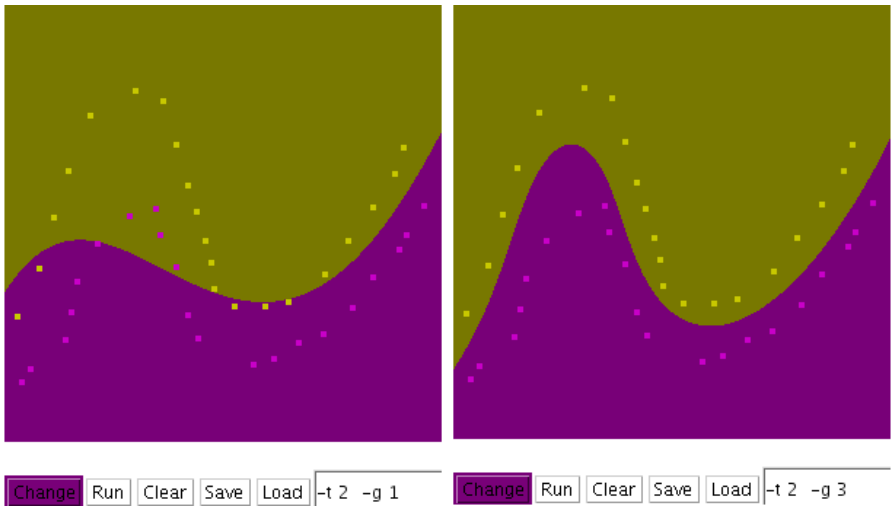


Abbildung 10.2.: Ermittlung der Hyperebene mit dem RBF-Kernel



10.1. Durchführung der Experimente und deren Ergebnisse

Weitere Testvorgänge, die mit dem LIBSVM-Grafik-Interface durchgeführt wurden, haben gezeigt, dass der Einsatz des Polynomial- und RBF-Kernels gute Ergebnisse liefen könnte. Um die Erkennungsraten miteinander vergleichen zu können, sind als Referenz fünf Experimente (01-TNM bis 05-TNM) ausgewählt worden, die im Kapitel 8 beschrieben wurden.

In der Abbildung 10.3 sind die Referenz-Erkennungsraten dargestellt und die Erkennungsraten der Experimente, die mit dem Polynomial-Kernel durchgeführt wurden. Die Experimente 01-TNM, 02-TNM, 03-TNM, 04-TNM und 05-TNM, gehören zu der Referenzgruppe 1 und die Experimente 1-WAVOpt und 2-WAVOpt wurden der Referenzgruppe 2 zugeordnet.

Die Experimente der Referenzgruppe 1 sind als Referenz-Experimente ausgewählt worden, weil sie die Ergebnisse darstellen, die zu derjenigen Experimenten-Gruppe gehören, die zur Reduktion der Daten in den Spalten der Merkmalsmatrix durchgeführt wurden. Diese fünf Experimente wurden mit 978 Train-Merkmalsvektoren und 972 Test-Merkmalsvektoren durchgeführt. Die Experimente hatten folgenden Längen der Merkmalsvektoren benutzt: das Experiment 01-TNM mit der kleinsten (noch als sinnvoll ermittelten) Größe des Merkmalsvektors (505 Werte) (Siehe Kapitel 8), das Experiment 02-TNM (605 Werte), das Experiment 03-TNM (705 Werte), das Experiment 04-TNM (805 Werte) und das Experiment 05-TNM hat den größten, Merkmalsvektor (mit der vollen Anzahl der Werte), der 1155 Werte beinhaltet. Deshalb sind diese Experimente besonders gut als Referenz geeignet, um die Ergebnisse der aktuellen Experimente, die mit den verschiedenen Kerneln durchführt werden, mit den Referenz-Experimenten zu vergleichen. Das Ziel ist, zu ermitteln, wie sich die Erkennungsraten der aktuellen Experimente mit verschiedenen Merkmalsvektors-Größen durch Verwendung eines anderen Kernels ändern werden.

Die Experimente der Referenzgruppe 2 sind als Referenz-Experimente ausgewählt worden, weil sie einerseits zur Experimenten-Gruppe gehören, die zur Reduktion der Daten in den Reihen der Merkmalsmatrix gehören und andererseits, weil diese zwei Experimente die bisher besten Erkennungsraten geliefert haben. Diese Experimente sind mit der Merkmalsvektor-Größe von 505 Werten und deutlich reduzierter Anzahl von Merkmalsvektoren durchgeführt worden. Die Anzahl der Merkmalsvektoren beträgt: in dem Experiment 1-WAVOpt 217 Train-Merkmalsvektoren, bzw. 216 Test-Merkmalsvektoren und in dem Experiment 2-WAVOpt 328 Train-Merkmalsvektoren, bzw.

328 Test-Merkhalsvektoren. (Siehe Abschnitt 9.6.) Das Ziel ist, zu ermitteln, ob sich diese Ergebnisse durch Verwendung eines anderen Kernels noch verbessern lassen.

Die Ergebnisse der Referenzgruppe 1 sind mit den Ergebnissen der Experimente 11-Poly, 12-Poly, 13-Poly und 14-Poly paarweise zu vergleichen (Experimenten-Paare 01-TNM/11-Poly, 02-TNM/12-Poly, 03-TNM/13-Poly und 04-TNM/14-Poly). Die Konfigurationsparameter dieser Experimenten-Paare sind bis auf den verwendeten Kernel-Typ identisch. Die Experimente der Referenzgruppe 1 wurden mit dem Linear-Kernel durchgeführt, und in den “xx-Poly“-Experimenten ist der Polynomial-Kernel (Kernel-Parameter -d 3) eingesetzt worden.

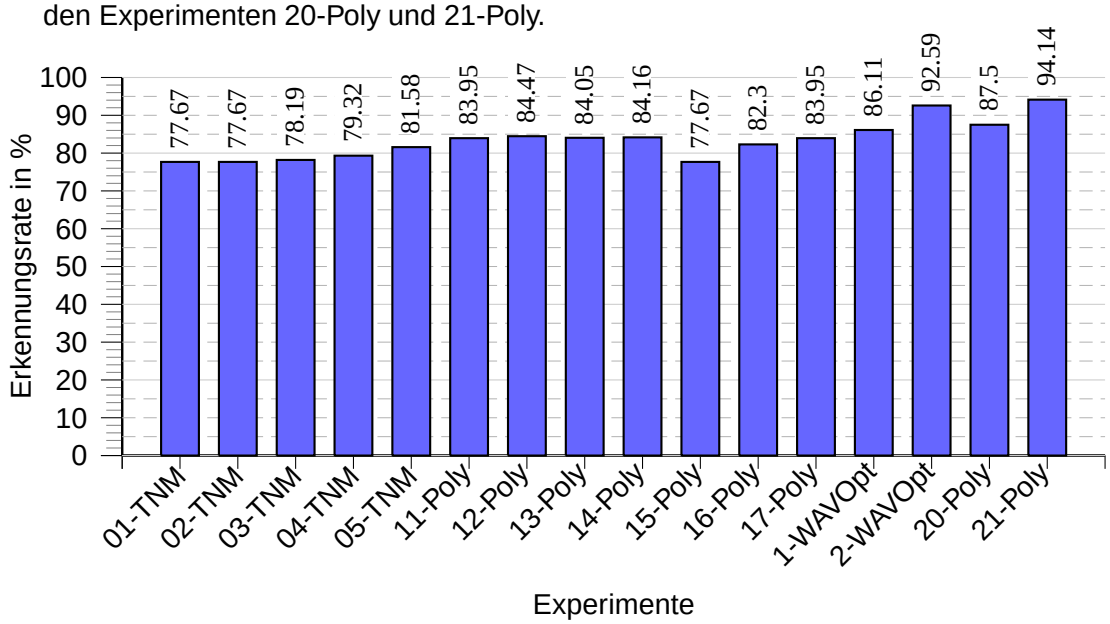
Durch die Verwendung des Polynomial-Kernels ist die Erkennungsrate deutlich besser geworden und beträgt durchschnittlich ungefähr 84 %. Zu beobachten ist, dass die Größe des Merkmalsvektors fast keinen Einfluss auf die Erkennungsrate hat. Die größte Erhöhung der Erkennungsrate ist in den Experimenten 01-TNM/11-Poly zu beobachten. Das ist besonders günstig, weil diese beiden Experimente die kleinsten Merkmalsvektoren (505 Werte) haben und deshalb relevant für die Ermittlung der Verbesserung der Erkennungsrate sind, die von 77.67 % (01-TNM) auf 83.95 % (11-Poly) gestiegen ist.

Die Experimente 15-Poly, 16-Poly und 17-Poly, sind mit identischen Konfigurationsparametern wie 01-TNM/11-Poly durchgeführt worden. Sie unterscheiden sich nur durch den Wert des Kernel-Parameters -d des Polynomial-Kernels. Die Kernel-Parameter der Experimente sind 15-Poly (-d 1), 16-Poly (-d 2) und 17-Poly (-d 3). Diese drei Experimente zeigen, dass der Kernel-Parameter (-d 3) die besten Ergebnisse unter Verwendung des Polynomial-Kernels geliefert hat. In den weiteren Experimenten wurden auch weitere Kernel-Parameter (Exponenten) getestet. Diese haben sich für die Klassifikation aber als nicht geeignet erwiesen.

Um diese Ergebnisse einer weiteren Untersuchung zu unterziehen, wurden noch zwei Experimente durchgeführt, die als Referenz die Experimente 1-WAVOpt und 2-WAVOpt (Referenzgruppe 2) haben. (Siehe Abschnitt 9.6.) Diese zwei Experimente sind als Referenz ausgewählt worden, weil sie schon gute, bzw. sehr gute Ergebnisse geliefert haben (1-WAVOpt (86.11 %) und 2-WAVOpt (92.59)). Das Ziel ist, zu prüfen, ob diese Ergebnisse noch verbessert werden können. Auch hier sind die Ergebnisse paarweise (Referenz/Experiment) zu vergleichen (Experimenten-Paare 1-WAVOpt/20-Poly und 2-WAVOpt/21-Poly). Die Experimenten-Paare unterscheiden sich nur durch den verwendeten Kernel-Typ. Bei den Referenz-Experimenten wurde ein Linear-Kernel benutzt, bei den aktuellen Experimenten dagegen ein Polynomial-Kernel mit dem Kernel-Parameter (-d 3). Die Erkennungsrate hat sich durchschnittlich um knapp 1,5 Prozentpunkte verbessert und beträgt im Experiment 20-Poly 87.50 %, bzw. im Experiment 21-Poly 94.14 % (Siehe Abbildung 10.3).

Abbildung 10.3.: Ergebnisse der Experimente mit dem Linear- und Polynomial-Kernel

• Ziel: Ermitteln, ob sich die Ergebnisse durch die Verwendung anderer Kernel-Typen verbessern lässt.
• Beschreibung: Die mit dem Linear-Kernel durchgeführten Experimente wurden unter der Verwendung der identischen Konfigurationsparameter mit dem Polynomial-Kernel wiederholt.
• Ergebnis: Die Erkennungsraten sind durch die Verwendung des Polynomial-Kernels verbessert worden. Besonders gutes Ergebnis lieferte Experiment 11-Poly, gefolgt von den Experimenten 20-Poly und 21-Poly.



Es ist eine ganze Reihe von Experimenten mit dem Radial-Basis-Function-Kernel (RBF) ($exp(-gamma||a-b||^2)$) und dem Sigmoid-Kernel ($tanh(s(a*b)+c)$) durchgeführt worden. Experimentiert wurde mit den verschiedenen -gamma-Werten im RBF-Kernel, bzw. -s- und -c-Parametern im Sigmoid-Kernel. Zusätzlich wurde auch mit den Lern-Parametern (wie cost-factor) der SVM^{light} Implementierung experimentiert. Alle diese Experimente haben keinerlei zufriedenstellende Ergebnisse geliefert.

11. Fazit und Ausblick

Die Ergebnisse der Experimente, die in dieser Arbeit durchgeführt wurden zeigen, dass sich die Erkennungsrate (gemessen an der Referenz-Erkennungsrate von 81.58 %) noch verbessern lässt.

Die Referenz-Erkennungsrate wurde in dem Experiment 05-TNM erreicht. (Siehe Kapitel 8.) In dem Experiment 05-TNM wurde eine Erkennungsrate von 81.58 % erreicht. Diese Erkennungsrate wurde durch Verwendung der vollen Größe der Merkmalsvektoren (1155 Werte) und der höchsten zur Verfügung stehenden Anzahl der Audiosignale (978 Train-Audiosignale und 972 Test-Audiosignale) erreicht. Das ist die beste durchschnittliche Erkennungsrate, die erreicht wurde. Unter dem Begriff “durchschnittliche Erkennungsrate“ ist die Verwendung von allen Audiosignalen gemeint, ohne die Audiosignale nach Selektions-Kriterien (Spieler, Saite, Bund, Klangvorgabe) zu selektieren. (In den Experimenten, in denen die Audiosignale selektiert wurden, wurden auch höhere Erkennungsraten erreicht.) Deshalb wird diese Erkennungsrate als Referenz genommen, die an dieser Stelle als Ausgangspunkt für die Vergleiche zwischen den erreichten Erkennungsraten eingesetzt wird.

Die ersten Experimente haben gezeigt, dass eine maximale Verkleinerung der Merkmalsvektoren auf 505 Werte pro Merkmalsvektor noch sinnvoll sein kann. Bei dieser Größe der Merkmalsvektoren muss man Einbußen in Kauf nehmen. Die Erkennungsrate ist dadurch auf 77,67 % gefallen.

Die weiteren Experimente, die als Ziel eine Reduktion der Anzahl von Audiosignalen hatten, haben Erkenntnisse geliefert, die zeigten, welche Klänge besonders gut für die Klassifikation geeignet sind. So zeigte sich, dass die Klänge, die mit der Klangvorgabe “warm“ aufgenommen wurden, etwas besser geeignet sind als andere. Die Klänge, die auf der Saite 1 (E) und 2 (H) gespielt wurden, werden deutlich besser erkannt als die anderen Klänge. Außerdem werden die Klänge, die auf dem Bund 1 gespielt wurden, besser erkannt als die, die auf den anderen Bünden aufgenommen wurden. Hier wurde eine Erkennungsrate von 92,59 % erreicht, unter Verwendung von 505 Werten pro Merkmalsvektor und der Klassifizierung der Klänge, die nur auf der Saite 1 und Saite 2 gespielt wurden.

Alle Experimente bis auf die im Kapitel 10 beschriebenen sind mit dem Linear-Kernel durchgeführt worden. Die Verwendung des Polynomial-Kernels hat noch eine Steigerung der Erkennungsrate gebracht. Die Erkennungsrate konnte, unter identischen Klassifikationsbedingungen (505 Werte pro Merkmalsvektor, 978 Train- und 972 Test-Audiosignalen) von 77,67 % auf 83,95 % erhöht werden. Dieses Ergebnis ist besser als die Referenz-Erkennungsrate, die 81.58 % beträgt.

Die Ergebnisse der Experimente bestätigen die Robustheit des Klassifikationsverfahtens, das sich als unempfindlich gegen den Einfluss des Spielers auf die Erkennungsrate zeigt. Die Verwendung der Skalierungsfaktoren auf die Merkmalswerte (MFCCs und Harmonische) hat nicht die erhofften Ergebnisse gebracht. Unter Verwendung der Skalierungsfaktoren sind die Ergebnisse sogar etwas schlechter ausgefallen. Genau so hat sich der Einsatz der Radial-Basis-Function-Kernel und Sigmoid-Kernel als nicht geeignet für diese Art von Daten erwiesen.

Die Softwareumgebung kann nur die Klassifizierung von drei Gitarren durchführen. Das liegt an dem azyklischen Diagramm (DAG), das für drei Gitarren programmiert wurde. Es ist denkbar, die Softwareumgebung so zu erweitern , dass man nur zwei oder mehr als drei Gitarren klassifizieren kann, bzw. die Software so zu programmieren, dass man den DAG für eine unterschiedliche Anzahl von Gitarren konfigurieren kann.

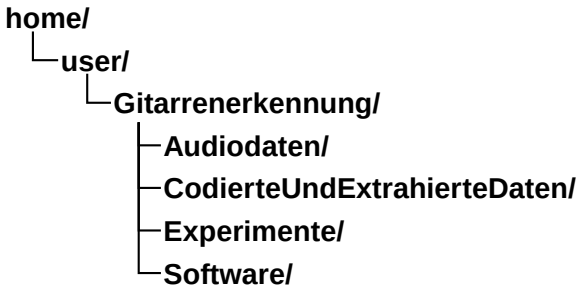
Bei der Anzahl von Experimenten (über 150), die in dieser Arbeit durchgeführt wurden, zeigte sich die Auswertung der Ergebnisse als mühsam und fehlerträchtig. Es wäre sinnvoll, die Softwareumgebung um konfigurierbare Ergebnis-Analyse- und Auswertung-Tools zu erweitern. So würde man die gewünschten Ergebnisse automatisch und fehlerfrei geliefert bekommen, ohne negative Einflüsse, die durch manuelle Auswertung entstehen können. Dateien, die man in den Analyseprozess einbeziehen sollte, sind die Log-Dateien der CE- und STT-Phase und die Dateien, die Ergebnisse und Debugg- bzw. Lern-Ausgaben der SVMs beinhalten.

A. Kurzbeschreibung zur Software und ihrer Installation

A.1. Verzeichnisstruktur

Die Software bzw. die Daten, die man zur Durchführung der Installation braucht, sind in mehrere Stamm- und Unterverzeichnisse verteilt. Die hier beschriebene Verzeichnisstruktur kann man letztendlich als einen Vorschlag verstehen. Alle Verzeichnisse, bis auf eine Ausnahme, kann man nach eigenen Wünschen benennen. Die Ausnahme sind die Test- und Train-Verzeichnisse, die im Audiodaten-Container-Verzeichnis zu finden sind. Sollte man sich entschließen, eine eigene Verzeichnisstruktur zu erzeugen, so wird man auch eine umfangreichere Anpassung der Konfigurationsdateien in Kauf nehmen müssen.

A.1.1. Hauptverzeichnis und Stammverzeichnisse des Projekts



Gitarrenerkennung - ist das Hauptverzeichnis des Projekts. **Audiodaten** - enthält eine Verzeichnisstruktur, in der die Audiodateien liegen, die in der Codierungs- und Extrahierungsphase gebraucht werden. **CodierteUndExtrahierteDaten** - enthält eine Verzeichnisstruktur, in der die Ergebnisse der Codierungs- und Extrahierungsphase (CE-Phase) liegen. **Experimente** - enthält die einzelnen Experimentverzeichnisse. **Software** - enthält die selbst entwickelten Programme, Startskripten, Konfigurationsdateien und die SVM Software.

A.1.2. Stammverzeichnis für Audiodaten

Das Stammverzeichnis für Audiodaten kann mehrere Verzeichnisse von Audiodaten-Containern enthalten. Die vorgeschlagene Strukturart (Train und Test Verzeichnis und jeweils ein Ordner pro Gitarre) sollte jedoch beibehalten werden.

Audiodaten/ . . . . . Stammverzeichnis Audiodaten
- **Demo/** . . . . . Audiodaten-Container für Test- und Trainingsdaten
  - **Test/** . . . . . Testdaten
    - **Hense/** . . . . . Gitarre 1
    - **Marin/** . . . . . Gitarre 2
    - **Wichmann/** . . . . . Gitarre 3
    - **xyz/** . . . . . Platzhalter für weitere Gitarren
  - **Train/**
    - **Hense/** . . . . . Gitarre 1
    - **Marin/** . . . . . Gitarre 2
    - **Wichmann/** . . . . . Gitarre 3
    - **xyz/** . . . . . Platzhalter für weitere Gitarren
- **xyz/** . . . . . Platzhalter für weiteren Audiodaten-Container

A.1.3. Stammverzeichnis für kodierte und extrahierte Daten

CodierteUndExtrahierteDaten/ . . . . . Stammverzeichnis CodierteUndExtrahierteDaten
- **Demo/** . . . . . Kodierte und extrahierte Test- und Trainingsdaten
  - **TestCodiert/** . . . . . Kodierte Testdaten
  - **TestExtrahiert/** . . . . . Extrahierte Testdaten
  - **TrainCodiert/** . . . . . Kodierte Trainingsdaten
  - **TrainExtrahiert/** . . . . . Extrahierte Trainingsdaten
  - **codExtrResult.txt** . . . . . Log-Datei der CE-Phase
- **xyz/** . . . . . Platzhalter für weiteren CUED-Verzeichnisse

A.1.4. Stammverzeichnis für Experimente

In diesem Verzeichnis werden die Ergebnisse aus der STT-Phase abgelegt.

Experimente/ . . . . . Enthält die einzelnen Experimente
- **Demo_01/** . . . . . Verzeichnisstruktur der Experimente
  - **SVM/** . . . . . Beinhaltet die Ausgaben der SVMs
    - **Testoutput/** . . . . . Beinhaltet die Ergebnisse des Testlaufs
    - **TrainierteSVMs/** . . . . . Beinhaltet die trainierten SVMs
    - **Trainingsoutput/** . . . . . Beinhaltet die Trainingsausgaben der SVMs
  - **TestSelektFiles/** . . . . . Beinhaltet die selektierten Testdaten, die für die Verwendung in der SVM Testphase vorbereitet sind
  - **TestSelektMatrix/** . . . . . Beinhaltet die selektierten Testdaten, die im Octave Matrix Format abgespeichert sind
  - **TrainSelektFiles/** . . . . . Beinhaltet die selektierten Trainingsdaten, die für die Verwendung in der SVM Trainingsphase vorbereitet sind
  - **TrainSelektMatrix/** . . . . . Beinhaltet die selektierten Trainingsdaten, die im Octave Matrix Format abgespeichert sind
  - **selektResult.txt** . . . . . Log-Datei STT-Phase
- **Demo_02/** . . . . . Verzeichnisstruktur der Experimente
- **Demo_03/** . . . . . Verzeichnisstruktur der Experimente
- **xyz/** . . . . . Platzhalter für weiteren Audiodaten-Container

A.1.5. Stammverzeichnis für die Software

Software/ . . . . . Stammverzeichnis für die selbst entwickelte Software und für SVM-Light
- **CodierenUndExtrahieren/** . Programme zum Kodieren und zur Merkmalsextraktion
- **GemeinsameDaten/** . . . . . Software, die in der CE- und STT-Phase benutzt wird
- **Selektieren/** . . . . . Software, die in der STT-Phase benutzt wird, bereitet die Daten für die Training- und Testphase vor
- **Startskripten/** . . . . . Skript- und Konfigurationsdateien
  - **Ce/** . . . . . Daten für die CE-Phase
    - **Demo/** . . . . . Demo-Projekt Startskript und Konfigurationsdatei
    - **xyz/** . . . . . Platzhalter für weitere Startskripten und Konfigurationsdateien
  - **Stt/** . . . . . Daten für die STT-Phase
    - **Demo/** . . . . . Demo-Projekt Startskript und Konfigurationsdatei
    - **xyz/** . . . . . Platzhalter für weitere Startskripten und Konfigurationsdateien
  - **CeStt/** . . . . . Daten für die CESTT-Phase
    - **Demo/** . . . . . Demo-Projekt Startskript und Konfigurationsdatei
    - **xyz/** . . . . . Platzhalter für weitere Startskripten und Konfigurationsdateien
- **Testen/** . . . . . Programme zum Testen der trainierten SVMs
- **Trainieren/** . . . . . Programme zum Trainieren der SVMs

A.2. Softwareinstallation

Um die Software erfolgreich zum Laufen zu bringen, benötigt man einen PC mit installiertem Linux Betriebssystem und die GNU Octave Skriptsprache.
Die einfachste Art, die Software zu installieren ist, das zur Verfügung gestellte Hauptverzeichnis “Gitarrenerkennung“ samt allen Unterverzeichnissen im *“/home/user/“* Verzeichnis ihrer Linux Distribution zu kopieren. In den Verzeichnissen *“/Software/Testen/“* und *“/Software/Trainieren/“* befindet sich schon das lauffähige und konfigurierte SVM-light Programm in der Version V6.01.

Die Softwareinstallation ist mit folgenden Programmen und folgender Hardware getestet worden:

- Rechner: 1 GB RAM, Prozessor: AMD Sempron (LE-1150) 2 GHz
- Betriebssystem Linux Ubuntu 8.10
- GNU Octave, Version 3.0.1 inklusive Octave-Pakete
  - octave-io, Version 1.0.6
  - octave-signal, Version 1.0.7
- Support Vector Machines SVM^{light}, Version V6.01
- Das von Anton Schwaighofer entwickelte Matlab-Interface zur SVM^{light}, Version 0.92

A.2.1. GNU Octave Installation

Die oben erwähnte GNU Octave zusammen mit den beiden Octave-Paketen werden als Standardpakete des Linux Ubuntu 8.10 Betriebssystems[14] geliefert. Zur Installation kann man das Installationsprogramm “Synaptic-Paketverwaltung“ des Betriebssystems benutzen, das die grafische Oberfläche zur Installation, Deinstallation und Verwaltung von Paketen unter Linux Ubuntu bietet. Das Programm “Synaptic-Paketverwaltung“ ist einfach und intuitiv zu bedienen, und zwar auch für die Benutzer, die keine Erfahrung mit den Unix- und Linux-basierten Betriebssystemen haben.

[14] Download unter: `http://wiki.ubuntuusers.de/Downloads`

A.2.2. SVM Installation

Die SVM^{light} Software kann man auf der Webseite des Autors[15] besorgen. Die Software kann als “Source Code“ oder “Binaries“ heruntergeladen werden und ist für die Betriebssysteme Solaris, Windows und Linux verfügbar. Außerdem ist die Software auch unter Cygwin[16] lauffähig. Wenn man sich für die “Source Code“ Version des Programms entscheidet, soll die Software zunächst kompiliert werden. Als Ergebnis des Kompilierungsprozesses werden zwei Programme erzeugt. Das sind `svm_learn` und `svm_classify`, die in der Trainings- bzw. in der Test-Phase verwendet werden. Einzelne Compilierungs-Befehle sind auf der Webseite des Autors beschrieben. Außerdem sind dort die Beispiele und Tutorien zur SVM zu finden.

A.2.3. Matlab-Interface zur SVM

Das von Anton Schwaighofer entwickelte Matlab-Interface[17] zu SVM^{light} kann auch innerhalb von Octave eingesetzt werden. In dem Interface sollen die Einträge in den Dateien `svm_learn.m` (Siehe Listing A.1) und `svm_classify.m` (Siehe Listing A.2) angepasst werden. Die Dateien sind unter den Pfaden *“/Gitarrenerkennung/Software/-Trainieren/“* und *“/Gitarrenerkennung/Software/Testen/“* zu finden.

Im Listing A.1 ist die Zeile 139 auskommentiert und durch die Zeile 140 ersetzt worden. Das Gleiche gilt für Listing A.2. Dort ist die Zeile 50 durch die Zeile 51 ersetzt worden. Diese Anpassung der Pfad-Angaben für die beiden SVM-Programme gewährleistet, dass die Programme an beliebiger Stelle im Klassifikationsprozess aus den Startskripten aufgerufen werden können.

Listing A.1: Anpassung der “svm_learn.m“ Datei

```
Zeile 139:  %evalstr = [fullfile(options.ExecPath, 'svm_learn') s '␣'
    ...
Zeile 140:  evalstr = [fullfile(options.ExecPath, '~/
   Gitarrenerkennung/Software/Trainieren/svm_learn') s '␣' ...
Zeile 141:            examples '␣' model '>>' svmOutput];
```

Listing A.2: Anpassung der “svm_classify.m“ Datei

```
Zeile 50:  %evalstr = [fullfile(options.ExecPath, 'svm_classify') s '
   ␣' ...
```

[15] Download unter: `http://svmlight.joachims.org/`
[16] Das ist eine UNIX-artige Umgebung unter Windows
[17] Download unter: `http://ida.first.fraunhofer.de/~anton/software.html`

```
Zeile 51:  evalstr = [fullfile(options.ExecPath, '~/Gitarrenerkennung
   /Software/Testen/svm_classify') s ' ' ...
Zeile 52:             data ' ' model ' ' predictions];
```

B. Tutorium

Der Klassifikationsprozess von Audiosignalen läuft in zwei getrennten Phasen ab. Normalerweise hat man eine Menge von Audiodateien vorliegen, deren Merkmale in der Regel nur einmal extrahiert und anschließend in geeigneter Form abgespeichert werden. So ist gewährleistet, dass man auf die gespeicherten Daten jederzeit zurückgreifen kann, um sie für verschiedene Experimente zu verwenden, ohne dabei eine sehr zeitraubende und rechenintensive Merkmalsextraktions-Phase wiederholen zu müssen.

1. **Kodieren und Extrahieren (CE-Phase)** In dieser Phase wird die Merkmalsextraktion durchgeführt, und anschließend werden die Audiodateien anhand ihrer Namen kodiert.
2. **Selektieren, Trainieren und Testen (STT-Phase)** In dieser Phase wird nur eine Untermenge der gesamten Audiodaten selektiert. Mit ausgewählten Daten wird anschließend mit SVMs trainiert und getestet.
3. **CE- und STT-Phase kombinieren (CESTT)** Die beide Phasen kann man auch nach Bedarf aus einem Startskript nacheinander aufrufen. Manchmal ist es günstig so vorzugehen, je nachdem was für ein Art von Experiment man durchführen möchte.

B.1. Kodieren und Extrahieren (CE-Phase)

Bei der im Rahmen dieser Bachelorarbeit selbst entwickelten Software handelt es sich um Octave-Skript- und -Funktions-Dateien, die durch ein Startskript aufgerufen werden. Dazu gehört auch eine Konfigurationsdatei, in der man festlegen kann, unter welchen Bedingungen die CE-Phase durchgeführt werden soll.

B.1.1. Demo-Projekt starten

Um festzustellen, ob die Software auf einem Rechner funktioniert, ist der schnellste Weg, das mitgelieferte Demo-Projekt im *"/home/user/"* Verzeichnis der Linux Distribution

zu kopieren und das Startskript *"/Gitarrenerkennung/Software/Startskripten/Ce/Demo/-start_ce_demo.m"* im Terminal zu starten. Siehe Abbildung B.1.

Abbildung B.1.: Starten der CE-Phase

Als Eingangsgrösse braucht die CE-Phase Audiodateien, die im Verzeichnis "Demo" *(/Gitarrenerkennung/Audiodaten/Demo)* liegen. Die Ergebnisse aus der CE-Phase werden in *"GitarrenerkennungCodierteUndExtrahierteDaten/Demo"* abgespeichert. Das Demo-Projekt benutzt nur je 6 Audiodateien pro Gitarre. Insgesamt sind es 36 Dateien, 18 Dateien für die SVM-Trainingsphase und 18 Dateien für die SVM-Testphase.

Nach der erfolgreich beendeten CE-Phase, die ein paar Minuten dauern kann, findet man im Stammverzeichnis "CodierteUndExtrahierteDaten" ein neu erzeugtes "Demo" Verzeichnis mit dazugehöriger Ordner- und Dateistruktur. Siehe Abschnitt A.1.3.

B.1.2. Wichtige Projektdateien in der CE-Phase

Hier folgt eine kurze Beschreibung wichtiger Projektdateien. Sehr wichtige Punkte werden genauer beschrieben, und der Rest wird nur blockweise erläutert. Weiter unten werden die folgende Dateien beschrieben.

1. **Startskript** *"start_ce_demo.m"*
2. **Konfigurationsdatei** *"config_ce_demo.txt"*
3. **Log-Datei** *"codExtrResult.txt"*

Startskript: Der komplette Inhalt des Startskripts *"start_ce_demo.m"* ist im Abschnitt D.1 zu sehen. In der Zeile 8 des Startskripts wird man in der Regel den Namen der Konfigurationsdatei anpassen müssen. Dabei gehe ich davon aus, dass man für jede neue CE-Phase, die man unter veränderten Bedingungen laufen lässt, auch eine neue Konfigurationsdatei erzeugen wird.

```
Zeile 8:  configFileName = 'config_ce_demo.txt';
```

Konfigurationsdatei: Der komplette Inhalt der Konfigurationsdatei *"config_ce_demo.txt"* ist im Abschnitt D.2 zu sehen. Anschließend ist eine Übersicht über die Variablen gegeben, deren Wert man in der Regel anpassen sollte. In diesem Fall handelt es sich um zwei Demoverzeichnisse. Das Verzeichnis "Demo" enthält in der Zeile 40 die Audiodateien, und in der Zeile 46 das Zielverzeichnis für die Ergebnisse aus der CE-Phase.

```
Zeile 40:  audioVerzeichnis   Demo
Zeile 46:  cuedVerzeichnis    Demo
```

Log-Datei: Während jeder CE-Phase wird eine *"codExtrResult.txt"* Log-Datei erzeugt. Siehe Abschnitt D.3. Dort werden Einträge und Meldungen über Ereignisse eingetragen, die in der CE-Phase passieren. Man kann in der Log-Datei folgende Blöcke erkennen und folgende Informationen entnehmen:

- Zeilen 1 - 6: Einige Statistische Daten wie Benutzername, Datum und Uhrzeit, verwendete Konfigurationsdatei, Audiodateien-Verzeichnis und das Verzeichnis für kodierte und extrahierte Daten.
- Zeilen 8 - 23: Validierung der vorhandenen und zur Laufzeit erzeugten Verzeichnisse.
- Zeilen 25 - 67: Statistik über die Spieler und die vorgefundenen Audiodateien, die in der CE-Phase verwendet werden.
- Zeilen 69 - 79: Konfigurationsparameter für die Merkmalsextraktions-Phase.
- Zeile 80: Länge der CE-Phase in Sekunden.

B.2. Selektieren, Trainieren und Testen (STT-Phase)

Auch die Software, die zum Selektieren, Trainieren und Testen entwickelt wurde, besteht aus mehreren Octave-Skript- und -Funktions-Dateien. Hier gibt es ein zentrales Hauptstartskript und ein oder mehrere weiteren Startskripten und Konfigurationsdateien, die alle einzelnen Experimente starten und konfigurieren. Darüber hinaus sind hier drei weitere Konfigurationsdateien zu finden, die ermöglichen, jede einzelne Phase (Selektieren, Trainieren und Testen) zusätzlich zu konfigurieren.

B.2.1. Demo-Projekt starten

Vorausgesetzt, dass die im Abschnitt B.1.1 beschriebenen Vorbereitungen getroffen worden sind, kann man das *"/Gitarrenerkennung/Software/Startskripten/Stt/Demo/-start_stt_demo_all.m"* Startskript im Terminal aufrufen. Siehe Abbildung B.2.

Abbildung B.2.: Starten der STT-Phase

Als Eingang für die STT-Phase werden extrahierte Daten gebraucht, die in *"/Gitarrenerkennung/CodierteUndExtrahierteDaten/Demo"* Verzeichnis liegen. Die Ergebnisse der STT-Phase werden im Stammverzeichnis *"Experimente"* in die aktuell erzeugte Ordner- und Dateistruktur gespeichert, die im Abschnitt A.1.4 beschrieben ist. Nach der erfolgreich beendeten STT-Phase sind im Stammverzeichnis *"Experimente"* 3 Ordner Demo_01, Demo_02 und Demo_03 zu finden.

B.2.2. Wichtige Projektdateien in der STT-Phase

Hier folgt eine kurze Beschreibung wichtiger Projektdateien. Sehr wichtige Punkte werden genauer beschrieben, und der Rest wird nur blockweise erläutert. Weiter unten werden die folgenden Dateien beschrieben.

1. **Hauptstartskript** (Abschnitt B.2.2) Inhalt der *"start_stt_demo_all.m"* Datei. (Listing E.1)
2. **Startskript** (Abschnitt B.2.2) Inhalt der *"start_stt_demo1.m"* Datei. (Listing E.2)
3. **Konfigurationsdatei** (Abschnitt B.2.2) Inhalt der *"config_stt_demo1.txt"* Datei. (Listing E.3)
4. **Skalierungsparameter** (Abschnitt B.2.2) Inhalt der *"getScalingValueDefault.m"* Datei. (Listing E.4)
5. **Optionen für SVM-Training** (Abschnitt B.2.2) Inhalt der *"trainOptionsDefault.m"* Datei. (Listing E.5)

6. **Optionen für SVM-Test** (Abschnitt B.2.2) Inhalt der *“testOptionsDefault.m“* Datei. (Listing E.6)
7. **Log-Datei** (Abschnitt B.2.2) Inhalt der *“selektResult.txt“* Datei. (Listing E.7)
8. **Klassifikationsergebnisse** (Abschnitt B.2.2) Inhalt der *“classifikationResult.txt“* Datei. (Listing E.8)
9. **Trainingsausgaben** (Abschnitt B.2.2) Inhalt der *“1HenseVs2Marin.linear“* Datei. (Listing E.9)

Hauptstartskript: (Listing E.1) ermöglicht, mehrere Experimente nacheinander laufen zu lassen. In diesem Skript werden mehreren Startskripten aufgerufen, gefolgt vom Octave Kommando *“clear all“*. Das Kommando *“clear all“* löscht alle aktuellen Octave Variablen und ist auf dieser Stelle erforderlich, damit es nicht zu Kollisionen und Problemen in den darauf folgenden Experimenten kommen kann. Im Demo-Projekt werden an dieser Stelle 3 Startskripten aufgerufen.

Startskript: (Listing E.2) wird immer exklusiv für ein Experiment erzeugt. In der Zeile 8 des Startskripts wird man in der Regel den Namen der Konfigurationsdatei anpassen müssen. Daher gehe ich davon aus, dass für jedes neue Experiment eine neue Konfigurationsdatei erzeugt wird.

```
Zeile 8:  configFileName = 'config_stt_demo1.txt';
```

Konfigurationsdatei: (Listing E.3) In der Konfigurationsdatei wird festgelegt, unter welchen Bedingungen die STT-Phase durchgeführt wird. Man kann in der Konfigurationsdatei folgende Blöcke erkennen und Informationen entnehmen:

- Zeilen 18 - 28: Gemeinsame Variablen, wie Projektpfad und Stammverzeichnisse des Projekts, die Gültigkeit für das ganze Projekt haben. In der Regel braucht man in diesem Block keine Anpassungen zu machen und Änderungen an den Werten der Variablen durchzuführen.
- Zeilen 31 - 129: In dem Block “Selektieren“ sollen eine Reihe von Werten angepasst werden. Z.B. in der Zeile 36 der Name des Verzeichnisses, das kodierte und extrahierte Daten enthält, und in der Zeile 49 der Name des Experiment-Verzeichnisses.

  ```
  Zeile 36:  cuedVerzeichnis   Demo
  Zeile 49:  expVerzeichnis    Demo_01
  ```

– Zeilen 60 - 82: Auswahl der Trainingsdaten
  Zur Auswahl der Trainingsdaten wird eine 48 Bit lange Bitmap benutzt. Diese Bitmap wird in 8 Teile eingeteilt. Somit ergeben sich die einzelnen Kategorien wie in den Zeilen 63 - 70 beschrieben. Die extrahierten und kodierten Audiodateien werden durch Setzen der einzelnen Bits in der Bitmap ausgewählt. Die längeren Bitmaps sind übersichtshalber durch ein Minuszeichen aufgeteilt worden.

  Zeilen 63 und 64 sagen aus, dass für das SVM Training nur die Audiodateien aus dem Trainingspool benutzt werden.
  Zeile 65 Zuordnung des Audiokanals: linker Kanal - rechter Kanal
  Zeile 66 Die Spieler sind lexikalisch kodiert von links nach rechts: hoffman - oba - ossig - schulz
  Zeile 67 Die Spielart ist lexikalisch kodiert vom links nach rechts: sonor - spitz - warm
  Zeile 68 Die Saiten von links nach rechts: E - H - G - D - A - E
  Zeile 69 Die Bünde von links nach rechts: Bund 1 - Bund 2 - .... - Bund 12
  Zeile 70 Reserve

```
Zeile 63:  trainTrain   1
Zeile 64:  testTrain    0
Zeile 65:  channelTrain 10
Zeile 66:  playerTrain  1111-0000-0000-0000-0000
Zeile 67:  timbreTrain  111
Zeile 68:  stringTrain  1111-11
Zeile 69:  fretTrain    1000-1100-1100
Zeile 70:  reserveTrain 000
```

– Zeilen 80 - 82: Auswahl der einzelnen Teilbereiche der Trainings-Merkmalsvektoren.
  Jede Merkmalsmatrix beinhaltet in den Spalten 1155 Werte. Die Matrixelemente können einzeln bei Nontonalen, und in Gruppen von 50 bzw. 40 Elementen bei Extrahierungsmerkmalen von MFCCs und Harmonischen ausgewählt werden. Durch Setzen der einzelnen Bits in die Bitmaps werden die Elemente in den Spalten der Merkmalsmatrix definiert und zur Weiterbearbeitung selektiert. Genaueres darüber in den Kapiteln 2, 3 und 4.

```
Zeile 80:  bm_nonTonalRegionenTrain 0000-0000-0000-000
Zeile 81:  bm_MFCCAnzahlTrain       0000-0000-00
Zeile 82:  bm_obertoeneAnzahlTrain  1111-1111-1111-1111
```

– Zeilen 87 - 94: Auswahl der Testdaten

Die oben beschriebenen Details (Zeilen 60-82) haben auch hier ihre Gültigkeit. Der Unterschied ist nur, dass es sich hier um Testdaten handelt.

```
Zeile 87:  trainTest   0
Zeile 88:  testTest    1
Zeile 89:  channelTest 10
Zeile 90:  playerTest  1111-0000-0000-0000-0000
Zeile 91:  timbreTest  111
Zeile 92:  stringTest  1111-11
Zeile 93:  fretTest    1000-1100-1100
Zeile 94:  reserveTest 000
```

– Zeilen 104 - 106: Auswahl den einzelnen Teilbereiche der Test-Merkmalsvektoren

```
Zeile 104:  bm_nonTonalRegionenTest 0000-0000-0000-000
Zeile 105:  bm_MFCCAnzahlTest       0000-0000-00
Zeile 106:  bm_obertoeneAnzahlTest  1111-1111-1111-1111
```

– Zeile 110: Auswahl der Datei, die die Skalierungsfaktoren konfiguriert
Hier wird der Name der Skalierungsfaktoren-Datei eingetragen, die für das aktuelle Experiment verwendet wird. Die Datei wird ohne ihre Endung eingetragen.

```
Zeile 110:  skalierungswerte getScalingValueDefault
```

- Zeilen 131 - 135 Trainieren
Hier wird der Name der SVM Trainingsoptions-Datei eingetragen, die für das aktuelle Experiment verwendet wird. Die Datei wird ohne ihre Endung eingetragen.

  ```
  Zeile 135:  trainOptionen trainOptionsDefault
  ```

- Zeilen 137 - 141 Testen
Hier wird der Name der SVM Testoptions-Datei eingetragen, die für das aktuelle Experiment verwendet wird. Die Datei wird ohne ihre Endung eingetragen.

  ```
  Zeile 141:  testOptionen testOptionsDefault
  ```

Skalierungsparameter: (Listing E.4) Hier können die Skalierungsfaktoren für jeden einzelnen von 10 MFCCs (Zeile 19) und 16 Harmonischen (Zeile 22) in Merkmalsvektoren festgelegt werden.

```
Zeile 18:  scalValuesMFCC = ...
Zeile 19:  [1000 , 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
   1000];
Zeile 20:  %1      2     3     4     5     6     7     8     9     10
Zeile 21:  scalValuesTemporal = ...
Zeile 22:  [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
   1000, 1000, 1000, 1000, 1000, 1000];
Zeile 23:  %1     2     3     4     5     6     7     8     9     10
       11    12    14   15    16
```

Optionen für SVM-Training: (Listing E.5) Auswahl der Optionen für die SVM Trainingsphase
Es ist nur notwendig, die Parameter einzutragen, die von SVM Default-Einstellungen abweichen. Sonst werden die Default-Werte genommen.

```
Zeile 19: optionsLinear = svmlopt('Verbosity', 1, 'C', 1.0, 'Biased',
    1, 'Kernel', 0);
```

Optionen für SVM-Test: (Listing E.6) Auswahl der Optionen für die SVM Testphase
Es ist nur notwendig die Parameter einzutragen, die von SVM Default-Einstellungen abweichen. Sonst werden die Default-Werte genommen.

```
Zeile 19: options = svmlopt('Verbosity', 1);
```

Log-Datei: (Listing E.7) Während jeder STT-Phase wird eine *"selektResult.txt"* Log-Datei erzeugt. Dort werden Einträge und Meldungen über die Ereignisse eingetragen, die in der STT-Phase passieren. Man kann in der Log-Datei folgende Blöcke erkennen und folgende Informationen entnehmen:

- Zeilen 1 - 8: Einige statistische Daten wie Benutzername, Datum und Uhrzeit, die verwendete Konfigurationsdatei, das Audiodateien-Verzeichnis und das Verzeichnis für kodierte und extrahierte Daten.
- Zeilen 10 - 22: Validierung der vorhandenen und zur Laufzeit erzeugten Verzeichnisse.
- Zeilen 25 - 54: Bitmaps zur Auswahl der Audiodateien und Teilbereiche der Merkmalsvektoren, die in der Trainings- und Testphase benutzt werden.
- Zeilen 56 - 84: Skalierungsfaktoren für 10 MFCCs- und 16 Harmonischen- Teilbereiche des Merkmalsvektors.

- Zeilen 86 - 96: Statistik über selektierte Trainings- und Test-Audiodateien
  Aufgelistet ist die Anzahl der Audiodateien pro Gitarre und Train- bzw. Testphase und die Länge der Selektionsphase in Sekunden.
- Zeilen 100 - 124: SVM Trainingsoptionen und Länge der Trainingsphase in Sekunden.
- Zeilen 128 - 152: SVM Testoptionen und die Länge der Testphase in Sekunden.

Klassifikaionsergebnsse: (Listing E.8) Am Ende des Klassifikationsprozesses wird für jedes Experiment (Demo_01, Demo_02 und Demo_03) die Datei “classifikationResult.tex“ erzeugt. In der Datei “classifikationResult.tex“ wird die Erkennungsrate in % ausgewiesen. Außerdem wird auch eine Statistik über die richtig und falsch klassifizierten Töne für jede einzelne Gitarre angelegt.

- Zeile 1: Gesamt-Erkennungsrate
- Zeilen 3 - 4: Gitarrenbezogene Erkennungsrate
- Zeilen 5 - 10: Statistik über die falsch erkannten Tönen für alle drei Gitarren
- Zeilen 11 - 40: Knoten-bezogene Statistik über die richtig und falsch erkannten Töne für alle drei Gitarren

Trainingsausgaben: (Listing E.9) In der Beispiel-Datei “1HenseVs2Marin.linear“ wird die Ausgabe der Trainingsphase abgespeichert. Die abgespeicherten Daten ermöglichen es nachträglich, die Qualität der trainierten SVMs zu überprüfen. Diese Datei enthält eine Reihe von Informationen über die Trainings-Phase. Hier einen Auszug:

- Dauer der Trainingphase in Sekunden
- Dauer der Trainingphase in der Anzahl der benötigten Iterationen
- Anzahl der Supportvektoren, die berechnet wurden

B.3. Kombinieren von CE- und STT-Phase

Alles was in den beiden vorherigen Abschnitten im einzelnen über die CE- und die STT-Phase geschrieben worden ist, gilt auch für den Fall, dass beide Phasen aus einem Startskript nacheinander aufgerufen werden. Man soll lediglich auf zwei Dinge achten. Erstens sollen sich alle Konfigurationsdateien und Startskripten für beide Phasen in demselben Verzeichnis befinden. Zweitens soll das Hauptstartskript um die Einträge des Startskripts für die CE-Phase erweitert werden. Siehe Softwarelistings F.1.

B.4. Überblick über die Software Implementierung

Die entwickelte Softwareumgebung kann in der aktuellen Version nur die Audiodateien von drei Gitarren klassifizieren. Einzelheiten darüber sind im Kapitel 4, (Abschnitt “Überblick“) zu finden.

B.5. DVD Inhalt

DVD Root/ . . . . . DVD Root Verzeichnis
- **Auswertungs_Tools/** . . . . . Beinhaltet die Auswertungs-Tools, die im Kapitel C beschrieben wurden
- **Bachelorarbeit_PDF/** . . . . . Beinhaltet die Bachelorarbeit im pdf-Format
- **DemoProjekte/** . . . . . Beinhaltet die Demo-Projekte, die in den Kapiteln B und C beschrieben wurden
- **Gitarrenerkennung/** . . . . Ist Main-Projektverzeichnis des Klassifikationsverfahrens, ist im Kapitel A beschrieben worden.

C. Auswertungs-Tools

Bei der Anzahl von Experimenten (über 150 Experimente), die in dieser Arbeit durchgeführt wurden, zeigte sich die Auswertung der Ergebnisse als mühsam und fehlerträchtig. Die Ergebnisse eines Experiments in Form einer Tabelle oder eines Diagramms darzustellen, hat sich als schwierig gezeigt, weil die Experimente eine Menge an Konfigurationsparametern haben, die man berücksichtigen muss, um Verlauf und Ergebnisse eines Experiments präzise zu beschreiben.

Der Auswertungsprozess wurde beschleunigt und die Auswertungsfehler wurden minimiert durch die Entwicklung von Octave Funktionen, die relevante Dateien (Log- und Result-Dateien) einlesen, auswerten und die gesuchten Parameter in einer Auswertungsdatei abspeichern. Die Funktionsweise wird am Beispiel von drei Demo-Experimenten (Experiment01_Demo, Experiment02_Demo und Experiment03_Demo) erklärt, die sich in dem *“Auswertung_Demo“* Verzeichnis befinden.

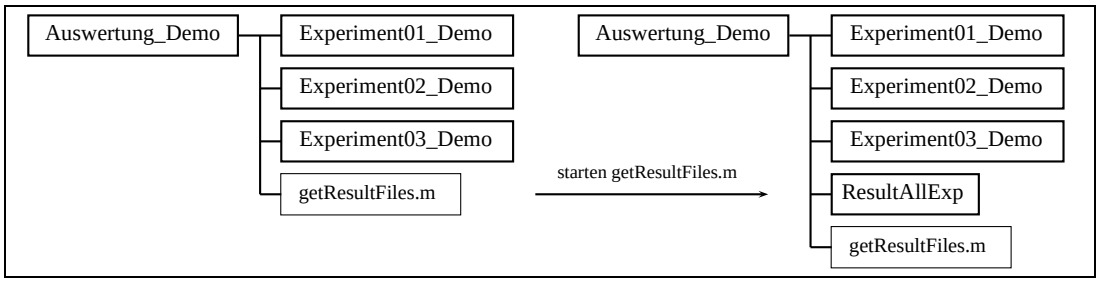
C.1. Result-Dateien kopieren

Man stellt sich die folgende Situation vor. Es sind mehreren Experimente (z.B 20 Experimente) durchgeführt worden. Pro Experiment sollen mindestens zwei Result-Dateien ausgewertet werden. Die Auswertung bezieht sich dabei auf mehrere Konfigurationsparameter pro Result-Datei. Als erstes ist es sinnvoll, die benötigten Dateien in ein Verzeichnis zu kopieren und somit die Auswertung der Dateien wesentlich zu vereinfachen.

Um ein Experiment vollständig auswerten zu können, braucht man die Informationen, die in den zwei Dateien der STT-Phase abgespeichert sind. Das sind *“classifikationResult.txt“* (mit dem Pfad z.B. *“Experiment01_Demo/SVM/Testoutput“*) und *“selektResult.txt“* Datei, die sich direkt im Experiment-Verzeichnis (z.B *“Experiment01_Demo/“*) befindet.

Folgende Schritte sind zu machen, um Result-Dateien in ein Verzeichnis zu kopieren:

Abbildung C.1.: Kopieren von Result-Dateien der ausgewählten Experimente

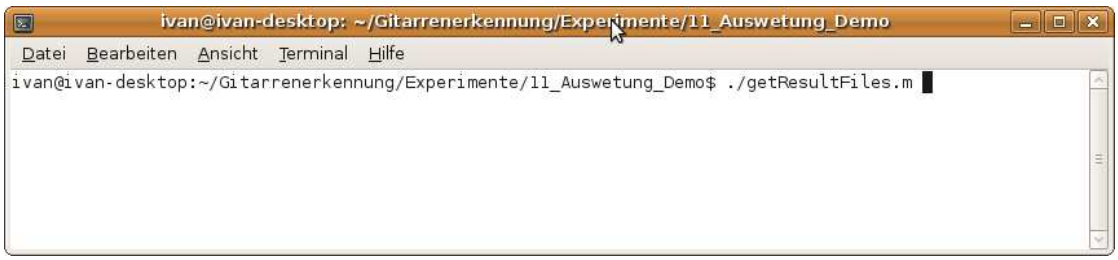


1. Das *"getResultFiles.m"* Skript und die Experimenten-Verzeichnisse, die man auswerten möchte, in ein leeres Verzeichnis (z.B *"Auswertung_Demo"*) kopieren. (Siehe die linke Seite der Abbildung C.1.)
2. Die "getResultFiles.m" Skript im Terminal aufrufen. (Siehe Abbildung C.2.)
3. Das *"ResultAllExp"* Verzeichnis wird erzeugt. (Siehe die rechte Seite der Abbildung C.1.) In dieses Verzeichnis werden die zwei Result-Dateien pro Experiment hineinkopiert. Das heißt, dass in dieses Verzeichnis insgesamt sechs Dateien kopiert werden. Die kopierten Result-Dateien werden umbenannt. Jede Datei bekommt den Namen nach dem folgenden Schema vergeben:

   `<Name des Experiments>\_<Name der Result-Dateien>`

   So sind die Result-Dateien eindeutig zu den bestimmten Experimenten zuweisbar. (Siehe das Listing C.1.)

Abbildung C.2.: Starten der "getResultFiles.m" Funktion



Listing C.1: Umbenannte Result-Dateien

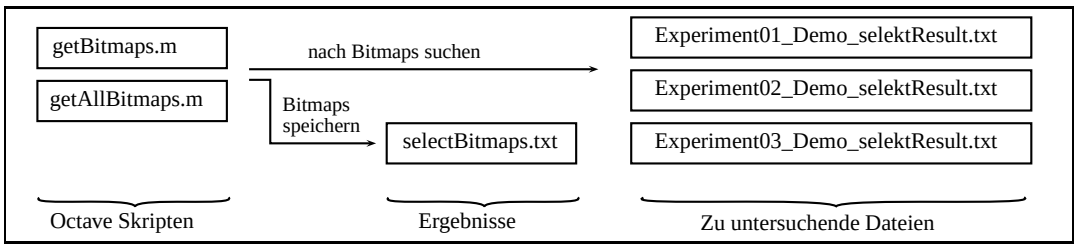
```
Experiment01_Demo_classifikationResult.txt
Experiment02_Demo_classifikationResult.txt
Experiment03_Demo_classifikationResult.txt
```

```
Experiment01_Demo_selektResult.txt
Experiment02_Demo_selektResult.txt
Experiment03_Demo_selektResult.txt
```

C.2. Result-Dateien auswerten

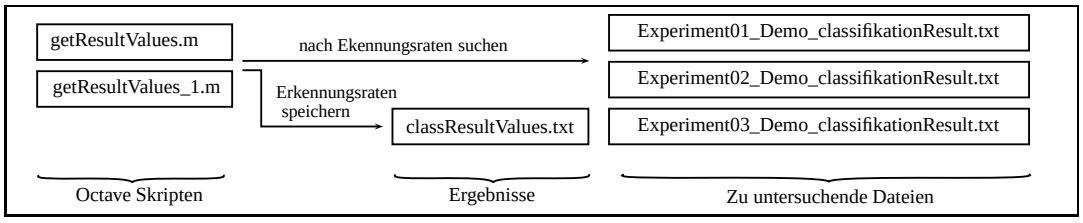
Die Dateien aus der Listing C.1 werden gezielt nach bestimmten Parametern durchsucht. So suchen die Skripten *"getBitmaps.m"* und *"getAllBitmaps.m"* nach den Bitmaps in den *"xxx_selectResult.txt"* Dateien und speichern die Ergebnisse in der *"selectBitmaps.txt"* Datei ab. Siehe Abbildung C.3. Es wird nach Bitmaps gesucht, die zur Auswahl von Audiovektoren und den extrahierten Merkmalen gebraucht werden. Die Bitmaps werden in einer Form abgespeichert, die mit den Latex Tabellen kompatibel ist. Dann kann man diese Daten durch Copy/Paste in ein Latex Dokument hineinkopieren.

Abbildung C.3.: Suchen nach den Bitmaps



Die Abbildung C.4 zeigt die Dateien, die bei der Suche nach den Erkennungsraten der Experimente beteiligt sind. Die Erkennungsraten werden in verschiedenen Formaten (für Excel-, Latex-Dokumente) vorbereitet.

Abbildung C.4.: Suchen nach den Erkennungsraten



Außerdem wurden mehreren Skripten entwickelt, die auf gleiche Art und Weise funktionieren wie die in der Abbildung C.3 dargestellten Verfahren. Diese Skripten (z.B. *"getSelektValues_MFCC_1.m"*) sind für jede Experimenten-Gruppe entwickelt worden. Die Ergebnisse der Suche werden in der *"selectValues.txt"* Datei abgespeichert.

D. Softwarelistings CE-Phase

D.1. CE-Phase Startskript “start_ce_demo.m“

Listing D.1: Startsktipt start_ce_demo.m

```
#!/usr/bin/octave --silent

% Octave load-path erweiterung auf "Software" Verzeichnis und
% alle Software Unterverzeichnisse
addpath (genpath('~/Gitarrenerkennung/Software/'));

% Lade Konfigurationsdatei und speichere Variablen in Cell-Array
configFileName = 'config_ce_demo.txt';
global actualConfigFileName = strcat(pwd, '/', configFileName);
[confVariable, confWert_01, confWert_02] = textread(actualConfigFileName, '%s %s %s');
global confFile = [[confVariable(:,1)], [confWert_01(:,1)], [confWert_02(:,1)]];

startCodierenUndExtrahieren;
```

D.2. CE-Phase Konfigurationsdatei “config_ce_demo.txt“

Listing D.2: Konfigurationsdatei config_ce_demo.txt

```
% Dies ist Konfigurationsdatei fuer CE-Phase (Codierungs- und Extrahierungsphase).
% Konfigurationsdatei sollte keine leere Zeilen haben und sollte auch mit eine
% leere Zeile nicht beendet werden. Grund dafuer ist dass die Octave "textread()"
% Funktion, die fuer Einlesen der Konfigurationsdatei verwendet wird, mit
% "leeren Zeilen" nicht umgehen kann. Lesen der Konfigurationsdatei wird
% abgebrochen durch eine Fehlermeldung.
% Vorschlag fuer vergebung des Namens der Konfigurationsdateien fuer CE-Phase:
% config_ce_xyz.txt
% config: bezeichnet das sich um ein Konfigurationsdatei handelt
% ce:     kuerzel fuer Codierungs- und Extrahierungsphase
% xyz:    Platzhalter fur Name der Codierungs- und Extrahierungsphase
%
% ------------------------------------------------------------------------
% --------------------- GEMEINSAME VARIABLEN -----------------------------
% ------------------------------------------------------------------------
%
% Einzelne Verzeichnisse die Projektpfad bilden
% Wert (Name) der "userVerzeichnis" Variable soll Angepasst werden
```

```
19 homeVerzeichnis      home
20 userVerzeichnis      beliebig
21 projektVerzeichnis   Gitarrenerkennung
22 %
23 % Stammverzeichnisse des Projekts
24 audioStammVerzeichnis   Audiodaten
25 cuedStammVerzeichnis    CodierteUndExtrahierteDaten
26 experStammVerzeichnis   Experimente
27 softStammVerzeichnis    Software
28 %
29 % Software Unterverzeichnisse
30 codierenUndExtrahieren  CodierenUndExtrahieren
31 selektieren             Selektieren
32 testen                  Testen
33 trainieren              Trainieren
34 %
35 % ------------------------------------------------------------------------
36 % -------------------- CODIEREN UND EXTRAHIEREN --------------------------
37 % ------------------------------------------------------------------------
38 %
39 % Wert (Name) der "audioVerzeichnis" Variable soll Angepasst werden
40 audioVerzeichnis   Demo
41 testAudio          Test
42 trainAudio         Train
43 %
44 % Diese Verzeichnisse werden in "Codieren und Extahieren-Phase" erzeugt
45 % Wert (Name) der "cuedVerzeichnis" Variable soll Angepasst werden
46 cuedVerzeichnis      Demo
47 cuedTestCodiert      TestCodiert
48 cuedTestExtrahiert   TestExtrahiert
49 cuedTrainCodiert     TrainCodiert
50 cuedTrainExtrahiert  TrainExtrahiert
51 %
52 % Merkmalsextraktion Configurationsparameter
53 % ---- Default Werte ------------------------------------------------------
54 % Nontonale Spectrum Parameter -------------------------------------------
55 % nonTonalRegionen        15 -------------------------------------------
56 %
57 % Zeitlicher Verlauf der Harmonischen Parameter ------------------------
58 % fensterGroesse          4096 -------------------------------------------
59 % obertoeneAnzahl         16 -------------------------------------------
60 % spaltenanzahlTemporal   40 -------------------------------------------
61 %
62 % MFCC Parameter ---------------------------------------------------------
63 % spaltenanzahlMFCC       50 -------------------------------------------
64 % framerate               25 -------------------------------------------
65 % MFCC_anzahl             10 -------------------------------------------
66 % ---- Ende Default Werte -------------------------------------------------
67 %
68 % Nontonale Spectrum Parameter
69 nonTonalRegionen    15
70 %
71 % MFCC Parameter
72 spaltenanzahlMFCC   50
73 framerate           25
74 MFCCAnzahl          10
75 %
76 % Zeitlicher Verlauf der Harmonischen Parameter
```

```
77 fensterGroesse        4096
78 obertoeneAnzahl       16
79 spaltenanzahlTemporal 40
80 %
81 % Ende der Konfigurationsdatei
```

D.3. CE-Phase Log-Datei “codExtrResult.txt“

Listing D.3: Log-Datei codExtrResult.txt

```
 1 % Codierungs- und Extrahierungsphase Statistik
 2 Benutzername: "ivan"
 3 Datum: "03-Nov-2009␣12:50:26"
 4 CE Konfigurationsdatei: "/home/ivan/Gitarrenerkennung/Software/Startskripten/Ce/Demo/
       config_ce_demo.txt"
 5 Audiodaten Quellverzeichnis: "/home/ivan/Gitarrenerkennung/Audiodaten/Demo"
 6 CUED Zielverzeichnis: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten/Demo"
 7 %
 8 % Projekt Verzeichnisse validierung
 9 Variable homeVerzeichnis: "home"  ist OK.
10 Variable userVerzeichnis: "ivan"  ist OK.
11 Variable projektVerzeichnis: "Gitarrenerkennung"  ist OK.
12 Variable projektVerzeichnisPfad: "/home/ivan/Gitarrenerkennung"  ist OK.
13 Variable cuedStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/
       CodierteUndExtrahierteDaten"  ist OK.
14 Variable experStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/Experimente"  ist OK.
15 Variable softStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/Software"  ist OK.
16 Variable audioVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/Audiodaten/Demo"  ist OK.
17 Variable trainAudioPfad: "/home/ivan/Gitarrenerkennung/Audiodaten/Demo/Train"  ist OK.
18 Variable testAudioPfad: "/home/ivan/Gitarrenerkennung/Audiodaten/Demo/Test"  ist OK.
19 Variable cuedVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten
       /Demo/TrainCodiert"  ist OK.
20 Variable cuedTrainCodiertPfad: "/home/ivan/Gitarrenerkennung/
       CodierteUndExtrahierteDaten/Demo/TrainCodiert"  ist OK.
21 Variable cuedTrainExtrahiertPfad: "/home/ivan/Gitarrenerkennung/
       CodierteUndExtrahierteDaten/Demo/TrainExtrahiert"  ist OK.
22 Variable cuedTestCodiertPfad: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten
       /Demo/TestCodiert"  ist OK.
23 Variable cuedTestExtrahiertPfad: "/home/ivan/Gitarrenerkennung/
       CodierteUndExtrahierteDaten/Demo/TestExtrahiert"  ist OK.
24 %
25 % Codierungs- und Extrahierungsphase.
26 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Train/
       Hense/
27 Spieler: hoffmann  ist 3  mal gefunden.
28 Spieler: oba  ist 3  mal gefunden.
29 Gesammt ist 6  Dateien gefunden.
30 Gesammt sind 2  Spieler gefunden.
31 %
32 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Train/
       Marin/
33 Spieler: hoffmann  ist 6  mal gefunden.
34 Spieler: oba  ist 6  mal gefunden.
35 Gesammt ist 12  Dateien gefunden.
36 Gesammt sind 2  Spieler gefunden.
37 %
```

```
38 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Train/
       Wichmann/
39 Spieler: hoffman  ist 9  mal gefunden.
40 Spieler: oba  ist 6  mal gefunden.
41 Spieler: ossig  ist 3  mal gefunden.
42 Gesammt ist 18  Dateien gefunden.
43 Gesammt sind 3  Spieler gefunden.
44 %
45 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Test/
       Hense/
46 Spieler: hoffman  ist 9  mal gefunden.
47 Spieler: oba  ist 9  mal gefunden.
48 Spieler: ossig  ist 3  mal gefunden.
49 Spieler: hoffmann  ist 3  mal gefunden.
50 Gesammt ist 24  Dateien gefunden.
51 Gesammt sind 4  Spieler gefunden.
52 %
53 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Test/
       Marin/
54 Spieler: hoffman  ist 9  mal gefunden.
55 Spieler: oba  ist 12  mal gefunden.
56 Spieler: ossig  ist 3  mal gefunden.
57 Spieler: hoffmann  ist 6  mal gefunden.
58 Gesammt ist 30  Dateien gefunden.
59 Gesammt sind 4  Spieler gefunden.
60 %
61 % Codierungsphase Ergebnisse fuer: /home/ivan/Gitarrenerkennung/Audiodaten/Demo/Test/
       Wichmann/
62 Spieler: hoffman  ist 12  mal gefunden.
63 Spieler: oba  ist 12  mal gefunden.
64 Spieler: ossig  ist 6  mal gefunden.
65 Spieler: hoffmann  ist 6  mal gefunden.
66 Gesammt ist 36  Dateien gefunden.
67 Gesammt sind 4  Spieler gefunden.
68 %
69 % Extahierungsphase Configurationswerte
70 % Nontonale Spectrum Parameter
71 nonTonalRegionen 15
72 % Zeitlicher Verlauf der Harmonischen Parameter
73 fensterGroesse 4096
74 obertoeneAnzahl 16
75 spaltenanzahlTemporal 40
76 % MFCC Parameter
77 spaltenanzahlMFCC 50
78 framerate 25
79 MFCCAnzahl 10
80 % Laenge der Codierungs- und  Extahierungsphase: 87 Sek
```

E. Softwarelistings STT-Phase

E.1. STT-Phase Hauptstartskript “start_stt_demo_all.m“

Listing E.1: Hauptstartskript start_stt_demo_all.m

```
#!/usr/bin/octave --silent

% Octave load-path erweiterung auf "Software" Verzeichnis und
% alle Software Unterverzeichnisse
addpath (genpath('~/Gitarrenerkennung/Software/'));

start_stt_demo1;
clear all;
start_stt_demo2;
clear all;
start_stt_demo3;
clear all;
```

E.2. STT-Phase Startskript “start_stt_demo1.m“

Listing E.2: Startskript start_stt_demo1.m

```
#!/usr/bin/octave --silent

% Octave load-path erweiterung auf "Software" Verzeichnis und
% alle Software Unterverzeichnisse
addpath (genpath('~/Gitarrenerkennung/Software/'));

% Lade Konfigurationsdatei und speichere Variablen in Cell-Array
configFileName = 'config_stt_demo1.txt';
global actualConfigFileName = strcat(pwd, '/', configFileName);
[confVariable, confWert_01, confWert_02] = textread(actualConfigFileName, '%s␣%s␣%s');
global confFile = [[confVariable(:,1)], [confWert_01(:,1)], [confWert_02(:,1)]];

startSelektieren;
learn;
testdagsvm;
```

E.3. STT-Phase Konfigurationsdatei “config_stt_demo1.txt“

Listing E.3: Konfigurationsdatei config_stt_demo1.txt

```
% Dies ist Konfigurationsdatei fuer STT-Phase (Selektions-, Trainings- und
% Testphase).
% Konfigurationsdatei sollte keine leere Zeilen haben und sollte auch mit eine
% leere Zeile nicht beendet werden. Grund dafuer ist dass die Octave "textread()"
% Funktion, die fuer Einlesen der Konfigurationsdatei verwendet wird, mit
% "leeren Zeilen" nicht umgehen kann. Lesen der Konfigurationsdatei wird
% abgebrochen durch eine Fehlermeldung.
% Vorschlag fuer vergebung des Namens der Konfigurationsdateien fuer CE-Phase:
% config_stt_xyz.txt
% config: bezeichnet das sich um ein Konfigurationsdatei handelt
% stt:    kuerzel fuer Selektions-, Trainings- und Testphase
% xyz:    Platzhalter fuer Name der Selektions-, Trainings- und Testphase
%
% ------------------------------------------------------------------------
% --------------------- GEMEINSAME VARIABLEN -----------------------------
% ------------------------------------------------------------------------
%
% Einzelne Verzeichnisse die Projektpfad bilden
% Wert (Name) der "userVerzeichnis" Variable soll Angepasst werden
homeVerzeichnis        home
userVerzeichnis        beliebig
projektVerzeichnis     Gitarrenerkennung
%
% Stammverzeichnisse des Projekts
audioStammVerzeichnis  Audiodaten
cuedStammVerzeichnis   CodierteUndExtrahierteDaten
experStammVerzeichnis  Experimente
softStammVerzeichnis   Software
%
% ------------------------------------------------------------------------
% ------------------------- SELEKTIEREN ----------------------------------
% ------------------------------------------------------------------------
%
% In "CODIEREN UND EXTRAHIEREN" Phase erzeugte und als output benutzte Verzei-
% chnisse werden in "SELEKTIEREN" Phase als input (Datenquelle) benutzt
cuedVerzeichnis        Demo
cuedTestCodiert        TestCodiert
cuedTestExtrahiert     TestExtrahiert
cuedTrainCodiert       TrainCodiert
cuedTrainExtrahiert    TrainExtrahiert
%
% Klangkategorie binaer codieren in einen 48 Bit Nummer
% ------------------------------------------------------------------------
% Train | Test  | Kanal  | Spieler | Klang  | Saite  | Bund   | Res
% 1 Bit | 1 Bit | 2 Bit  | 20 Bit  | 3 Bit  | 6 Bit  | 12 Bit | 3 Bit
% ------------------------------------------------------------------------
%
% Diese Verzeichnisse werden in "Selektieren-Phase" erzeugt
expVerzeichnis         Demo_01
expTrainFiles          TrainSelektFiles
expTrainMartix         TrainSelektMatrix
```

```
52  expTestFiles           TestSelektFiles
53  expTestMartix          TestSelektMatrix
54  %
55  svmVerzeichnis         SVM
56  svmTestoutput          Testoutput
57  svmTrainierteSVMs      TrainierteSVMs
58  svmTrainingsoutput     Trainingsoutput
59  %
60  % Trainingsdaten auswaehlen
61  % Fuer jede Klangkategorie steht folgende Anzahl an Bit-Stellen zum Verfuegung
62  % 1: Klangkategorie auswaehlen, 0: Klangkategorie nicht auswaehlen
63  trainTrain     1
64  testTrain      0
65  channelTrain   10
66  playerTrain    1111-0000-0000-0000-0000
67  timbreTrain    111
68  stringTrain    1111-11
69  fretTrain      1000-1100-1100
70  reserveTrain   000
71  %
72  % Extrahierte Merkmale werden hier ausgewaehlt
73  % 1: Merkmal auswaehlen, 0: Merkmal nicht auswaehlen
74  % Default Werte alles ausgewaehlt ------------------------
75  %bm_nonTonalRegionenTrain  1111-1111-1111-111   15 -------
76  %bm_MFCCAnzahlTrain        1111-1111-11         10 -------
77  %bm_obertoeneAnzahlTrain   1111-1111-1111-1111  16 -------
78  % ----------------------------------------------------------
79  % Bitmaps
80  bm_nonTonalRegionenTrain  0000-0000-0000-000
81  bm_MFCCAnzahlTrain        0000-0000-00
82  bm_obertoeneAnzahlTrain   1111-1111-1111-1111
83  %
84  % Testdaten auswaehlen
85  % Fuer jede Klangkategorie steht folgende Anzahl an Bit-Stellen zum Verfuegung
86  % 1: Klangkategorie auswaehlen, 0: Klangkategorie nicht auswaehlen
87  trainTest     0
88  testTest      1
89  channelTest   10
90  playerTest    1111-0000-0000-0000-0000
91  timbreTest    111
92  stringTest    1111-11
93  fretTest      1000-1100-1100
94  reserveTest   000
95  %
96  % Extrahierte Merkmale werden hier ausgewaehlt
97  % 1: Merkmal auswaehlen, 0: Merkmal nicht auswaehlen
98  % Default Werte alles abgewaehlt ------------------------
99  %bm_nonTonalRegionenTest  0000-0000-0000-000    15 -------
100 %bm_MFCCAnzahlTest        0000-0000-00         10 -------
101 %bm_obertoeneAnzahlTest   0000-0000-0000-0000   16 -------
102 % ----------------------------------------------------------
103 % Bitmaps
104 bm_nonTonalRegionenTest  0000-0000-0000-000
105 bm_MFCCAnzahlTest        0000-0000-00
106 bm_obertoeneAnzahlTest   1111-1111-1111-1111
107 %
108 % Funktion "getScalingValueXXX" auswaehlen.
109 % XXX: ist Platzhalter fuer zusaetzlichen Teil des Namens der Funktion
```

```
110 skalierungswerte  getScalingValueDefault
111 %
112 % Gitarren fÃ¼r Selektions- Trainings- und Testphase auswaehlen
113 auswahlArt     1
114 %
115 % AlleGitarren auswaehlen [auswahlArt == 1]
116 gitarAnzahl     3
117 %
118 % Berich auswaehlen StartIndex bis EndIndex [auswahlArt == 2]
119 gitarStartIndex   0
120 gitarEndIndex    0
121 %
122 % Bestimmte Gitarren auswaehlen Ã¼ber 8 Bit-Wort Index [auswahlArt == 3]
123 gitarIndexWort_01    0000-0000
124 gitarIndexWort_02    0000-0000
125 gitarIndexWort_03    0000-0000
126 %
127 % Berich auswaehlen ungeraden oder geraden Index [auswahlArt == 4]
128 gitarUngeradenIndex 0
129 gitarGeradenIndex  0
130 % --------------------------------------------------------------------------
131 % ------------------------- TRAINIEREN -------------------------------------
132 % --------------------------------------------------------------------------
133 %% Funktion "trainOptionsXXX" auswaehlen.
134 % XXX: ist Platzhalter fuer zusaetzlichen Teil des Namens der Funktion
135 trainOptionen  trainOptionsDefault
136 % --------------------------------------------------------------------------
137 % ------------------------- TESTEN -----------------------------------------
138 % --------------------------------------------------------------------------
139 %%% Funktion "testOptionsXXX" auswaehlen.
140 % XXX: ist Platzhalter fuer zusaetzlichen Teil des Namens der Funktion
141 testOptionen  testOptionsDefault
142 %
```

E.4. STT-Phase Skalierungsparameter “getScalingValueDefault.m“

Listing E.4: Skalierungsparameter getScalingValueDefault.m

```
1  function scalingValue = getScalingValueDefault(arrayIndex, scalTyp)
2
3  % --------------------------------------------------------------------------
4  % Liefert Skalierungswerte fuer MFCC und Temporal Klassifizierung
5  %
6  % Inputs:
7  % arrayIndex: Skalar, MFCC und Temoral Arrayindex
8  % scalTyp: String, Skalierungsmodus moegliche Werte sind "mfcc" und "temporal"
9  %
10 % Outputs:
11 % scalingValue: Skalar, Skalierungswert
12 %
13 % Author:
14 % Ivan Turkalj ivan.turkalj@haw-hamburg.de
15 % July 2009
16 % --------------------------------------------------------------------------
```

```
17
18   scalValuesMFCC =...
19   [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000];
20   % 1     2     3     4     5     6     7     8     9     10
21   scalValuesTemporal =...
22   [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
         1000, 1000];
23   %  1    2     3     4     5     6     7     8     9     10    11    12    13    14
         15    16
24
25   scalingValue = 1;
26
27   if strcmp(scalTyp, 'mfcc') == 1
28     scalingValue = scalValuesMFCC(arrayIndex);
29   elseif strcmp(scalTyp, 'temporal') == 1
30     scalingValue = scalValuesTemporal(arrayIndex);
31   else
32    scalingValue = 1;
33   end %if
34
35 endfunction
```

E.5. STT-Phase Optionen für SVM-Training “trainOptionsDefault.m“

Listing E.5: Optionen für SVM-Training trainOptionsDefault.m

```
1  function optionsLinear = trainOptionsDefault()
2
3  % -------------------------------------------------------------------------
4  % Ruft Trainings SVM-Optionen "svmlopt()" Funktion auf. Diese Funktion wird mit
5  % hilfe Octave "eval()"  Funktion aufgerufen. Welche "trainOptionsXXX()"
6  % Funktion aufgerufen wird, wird in Konfiguratzionsdatei festgelegt.
7  %
8  % Inputs:
9  % keine
10 %
11 % Outputs:
12 % keine
13 %
14 % Author:
15 % Ivan Turkalj ivan.turkalj@haw-hamburg.de
16 % July 2009
17 % -------------------------------------------------------------------------
18
19   optionsLinear = svmlopt('Verbosity', 1, 'C', 1.0, 'Biased', 1, 'Kernel', 0);
20
21 endfunction
```

E.6. STT-Phase Optionen für SVM-Test “testOptionsDefault.m“

Listing E.6: Optionen für SVM-Test testOptionsDefault.m

```
function options = testOptionsDefault()

% ------------------------------------------------------------------------
% Ruft Test SVM-Optionen "svmlopt()" Funktion auf. Diese Funktion wird mit
% hilfe Octave "eval()"  Funktion aufgerufen. Welche "trainOptionsXXX()"
% Funktion aufgerufen wird, wird in Konfiguratzionsdatei festgelegt.
%
% Inputs:
% keine
%
% Outputs:
% keine
%
% Author:
% Ivan Turkalj ivan.turkalj@haw-hamburg.de
% July 2009
% ------------------------------------------------------------------------

  options = svmlopt('Verbosity', 1);

endfunction
```

E.7. STT-Phase Log-Datei “selektResult.txt“

Listing E.7: Log-Datei selektResult.txt

```
% Experiment Statistik
Benutzername: "ivan"
Datum: "03-Nov-2009␣13:06:23"
Experimentname: "/home/ivan/Gitarrenerkennung/Experimente/Demo_01"
STT Konfigurationsdatei: "/home/ivan/Gitarrenerkennung/Software/Startskripten/Stt/Demo/
    config_stt_demo1.txt"
CE codExtrResult.txt Datei: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten/
    Demo/codExtrResult.txt"
Skalierungswertedatei: "getScalingValueDefault"
SVM Trainingsoptionendatei: "trainOptionsDefault"
%
% Projekt Verzeichnisse validierung
Variable homeVerzeichnis: "home"  ist OK.
Variable userVerzeichnis: "ivan"  ist OK.
Variable projektVerzeichnis: "Gitarrenerkennung"  ist OK.
Variable projektVerzeichnisPfad: "/home/ivan/Gitarrenerkennung"  ist OK.
Variable cuedStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/
    CodierteUndExtrahierteDaten"  ist OK.
Variable experStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/Experimente"  ist OK.
Variable softStammVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/Software"  ist OK.
Variable cuedVerzeichnisPfad: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten
    /Demo/TrainCodiert"  ist OK.
Variable cuedTrainCodiertPfad: "/home/ivan/Gitarrenerkennung/
    CodierteUndExtrahierteDaten/Demo/TrainCodiert"  ist OK.
Variable cuedTrainExtrahiertPfad: "/home/ivan/Gitarrenerkennung/
    CodierteUndExtrahierteDaten/Demo/TrainExtrahiert"  ist OK.
Variable cuedTestCodiertPfad: "/home/ivan/Gitarrenerkennung/CodierteUndExtrahierteDaten
    /Demo/TestCodiert"  ist OK.
```

```
22 Variable cuedTestExtrahiertPfad: "/home/ivan/Gitarrenerkennung/
       CodierteUndExtrahierteDaten/Demo/TestExtrahiert"  ist OK.
23 %
24 % Flags und Variablen validierung
25 %
26 % Train Variablen
27 Variable trainTrain: "1"  ist OK.
28 Variable testTrain: "0"  ist OK.
29 Variable channelTrain: "10"  ist OK.
30 Variable playerTrain: "11110000000000000000"  ist OK.
31 Variable timbreTrain: "111"  ist OK.
32 Variable stringTrain: "111111"  ist OK.
33 Variable fretTrain: "100011001100"  ist OK.
34 Variable reserveTrain: "000"  ist OK.
35 %
36 % Test Variablen
37 Variable trainTest: "0"  ist OK.
38 Variable testTest: "1"  ist OK.
39 Variable channelTest: "10"  ist OK.
40 Variable playerTest: "11110000000000000000"  ist OK.
41 Variable timbreTest: "111"  ist OK.
42 Variable stringTest: "111111"  ist OK.
43 Variable fretTest: "100011001100"  ist OK.
44 Variable reserveTest: "000"  ist OK.
45 %
46 % Train Merkmalsbitmaps
47 Variable bm_nonTonalRegionenTrain: "000000000000000"  ist OK.
48 Variable bm_MFCCAnzahlTrain: "0000000000"  ist OK.
49 Variable bm_obertoeneAnzahlTrain: "1111111111111111"  ist OK.
50 %
51 % Test Merkmalsbitmaps
52 Variable bm_nonTonalRegionenTest: "000000000000000"  ist OK.
53 Variable bm_MFCCAnzahlTest: "0000000000"  ist OK.
54 Variable bm_obertoeneAnzahlTest: "1111111111111111"  ist OK.
55 %
56 % MFCC Skalierungswerte
57 Skalierungswert 1: "1000"
58 Skalierungswert 2: "1000"
59 Skalierungswert 3: "1000"
60 Skalierungswert 4: "1000"
61 Skalierungswert 5: "1000"
62 Skalierungswert 6: "1000"
63 Skalierungswert 7: "1000"
64 Skalierungswert 8: "1000"
65 Skalierungswert 9: "1000"
66 Skalierungswert 10: "1000"
67 %
68 % Temporal Skalierungswerte
69 Skalierungswert 1: "1000"
70 Skalierungswert 2: "1000"
71 Skalierungswert 3: "1000"
72 Skalierungswert 4: "1000"
73 Skalierungswert 5: "1000"
74 Skalierungswert 6: "1000"
75 Skalierungswert 7: "1000"
76 Skalierungswert 8: "1000"
77 Skalierungswert 9: "1000"
78 Skalierungswert 10: "1000"
```

```
Skalierungswert 11: "1000"
Skalierungswert 12: "1000"
Skalierungswert 13: "1000"
Skalierungswert 14: "1000"
Skalierungswert 15: "1000"
Skalierungswert 16: "1000"
%
Start Selektionsphase 03-Nov-2009 13:06:24
Selektiert train_1_Hense Gitarre: 6 Audio Files.
Selektiert train_2_Marin Gitarre: 6 Audio Files.
Selektiert train_3_Wichmann Gitarre: 6 Audio Files.
Selektiert insgesamt 18 Trainings-Audiodateien.
Selektiert test_1_Hense Gitarre: 6 Audio Files.
Selektiert test_2_Marin Gitarre: 6 Audio Files.
Selektiert test_3_Wichmann Gitarre: 6 Audio Files.
Selektiert insgesamt 18 Test-Audiodateien.
Ende Selektionsphase 03-Nov-2009 13:06:31
% Laenge der Selektionsphase: 7 Sek
%
Start Trainingsphase 03-Nov-2009 13:06:31
%
% SVM Tainingsoptionen
ExecPath = []
Verbosity = 1
Regression = []
C = 1
TubeWidth = []
CostFactor = []
Biased = 1
RemoveIncons = []
ComputeLOO = []
XialphaRho = []
XialphaDepth = []
TransPosFrac = []
Kernel = 0
KernelParam = []
MaximumQP = []
NewVariables = []
CacheSize = []
EpsTermin = []
ShrinkIter = []
ShrinkCheck = []
TransLabelFile = []
AlphaFile = []
Ende Trainingsphase 03-Nov-2009 13:06:32
% Laenge der Trainingsphase: 0 Sek
%
Start Testphase 03-Nov-2009 13:06:32
%
% SVM Testoptionen
ExecPath = []
Verbosity = 1
Regression = []
C = []
TubeWidth = []
CostFactor = []
Biased = []
RemoveIncons = []
```

```
137 ComputeLOO = []
138 XialphaRho = []
139 XialphaDepth = []
140 TransPosFrac = []
141 Kernel = []
142 KernelParam = []
143 MaximumQP = []
144 NewVariables = []
145 CacheSize = []
146 EpsTermin = []
147 ShrinkIter = []
148 ShrinkCheck = []
149 TransLabelFile = []
150 AlphaFile = []
151 Ende Testphase 03-Nov-2009 13:06:34
152 % Laenge der Testphase: 1 Sek
```

E.8. STT-Phase Klassifikationsergebnisse “classifikationResult.txt“

Listing E.8: Klassifikationsergebnisse classifikationResult.txt

```
 1 Gesamt: 18  richtig: 12  66.666667
 2 Hense:       gesamt: 6  richtig: 5   83.333333
 3 Marin:       gesamt: 6  richtig: 3   50.000000
 4 Wichmann:    gesamt: 6  richtig: 4   66.666667
 5 Hense als Marin:        0
 6 Hense als Wichmann:     1
 7 Marin als Hense:        1
 8 Marin als Wichmann:     2
 9 Wichmann als Hense:     0
10 Wichmann als Marin:     2
11 Hense
12 Erster Knoten
13 Hense als Nicht Hense: 1
14 Hense als Nicht Marin: 5
15 Zweiter Knoten
16 Hense als Nicht Hense: 0
17 Hense als Nicht Wichmann: 5
18 Dritter Knoten
19 Hense als Nicht Marin: 1
20 Hense als Nicht Wichmann: 0
21 Marin
22 Erster Knoten
23 Marin als Nicht Hense: 4
24 Marin als Nicht Marin: 2
25 Zweiter Knoten
26 Marin als Nicht Hense: 1
27 Marin als Nicht Wichmann: 1
28 Dritter Knoten
29 Marin als Nicht Marin: 1
30 Marin als Nicht Wichmann: 3
31 Wichmann
32 Erster Knoten
33 Wichmann als Nicht Hense: 5
```

```
34 Wichmann als Nicht Marin: 1
35 Zweiter Knoten
36 Wichmann als Nicht Hense: 1
37 Wichmann als Nicht Wichmann: 0
38 Dritter Knoten
39 Wichmann als Nicht Marin: 3
40 Wichmann als Nicht Wichmann: 2
```

E.9. STT-Phase Trainingsausgaben “1HenseVs2Marin.linear“

Listing E.9: Trainingsausgaben 1HenseVs2Marin.linear

```
1  Scanning examples...done
2  Reading examples into memory...OK. (12 examples read)
3  Optimizing...done. (4 iterations)
4  Optimization finished (0 misclassified, maxdiff=0.00007).
5  Runtime in cpu-seconds: 0.00
6  Number of SV: 9 (including 0 at upper bound)
7  L1 loss: loss=0.00000
8  Norm of weight vector: |w|=0.00012
9  Norm of longest example vector: |x|=741259.12891
10 Estimated VCdim of classifier: VCdim<=7344.85955
11 Computing XiAlpha-estimates...done
12 Runtime for XiAlpha-estimates in cpu-seconds: 0.00
13 XiAlpha-estimate of the error: error<=75.00% (rho=1.00,depth=0)
14 XiAlpha-estimate of the recall: recall=>16.67% (rho=1.00,depth=0)
15 XiAlpha-estimate of the precision: precision=>20.00% (rho=1.00,depth=0)
16 Number of kernel evaluations: 312
17 Writing model file...done
```

E.10. STT-Phase SVM Konfigurationsparameter “svmlopt.m“

Listing E.10: SVM Optionen “svmlopt.m“

```
1  %   The correspondence of the SVM light options to the field in the
2  %   OPTIONS structure is as follows:
3  %   Field          SVM light  Range, description
4  %   'Verbosity'      -v       {0 .. 3}, default value 1
5  %                             Verbosity level
6  %   'Regression'     -z       {0, 1}, default value 0
7  %                             Switch between regression [1] and
8  %                             classification [0]
9  %   'C'              -c       (0, Inf), default value (avg. x*x)^-1
10 %                             Trade-off between error and margin
11 %   'TubeWidth'      -w       (0, Inf), default value 0.1
12 %                             Epsilon width of tube for regression
13 %   'CostFactor'     -j       (0, Inf), default value 1
14 %                             Cost-Factor by which training errors on
15 %                             positive examples outweight errors on
```

```
16  %                               negative examples
17  %   'Biased'          -b        {0, 1}, default value 1
18  %                               Use biased hyperplane x*w+b0 [1] instead of
19  %                               unbiased x*w0 [0]
20  %   'RemoveIncons'    -i        {0, 1}, default value 0
21  %                               Remove inconsistent training examples and
22  %                               retrain
23  %   'ComputeLOO'      -x        {0, 1}, default value 0
24  %                               Compute leave-one-out estimates [1]
25  %   'XialphaRho'      -o        )0, 2), default value 1.0
26  %                               Value of rho for XiAlpha-estimator and for
27  %                               pruning leave-one-out computation
28  %   'XialphaDepth'    -k        {0..100}, default value 0
29  %                               Search depth for extended XiAlpha-estimator
30  %   'TransPosFrac'    -p        (0..1), default value ratio of
31  %                               positive and negative examples in the
32  %                               training data. Fraction of unlabeled
33  %                               examples to be classified into the positive
34  %                               class
35  %   'Kernel'          -t        {0..4}, default value 1
36  %                               Type of kernel function:
37  %                               0: linear
38  %                               1: polynomial (s a*b+c)^d
39  %                               2: radial basis function exp(-gamma ||a-b||^2)
40  %                               3: sigmoid tanh(s a*b + c)
41  %                               4: user defined kernel from kernel.h
42  %   'KernelParam'     -d, -g, -s, -r, -u
43  %                               Depending on the kernel, this vector
44  %                               contains [d] for polynomial kernel, [gamma]
45  %                               for RBF, [s, c] for tanh kernel, string for
46  %                               user-defined kernel
47  %   'MaximumQP'       -q        {2..}, default value 10
48  %                               Maximum size of QP-subproblems
49  %   'NewVariables'    -n        {2..}, default value is the value chosen
50  %                               for 'MaximumQP'. Number of new variables
51  %                               entering the working set in each
52  %                               iteration. Use smaller values to prevent
53  %                               zig-zagging
54  %   'CacheSize'       -m        (5..Inf), default value 40.
55  %                               Size of cache for kernel evaluations in MB
56  %   'EpsTermin'       -e        (0..Inf), default value 0.001
57  %                               Allow that error for termination criterion
58  %                               [y [w*x+b] - 1] < eps
59  %   'ShrinkIter'      -h        {5..Inf}, default value 100.
60  %                               Number of iterations a variable needs to be
61  %                               optimal before considered for shrinking
62  %   'ShrinkCheck'     -f        {0, 1}, default value 1
63  %                               Do final optimality check for variables
64  %                               removed by shrinking. Although this test is
65  %                               usually positive, there is no guarantee
66  %                               that the optimum was found if the test is
67  %                               omitted.
68  %   'TransLabelFile' -l         String. File to write predicted labels of
69  %                               unlabeled examples into after transductive
70  %                               learning.
71  %   'AlphaFile'       -a        String. Write all alphas to this file after
72  %                               learning (in the same order as in the
73  %                               training set).
```

```
74  %
75  %   See also SVML,SVM_LEARN,SVM_CLASSIFY
76  %
77
78  %
79  % Copyright (c) by Anton Schwaighofer (2001)
80  % $Revision: 1.7 $ $Date: 2002/05/30 10:33:28 $
81  % mailto:anton.schwaighofer@gmx.net
82  %
83  % This program is released unter the GNU General Public License.
84  %
```

F. Softwarelistings CE- und STT-Phase

F.1. CE- und STT-Phase Hauptstartskript “start_cestt_demo_all.m“

Listing F.1: Hauptstartskript start_cestt_demo_all.m

```
1  #!/usr/bin/octave --silent
2
3  % Octave load-path erweiterung auf "Software" Verzeichnis und
4  % alle Software Unterverzeichnisse
5  addpath (genpath('~/Gitarrenerkennung/Software/'));
6
7  start_ce_demo;
8  clear all;
9  start_stt_demo1;
10 clear all;
11 start_stt_demo2;
12 clear all;
13 start_stt_demo3;
14 clear all;
```

Literaturverzeichnis

[DFM08] Kerstin Dosenbach, Wolfgang Fohl, and Andreas Meisel. Identification of individual guitar sounds by support vector machines. *Proc. of the 11th Int. Conference on Digital Audio Effects (DAFx-08), Espoo, Finland, September 1-4, 2008*, 2008.

[Dos07] Kerstin Dosenbach. Klassifikation von audiosignalen mit support vector machines. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2007.

[DSC06] Da Deng, Christian Simmermacher, and Stephen Cranefield. Finding the right features for instrument classification of classical music 2006. aidm'06, seiten 34–41. *International Workshop on Integrating AI and Data Mining (AIDM'06)*, 2006.

[FDCC06] Fragoulis, Dimitrios, Constantin, and Constantin. Automated classification of piano-guitar notes. *IEEE Transactions on Speech and Audio Processing,Band 14(3):Seiten 1040– 1050*, 2006.

[Joa99] Thorsten Joachims. Making large-scale svm learning practical. *B. Schölkopf, C. Burges und A. Smola (Herausgeber), Advances in Kernel Methods-Support Vector Learning*, 1999.

[Mey85] Jürgen Meyer. *Akustik der Gitarre in Einzeldarstellungen*, volume Das Musikinstrument, Band 42. Verlag Erwin Bochinsky, 1985.

[MM99] Janet Marques and Pedro J. Moreno. A study of musical instrument classification using gaussian mixture models and support vector machines. Technical report, Cambridge Research Laboratory, 1999. Zugriffsdatum: October 2009.

[SC08] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 2008.

[Sla98] Malcolm Slaney. Auditory toolbox. a matlab toolbox for auditory modeling work. version 2, 1998.

[SS02] Bernhard Schölkopf and Alex Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[ST04] D. Van Steelant and K. Tanghe. Classification of percussive sounds using support vector machines. *Proceedings of the annual machine learning conference of Belgium and The Netherlands, Brussels, Belgium, January 8-9, 2004.*, 2004.

[TS01] C. Traube and J. O. Smith. Extracting the fingering and the plucking points on a guitar string from a recording. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA'01)*, 2001.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*, volume Adaptive and learning Systems for signal processing, communications and control. Wiley-Interscience, 1998.

[Vap00] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*, volume Statistics for Engineering and Information Science. Springer-Verlag, 2000.

[Voi07] Julia Voigt. Klanguntersuchungen an konzertgitarren: Korrelation zwischen physikalischen signalmerkmalen und psychoakustischen attributen. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2007.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. April 2010

Ort, Datum

Unterschrift