# Vokalerkennung durch Extraktion spektraler Eigenschaften und Klassifizierung durch Machine Learning

Christoph Ennser


MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2017

# Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 26. Juni 2017

Christoph Ennser

# Inhaltsverzeichnis

**Erklärung** iii

**Kurzfassung** vi

**Abstract** vii

**1 Einleitung** 1
- 1.1 Motivation . . . 1
- 1.2 Zielsetzung . . . 1
- 1.3 Struktur . . . 2

**2 State of the Art** 3
- 2.1 Aufbau eines Spracherkennungssystems . . . 3
- 2.2 Extrahierung der Eigenschaften durch Signalverarbeitung . . . 4
  - 2.2.1 Linear Predictive Coding . . . 4
  - 2.2.2 Mel Frequency Cepstrum . . . 4
  - 2.2.3 Perceptual Linear Prediction . . . 5
- 2.3 Übersicht von Spracherkennungssystemen . . . 5
  - 2.3.1 Frei verfügbare Systeme . . . 6
  - 2.3.2 Kommerzielle Systeme . . . 7

**3 Vokalerkennung** 9
- 3.1 Darstellung von Audio Signalen . . . 9
  - 3.1.1 Waveform . . . 9
  - 3.1.2 Fourier-Transformation . . . 9
  - 3.1.3 Spektrum . . . 9
  - 3.1.4 Spektrogramm . . . 11
- 3.2 Menschliche Sprachproduktion . . . 11
- 3.3 Eigenschaften von Lauten . . . 12
  - 3.3.1 Grundfrequenz . . . 12
  - 3.3.2 Stimmhafte und stimmlose Laute . . . 13
  - 3.3.3 Vokale und Konsonanten . . . 13
  - 3.3.4 Eigenschaften von Vokalen . . . 14
- 3.4 Machine Learning . . . 15
  - 3.4.1 Arten von Machine Learning . . . 16
  - 3.4.2 Arten von Supervised Learning . . . 16

3.4.3 Generalisierung, Overfitting und Underfitting . . . . . 17
3.4.4 Decision Boundary . . . . . 18
3.4.5 Feature Selection . . . . . 18
3.4.6 Feature Scaling . . . . . 19
3.4.7 Supervised Machine Learning Algorithmen . . . . . 19
3.4.8 Modell-Evaluierung . . . . . 26

**4 Umsetzung** **30**
4.1 Technologie . . . . . 30
4.1.1 Praat . . . . . 30
4.1.2 Python . . . . . 30
4.2 Digitalisierung . . . . . 32
4.2.1 Testdaten . . . . . 32
4.3 Eigenschaften der Vokale . . . . . 32
4.3.1 Kennzeichnung von Vokalen . . . . . 32
4.3.2 Extraktion von Merkmalen . . . . . 33
4.4 Aufbereitung der Daten . . . . . 34
4.4.1 Berechnung der Formanteneigenschaften . . . . . 35
4.4.2 Differenzen der Formantenfrequenzen . . . . . 39
4.4.3 Resultierendes Datenkonstrukt . . . . . 40
4.5 Wahl des Machine Learning Algorithmus . . . . . 40
4.5.1 Feature Scaling . . . . . 42
4.5.2 Feature Selection . . . . . 43
4.5.3 Modell-Evaluierung . . . . . 44
4.5.4 Finale Auswahl des Algorithmus . . . . . 48

**5 Resultat** **49**

**6 Schlussbemerkung** **52**
6.1 Zusammenfassung . . . . . 52
6.2 Ausblick . . . . . 52

**A Installationsanleitung der Umsetzung** **54**
A.1 Praat . . . . . 54
A.2 Python . . . . . 54

**B Inhalt der CD-ROM** **55**
B.1 PDF-Dateien . . . . . 55
B.2 Projektdaten . . . . . 55
B.3 Abbildungen . . . . . 55

**Quellenverzeichnis** **56**
Literatur . . . . . 56
Online-Quellen . . . . . 58

# Kurzfassung

Diese Arbeit zeigt die Erstellung eines Vokalerkennungssystems, das gesprochene deutsche Vokale anhand deren spektraler Eigenschaften erkennen soll. Während der Signalverarbeitung werden mehrere Eigenschaften aus den gesprochenen Vokalen extrahiert und errechnet. Besonderes Augenmerk wird dabei auf die dominanten Frequenzen (Formanten) und deren zeitlichen Verlauf gelegt. Zur Erstellung des Vokalerkennungsmodells wird ein Machine Learning Ansatz verfolgt. Die Vorverarbeitung der Daten sowie die Evaluierung der Machine Learning Modelle werden detailliert behandelt. Die bekanntesten Machine Learning Algorithmen werden mit den zuvor extrahierten Daten durch intensive Tests evaluiert. Der Algorithmus, der diese Tests mit der besten Treffsicherheit und gleichzeitig dem geringsten Vorverarbeitungsaufwand abschließt, wird schlussendlich verwendet werden, um das Vokalerkennungsmodell zu erstellen. Das finale Vokalerkennungssystem weist eine Erkennungsgenauigkeit von 92% auf.

# Abstract

This thesis shows the creation of a vowel recognition system that identifies spoken, German vowels based on their spectral features. During signal processing, several features are extracted and calculated. The focus is on the dominant frequencies (formants) and their variation over time. A machine Learning approach is used to create the vowel recognition model. The preprocessing of the data as well as the evaluations of the machine learning models are covered in detail. Well known machine learning algorithms are evaluated by intense testing, given the previously extracted features. The algorithm that reached the best score with the lowest preprocessing costs will be used to build the vowel recognition model. The final vowel recognition system comes up with an accuracy of 92%.

# Kapitel 1

# Einleitung

## 1.1 Motivation

Der Einsatz von Spracherkennung ist heutzutage weit verbreitet und nimmt stetig zu. Genauso individuell wie die Einsatzgebiete sind auch die Geräte, auf denen sie eingesetzt wird. Um ein Sprachsignal analysieren zu können, werden in der Spracherkennung meist komplexe Algorithmen verwendet, die das Signal mehrfach vorverarbeiten, transformieren und analysieren, um aussagekräftige, die Laute repräsentierende, Eigenschaften zu extrahieren. Laute können anhand ihrer Spektraleigenschaften beschrieben werden. Stimmhafte und stimmlose Laute sind dabei sehr gut zu unterscheiden, während die Unterscheidung innerhalb der stimmhaften Laute, zu denen auch Vokale zählen, ein schwieriges Unterfangen ist. Deshalb versuchen moderne Spracherkennungssysteme diese zusätzlich anhand von Zusammenhängen mit anderen Buchstaben und erlernten Auftretungshäufigkeiten zu unterscheiden. Eine Erkennung von Vokalen nur anhand deren Spektraleigenschaften wäre daher ein wichtiger Schritt, um den Vorgang der Spracherkennung performanter zu gestalten.

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, ein Spracherkennungssystem zu erstellen, das gesprochene deutsche Vokale anhand ihrer spektralen Eigenschaften und dank eines modernen Machine Learning Ansatzes richtig klassifizieren kann. Die gesprochenen Vokale werden digitalisiert, um aus dem umgewandelten Signal des Vokals Eigenschaften zu extrahieren, die diesen repräsentieren. Das Hauptaugenmerk wird auf die dominanten Frequenzen (Formanten), deren Verhältnis zueinander und deren zeitlichen Verlauf gelegt. Mit den dadurch erhaltenen Eigenschaften wird ein Machine Learning Modell generiert und trainiert, das die Klassifizierung schlussendlich durchführt. Das resultierende Spracherkennungssystem soll somit in der Lage sein, Vokale nur anhand der extrahierten, spektralen Eigenschaften zu erkennen.

## 1.3 Struktur

Diese Arbeit wurde zur besseren Strukturierung in mehrere Kapitel unterteilt. In Kapitel 2 wird eine Übersicht über den Aufbau und die Funktionsweise von modernen Spracherkennungssystemen sowie über die Möglichkeiten der Extrahierung von Merkmalen gegeben. Außerdem werden die bekanntesten Hersteller und Produkte von Spracherkennungssoftware und Spracherkennungssystemen aufgezeigt. Nachfolgend werden in Kapitel 3 die notwendigen theoretischen Kenntnisse vermittelt, um ein Vokalerkennungssystem zu erstellen. Dabei werden die Möglichkeiten der Darstellung von Audiosignalen und die daraus resultierenden Informationen vorgestellt sowie die menschliche Sprachproduktion und die Charakteristiken der Laute beschrieben. Auch liefert dieses Kapitel einen detaillierten Überblick über die Funktionsweise von Machine Learning und dessen bekannteste Algorithmen. Kapitel 4 schildert die im Zuge des Experiments durchgeführten Arbeitsschritte. Auf die Digitalisierung des Sprachsignals folgt die Extraktion und Berechnung der Spektraleigenschaften. Des Weiteren wird in diesem Kapitel versucht anhand der erhaltenen Eigenschaften jenes Machine Learning Modell zu eruieren, welches die beste Treffsicherheit aufweist. In Kapitel 5 wird das finale Vokalerkennungsmodell beschrieben und analysiert. Vor allem das Finden von Problembereichen steht dabei im Vordergrund. Abschließend wird in Kapitel 6 noch ein Resümee über das Projekt gezogen und ein Ausblick auf mögliche weitere Forschungen und Experimente im Bereich der Vokalerkennung anhand spektraler Eigenschaften gewagt.

# Kapitel 2

# State of the Art

Spracherkennungen und Sprachsteuerungen sind mittlerweile allgegenwärtig und werden in den verschiedensten Arten und Formen eingesetzt. Durch ihren Einsatz wird die Interaktion mit einem persönlichen virtuellen Assistenten ermöglicht, der unter anderem Kalendereinträge verfasst, Fragen beantwortet oder Einkäufe tätigt. Moderne Häuser kommunizieren mit ihren Bewohnern und Multimediasysteme von Autos führen auf Sprachbefehl Operationen aus oder versenden beispielsweise eine diktierte Nachricht via SMS oder Email. Dies sind nur einige Beispiele, die die weite Verbreitung und den aktiven Einsatz von Spracherkennung und Sprachsteuerung aufzeigen. Auch für Menschen mit körperlicher Beeinträchtigung ergeben sich mit Hilfe von Sprachsteuerungssystemen neue Kommunikations- und Interaktionsmöglicheiten, die ihnen den Alltag erleichtern.

## 2.1 Aufbau eines Spracherkennungssystems

Laut Huang [9] besteht ein typisches Spracherkennungssystem aus mehreren Komponenten, die jeweils einen eigenen Aufgabenbereich erfüllen und im Zusammenspiel die Spracherkennung ermöglichen. Am Beginn steht die Signalverarbeitung. Hier werden die Eigenschaften des Sprachsignals extrahiert anhand dieser das System Rückschlüsse auf die gesprochenen Buchstaben bzw. Wörter ziehen kann. Für diese Rückschlüsse ist dann der Dekoder zuständig. Dabei vergleicht er die extrahierten Eigenschaften mit Einträgen in einem Modell und liefert das Ergebnis mit der höchsten Übereinstimmungswahrscheinlichkeit zurück. Dieses Modell ist in ein Sprach- und ein Akustik-Modell unterteilt. Das Sprachmodell beinhaltet unter anderem Informationen über den Aufbau von Wörtern und welche Wörter in welcher Sequenz des Öfteren miteinander auftreten. Das Akustikmodell beinhaltet unter anderem Informationen über die Akustik, Phonetik, Mikrofon- und Umwelt-Eigenschaften, sowie geschlechterspezifische und durch Dialekte hervorgerufene Unterschiede in der Sprache. Der Dekoder macht das System außerdem lernfähig, indem er Informationen bereitstellt, die durch die Anpassungs-Einheit dann das Sprach- und Akustik-Modell so modifizieren, dass zukünftig bessere Ergebnisse erzielt werden können. In Abbildung 2.1 ist dieser Ablauf dargestellt.

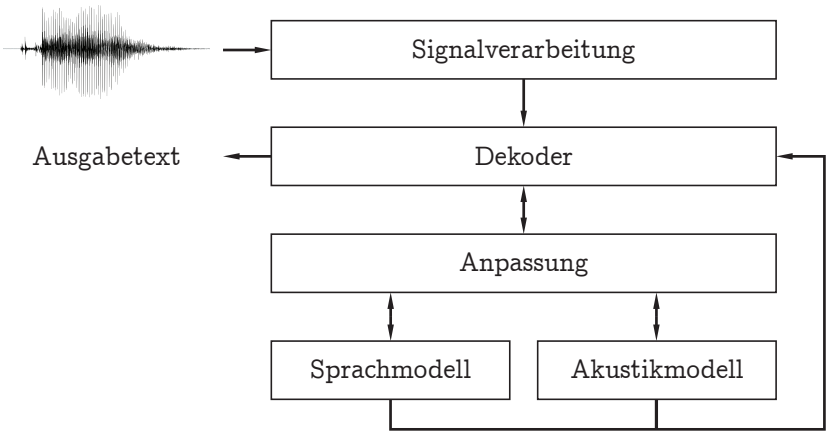



**Abbildung 2.1:** Grundlegende Systemarchitektur eines Spracherkennungssystems.

## 2.2 Extrahierung der Eigenschaften durch Signalverarbeitung

Um ein Sprachsignal möglichst eindeutig beschreiben zu können, werden Methoden verwendet, die aus dem Signal Koeffizienten extrahieren, die dieses möglichst gut repräsentieren. Es gibt eine Vielzahl an Methoden, die in der Praxis zum Einsatz kommen. Nachfolgend sind die drei am weitesten verbreiteten Methoden aufgelistet [20].

### 2.2.1 Linear Predictive Coding

Laut Rabiner [19] ist *Linear Predictive Coding* (LPC) eine robuste, verlässliche und zutreffende Methode, um Parameter zu extrahieren, die die menschliche Spracherzeugung charakterisieren. Basis für LPC ist ein einfaches Quelle-Filter-Modell. Die Quelle ist der Lunge nachempfunden und der menschliche Vokaltrakt wird als veränderbare Röhre dargestellt. Anhand dieses Modells wird nun durch Anpassung der Röhre versucht, das Sprachsignal nachzubilden. Durch die extrahierten Filterkoeffizienten, die unter anderem den Zustand der Röhre beschreiben, kann das Ursprungssignal somit beschrieben werden. In Abbildung 2.2 ist der Ablauf einer LPC-Analyse grafisch dargestellt.

### 2.2.2 Mel Frequency Cepstrum

*Mel Frequency Cepstrum* (MFC) ist eine der meist verbreiteten und vielversprechendsten Methoden, um Eigenschaften zu extrahieren [15]. Anders als bei Linear Predictive Coding wird hier das Signal in den Frequenzbereich transformiert. Wie Logan [14] erklärt, sind die tiefen Frequenzen für die Sprache wichtiger als die hohen. Deshalb wird bei MFC ein sogenanntes *Mel Filter* angewandt, das die Frequenzbereiche entsprechend priorisiert. Durch anschließende Logarithmierung und diskrete Kosinus-Transformation werden die Koeffizienten extrahiert. In Abbildung 2.3 wird dieser Ablauf grafisch dargestellt.

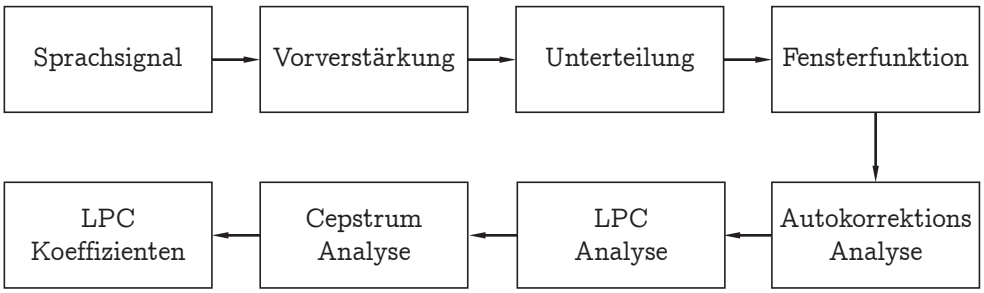



**Abbildung 2.2:** Ablaufdiagramm der *Linear Predictive Coding* Methode nach Kepuska [10].

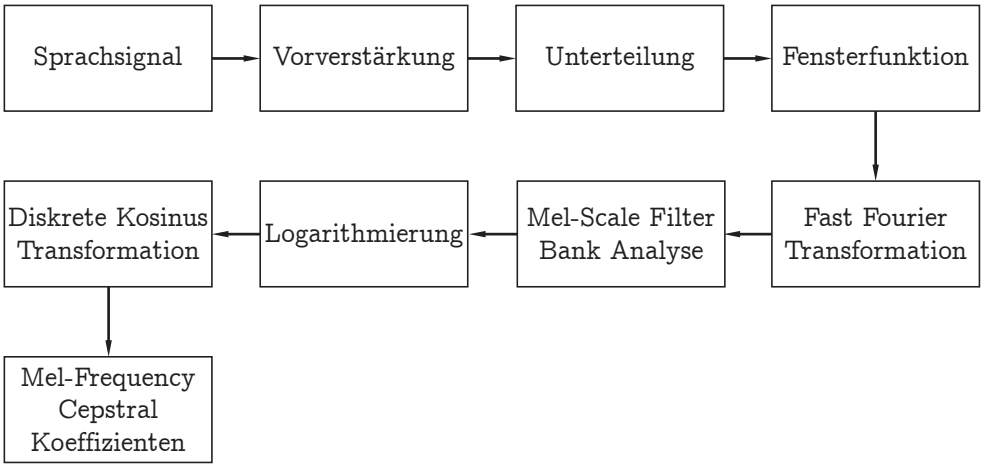



**Abbildung 2.3:** Ablaufdiagramm der *Mel Frequency Cepstrum* Methode nach Kepuska [10].

### 2.2.3 Perceptual Linear Prediction

*Perceptual Linear Prediction* (PLP) wurde 1990 von Hermansky entwickelt [8]. Bei PLP wird versucht, das Spektrum dem menschlichen Gehör anzupassen. Dieses reagiert auf gewisse Frequenzbereiche sensibler als auf andere. Wie Kumar [12] erklärt, wird deswegen bei PLP eine Gewichtung des Spektrums anhand psychophysischer Transformationen vorgenommen. Der Ablauf der PLP Methode wird in Abbildung 2.4 dargestellt.

## 2.3 Übersicht von Spracherkennungssystemen

Es existiert eine Vielzahl an Anbietern für Spracherkennungssysteme. Die bekanntesten frei verfügbaren und kommerziellen Systeme werden hier kurz vorgestellt.

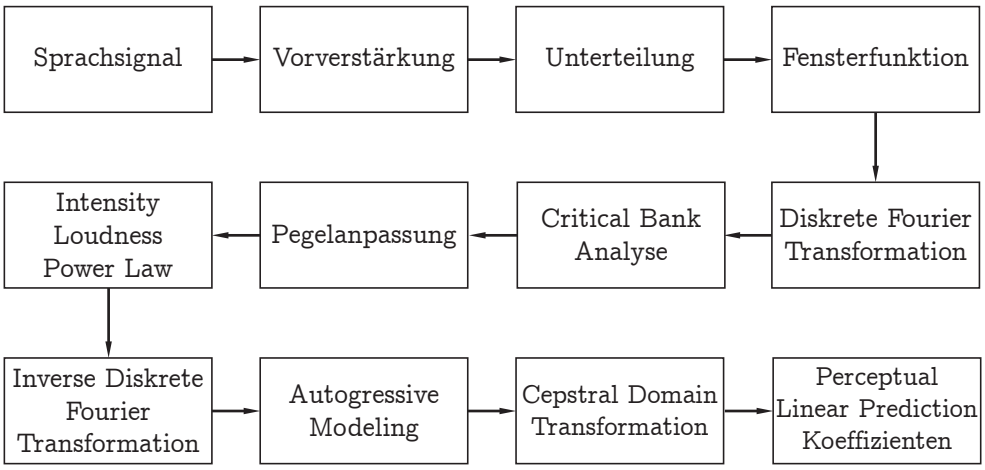



**Abbildung 2.4:** Ablaufdiagramm der *Perceptual Linear Prediction* Methode nach Kepuska [10].

### 2.3.1 Frei verfügbare Systeme

Es gibt drei frei verfügbare Systeme, die sich aufgrund ihrer Erfahrung, Leistung und der großen Entwicklergemeinschaft von den anderen abheben.

#### CMU Sphinx

Das seit 1986 an der Carnegie Mellon University entwickelte Open Source System CMU Sphinx[1] wurde in der Programmiersprache Java entwickelt. Die aktuelle Version des Spracherkennungssystems Sphinx-4, wurde in Zusammenarbeit von Carnegie Mellon University, Sun Microsystems Laboratories und Mitsubishi Electric Research Laboratories (MERL) entwickelt [13]. Sphinx setzt bei der Merkmalsgewinnung auf die (in Abschnitt 2.2.2 beschriebenen) MFC-Koeffizienten.

#### Kaldi

Ein weiteres Open Source System ist das seit 2009 in der Programmiersprache C++ entwickelte und rasant wachsende Kaldi[2]. Obwohl es noch ein eher junges System ist, besticht es durch seine Leistungsfähigkeit und die vielen Möglichkeiten, das System zu trainieren und zu erweitern. Kaldi unterstützt bei der Merkmalsgewinnung die Extraktionsmethoden MFC und PLP.

[1] http://cmusphinx.sourceforge.net/
[2] http://kaldi-asr.org/

#### Hidden Markov Model Toolkit

Das Hidden Markov Model Toolkit[3] (HTK) wurde erstmals 1989 von Steve Young an der Cambridge Universität veröffentlicht. Das zwischenzeitlich von *Microsoft* gekaufte System steht zur nichtkommerziellen Verwendung und Erweiterung zur Verfügung. Jede in Abschnitt 2.2 beschriebene Extraktionsmethode kann bei HTK gewählt werden.

### 2.3.2 Kommerzielle Systeme

Bei den kommerziellen Systemen gibt es mehrere Anbieter, die die Entwicklung ihrer Spracherkennungssysteme in den letzten Jahren stark vorangetrieben haben. Aufgrund ihrer weiten Verbreitung und intensiven Nutzung heben sich einige Dienste vom Rest ab.

#### Siri

*Siri*[4] wurde erstmals 2011 von Apple veröffentlicht und ist mittlerweile in fast all ihren Produkten inkludiert. Für Entwicklerinnen und Entwickler steht das *SiriKit* zur Verfügung, das eine Nutzung von Siri in der eigenen Applikation ermöglicht. Ende 2016 verzeichneten die Apple Server zwei Milliarden Siri Anfragen pro Woche [25], was die intensive Nutzung des Service bestätigt.

#### Google Assistant

Google hat mit *Google Assistant*[5] einen Spracherkennungsdienst, der 2016 veröffentlicht wurde. Dieser kommt unter anderem in dem, zur Zeit erst in Amerika und England erhältlichen, smarten Lautsprecher *Google Home*[6] und bei den *Pixel*[7] Mobiltelefonen zum Einsatz. Bei der unter *Android* und *iOS* verfügbaren *Google App* wird jedoch noch der 2012 erschienene Dienst *Google Now*[8] eingesetzt. Von Entwicklerinnen und Entwicklern kann außerdem die *Google Cloud Speech API*[9] genutzt werden, die für die ersten 60 Minuten kostenlos zur Verfügung steht.

#### Alexa

Der Spracherkennungsdienst von Amazon heißt *Alexa* und kommt unter anderem bei Fernsehern, in Autos oder auf mobilen Geräten zum Einsatz. Eine große Verbreitung erlangte *Alexa* durch die *Echo* Produktfamilie, Amazons persönlichen Assistenten in Form eines Lautsprechers. Um *Alexas* Funktionalitäten in das eigene Produkt zu integrieren, kann das Cloud basierte Sprachservice *Alexa Voice Service*[10] kostenlos verwendet werden.

[3] http://htk.eng.cam.ac.uk/
[4] https://www.apple.com/ios/siri/
[5] https://assistant.google.com/
[6] https://madeby.google.com/home/
[7] https://madeby.google.com/phone/
[8] https://www.google.com/intl/de/landing/now/
[9] https://cloud.google.com/speech/
[10] https://developer.amazon.com/de/alexa-voice-service

### Cortana

Der 2014 erschienene Dienst *Cortana*[11] wurde von Microsoft entwickelt und ist bei allen *Windows 10* Geräten sowie der Spielekonsole *Xbox One* inkludiert. Nach *Google Home* und *Amazon Echo* wurde auch ein smarter Lautsprecher mit *Cortanta* für Ende 2017 angekündigt. Microsoft bietet für Entwicklerinnen und Entwickler die *Bing Speech API*[12] an, welche Sprache in Text umwandelt und für bis zu 5000 Transaktionen im Monat gratis zur Verfügung steht.

[11] https://www.microsoft.com/en/mobile/experiences/cortana/
[12] https://azure.microsoft.com/en-us/services/cognitive-services/speech/

# Kapitel 3

# Vokalerkennung

Ziel dieser Arbeit ist es, aussagekräftige Eigenschaften aus einem Sprachsignal zu extrahieren und mit Hilfe von Machine Learning ein Vokalerkennungsmodell zu generieren, das die gesprochenen Vokale möglichst genau erkennen kann. Die dafür benötigten theoretischen Kenntnisse werden in diesem Kapitel vermittelt.

## 3.1 Darstellung von Audio Signalen

Um spektrale Eigenschaften extrahieren zu können, muss das Sprachsignal zuerst digitalisiert werden. Anschließend kann das digitalisierte Signal auf verschiedene Arten konvertiert und dargestellt werden. Welche Eigenschaften man aus welcher Darstellung herauslesen kann und was sie repräsentiert, wird folgend erläutert.

### 3.1.1 Waveform

Die Waveform ist eine grafische Darstellung eines Sprachsignals im Zeitbereich. Dabei wird die Änderung der Amplitude über die Zeit angezeigt. Diese Darstellung ermöglicht es, unterschiedliche Bereiche in einem Signal zu identifizieren, da sich beispielsweise laute, energievolle Passagen klar von ruhigen Passagen abheben. Auf Abbildung 3.1 wird die Waveform-Darstellung eines gesprochenen „a“ dargestellt.

### 3.1.2 Fourier-Transformation

Um ein Signal vom Zeitbereich in den Frequenzbereich zu transformieren, kommt die Fourier-Transformation zum Einsatz. Dabei werden alle Frequenzen, die in einem Bereich eines kontinuierlichen Signals vorhanden sind, ermittelt. Dies kann bewerkstelligt werden, da jedes Signal in seine Sinus- und Kosinus-Anteile zerlegt werden kann, wodurch man eine konkrete Auflistung aller Einzelfrequenzen erhält. Huang [9] liefert eine detaillierte Erklärung der Funktionsweise der Fourier-Transformation.

### 3.1.3 Spektrum

Das Spektrum ist die grafische Darstellung der durch die Fourier Transformation erhaltenen Frequenzkomponenten eines Sprachsignals im Frequenzbereich. Für jede im

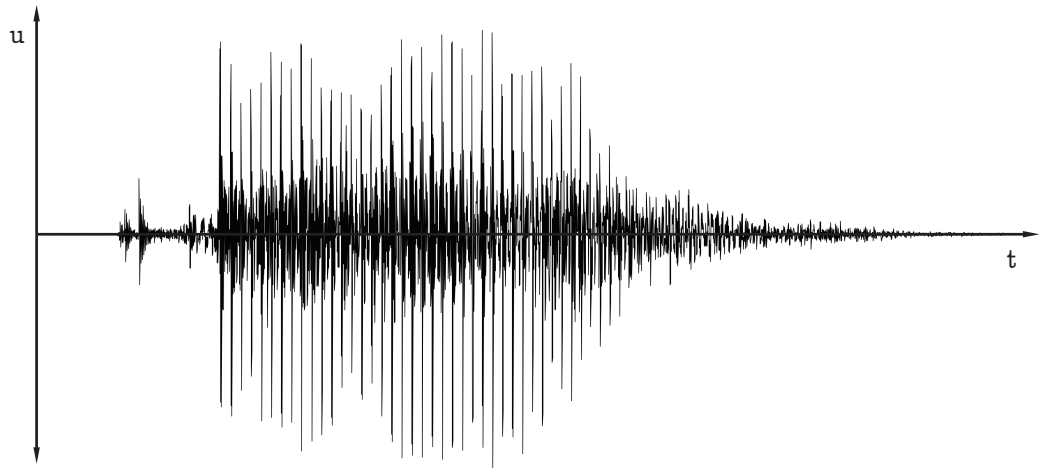



**Abbildung 3.1:** Waveform-Darstellung eines gesprochenen Vokals „a“. Die X-Achse repräsentiert die Zeit, die Y-Achse die Amplitude.

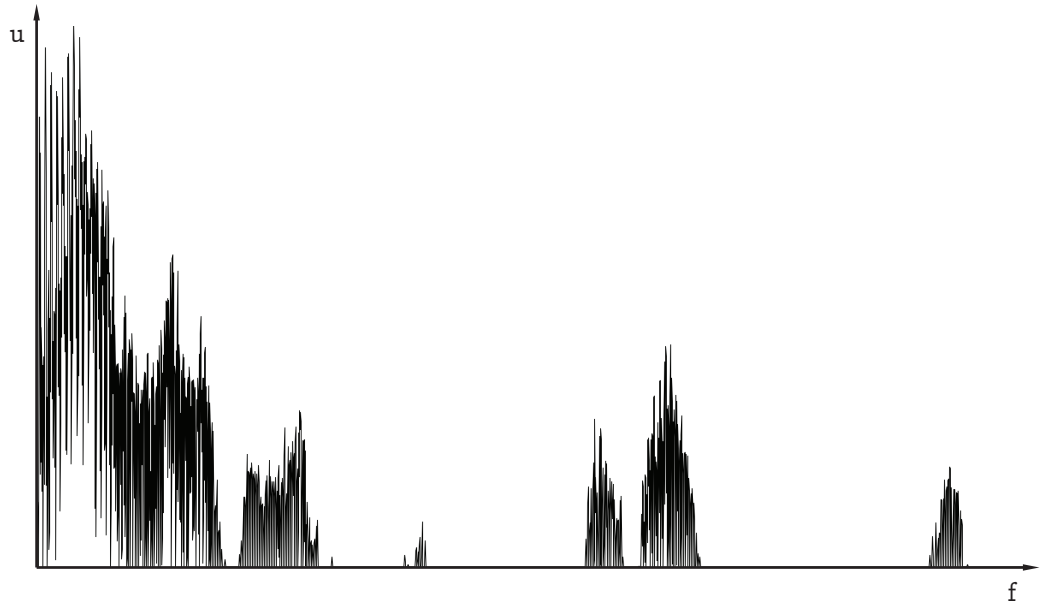



**Abbildung 3.2:** Spektrum eines gesprochenen Vokals „a“. Die X-Achse repräsentiert die Frequenz, die Y-Achse die Amplitude.

Signal vorhandene Frequenz wird die dazugehörige Amplitude dargestellt, wodurch die dominanten Frequenzen im Spektrum sehr gut herauszulesen sind. Diese Frequenzen sind gerade für die Vokalerkennung sehr interessant, da deren Zusammenhang und zeitlicher Verlauf einen Vokal sehr gut charakterisieren. Ein Beispiel für ein Spektrum eines gesprochenen „a“ wird auf Abbildung 3.2 dargestellt.

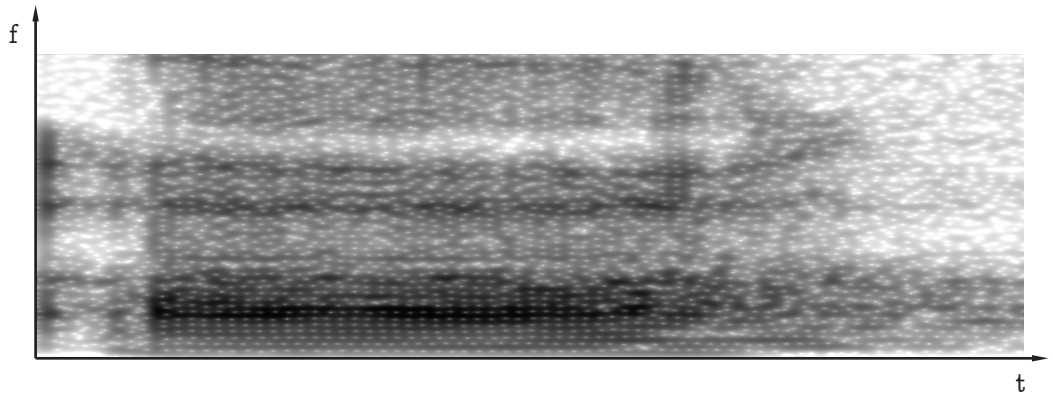



**Abbildung 3.3:** Spektrogramm eines gesprochenen Vokals „a“. Die X-Achse repräsentiert die Zeit, die Y-Achse die Frequenz und der Schwarzwert die Amplitude.

### 3.1.4 Spektrogramm

Das Spektrogramm dient dazu, den zeitlichen Verlauf eines Signals im Frequenzbereich abbilden zu können. Das kontinuierliche Signal wird in einem definierten Abstand in kleine Teilbereiche zerlegt, welche jeweils in den diskreten Frequenzbereich transformiert werden. Jedes Teilspektrum wird als vertikaler Balken dargestellt. Aneinandergereiht ergeben diese Balken das Spektrogramm. Um neben der Zeit und Frequenz auch noch die Amplitude darzustellen, wird diese Ebene im Spektrogramm durch einen Farb- oder Schwarz-Wert repräsentiert. Dadurch wird der zeitliche Verlauf der dominanten Frequenzen sichtbar, was für die Vokalerkennung von großer Bedeutung ist. Auf Abbildung 3.3 wird das Spektrogramm eines gesprochenen „a“ dargestellt.

## 3.2 Menschliche Sprachproduktion

Die menschliche Sprachproduktion wird durch ein Zusammenspiel von mehreren Komponenten ermöglicht. Eine Übersicht der beteiligten Organe ist auf Abbildung 3.4 zu sehen. Fellbaum [6] erklärt den Prozess der Spracherzeugung auf sehr detaillierte Weise.

Um einen Laut produzieren zu können, muss zuerst eingeatmet werden. Dabei wird das Zwerchfell gesenkt und der Brustkorb gehoben, um die Lunge durch Erzeugung von Unterdruck mit Luft zu befüllen. Danach wird die Luft aus der Lunge in die Luftröhre gepresst. Diese Luft muss nun die Glottis passieren, welche die Stellung der Stimmbänder und damit die Größe der Öffnung zwischen den Stimmbändern steuert. Durch die durchgepresste Luft werden die Stimmbänder in Schwingung versetzt und ein Schalldrucksignal erzeugt. Durch dieses Signal wird dann der anschließende Luftraum angeregt, welcher aus Rachen-, Mund- und Nasenraum besteht. Die Stellung der Glottis sowie die Stellung der in Abbildung 3.4 ersichtlichen Komponenten des Rachen-, Mund- und Nasenraums sind maßgeblich für den resultierenden Laut verantwortlich.

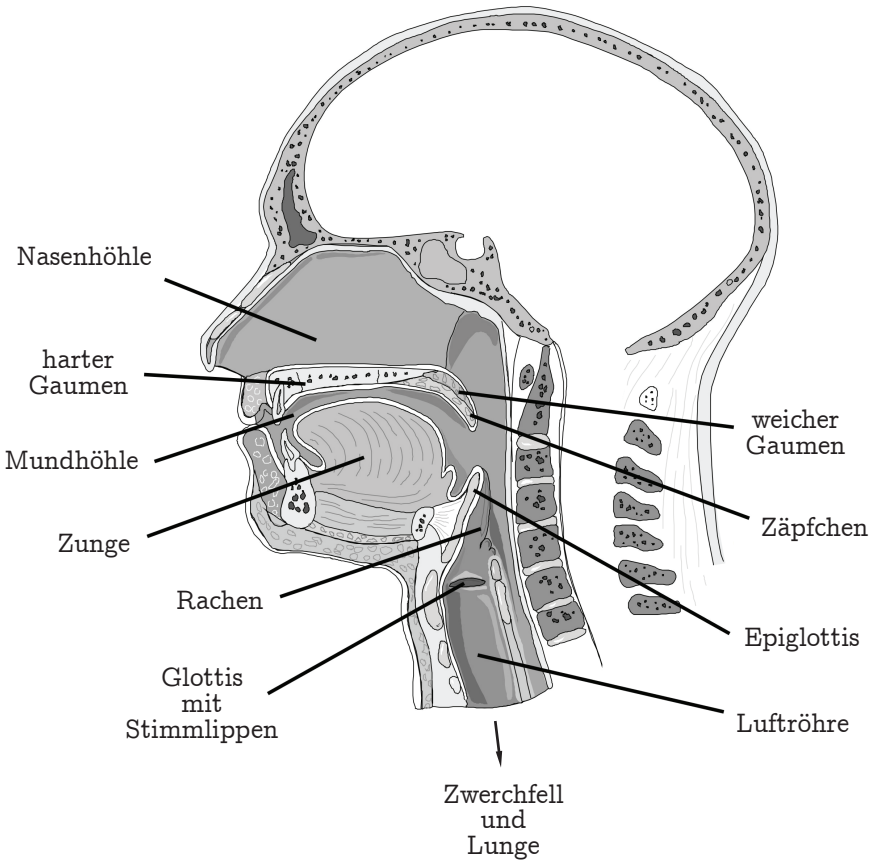



**Abbildung 3.4:** Komponenten der menschlichen Sprachproduktion. Bild nach [6].

**Tabelle 3.1:** Die Grundfrequenzbereiche von Männern, Frauen und Kindern nach Brenner [3].

| | Grundfrequenzbereich |
|---|---|
| Männer | 100 – 150 Hz |
| Frauen | 190 – 250 Hz |
| Kinder | 350 – 500 Hz |

## 3.3 Eigenschaften von Lauten

### 3.3.1 Grundfrequenz

Die Grundfrequenz ist die Basisfrequenz eines Lautes und entspricht der Frequenz des durch die Stimmlippen erzeugten Quellensignals [3]. Sie ist von der Länge und Spannung der Stimmbänder abhängig, weshalb aus anatomischen Gründen Männer, Frauen und Kinder anhand ihrer Grundfrequenzen relativ gut voneinander unterscheidbar sind. In Tabelle 3.1 werden die verschiedenen Grundfrequenzbereiche nach Brenner [3] aufgelistet.

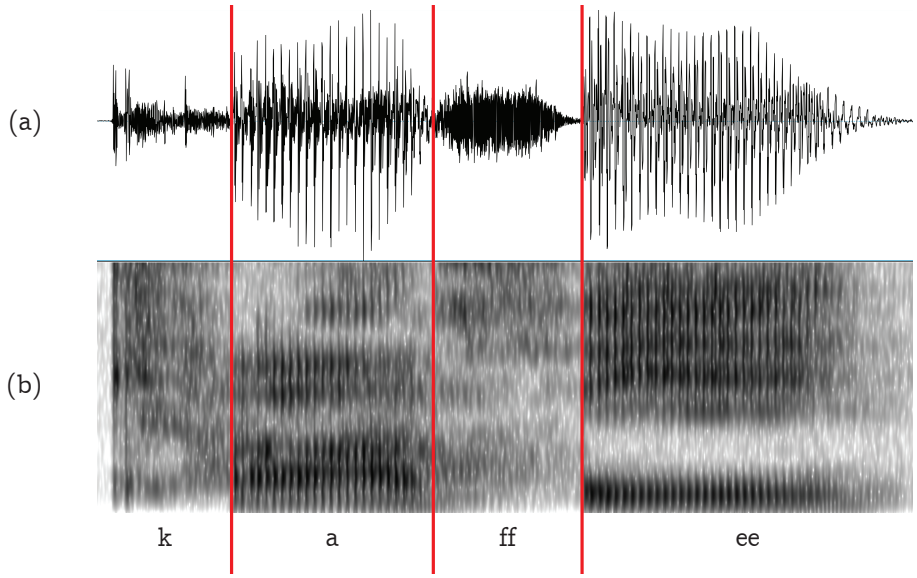



**Abbildung 3.5:** Waveform und Spektrogramm des gesprochenen Wortes „Kaffee“. Die Waveform Darstellung (a) zeigt, dass stimmlose Laute weniger Energie aufweisen als stimmhafte. Im Spektrogramm (b) kann man bei den stimmhaften Lauten deutliche, kontinuierliche Bahnen erkennen, während bei stimmlosen Lauten kaum Muster erkennbar sind.

### 3.3.2 Stimmhafte und stimmlose Laute

Bei der menschlichen Spracherzeugung wird zwischen stimmhaften und stimmlosen Lauten unterschieden. Ist die Glottis weitgehend geschlossen, ist der Laut stimmhaft. Ist sie geöffnet, werden die Stimmbänder nicht angeregt und ein stimmloser Laut entsteht [6]. Stimmhafte Laute weisen in ihrer Zeit- und Frequenzstruktur ein ziemlich kontinuierliches Muster auf, was bei stimmlosen Lauten nicht der Fall ist. Laut Huang [9] haben stimmhafte Laute auch meist mehr Energie als stimmlose. Die Abbildung 3.5 zeigt den Unterschied von stimmhaften und stimmlosen Lauten in dem gesprochenen Wort „Kaffee“, anhand dessen Waveform-Darstellung und des Spektrogramms.

### 3.3.3 Vokale und Konsonanten

Laute werden üblicherweise in Konsonanten und Vokale unterteilt. Während man bei Konsonanten zwischen stimmhaften und stimmlosen unterscheidet, sind alle Vokale stimmhaft.

Bei der Artikulation von Konsonanten passiert der Luftausstoß nicht ungehindert. Das heißt, dass der Luftstrom durch Verengung oder Verschließung der Sprachorgane unterbrochen oder behindert wird. Bei den Vokalen hingegen kann der Luftstrom ungehindert austreten.

**Tabelle 3.2:** Alle Vokale der deutschen Sprache.

| Monophthonge | | Diphthonge |
|---|---|---|
| Hauptvokale | Umlaute | Zwielaute |
| a, e, i, o, u | ä, ö, ü | ai, au, äu, ei, eu, ui |

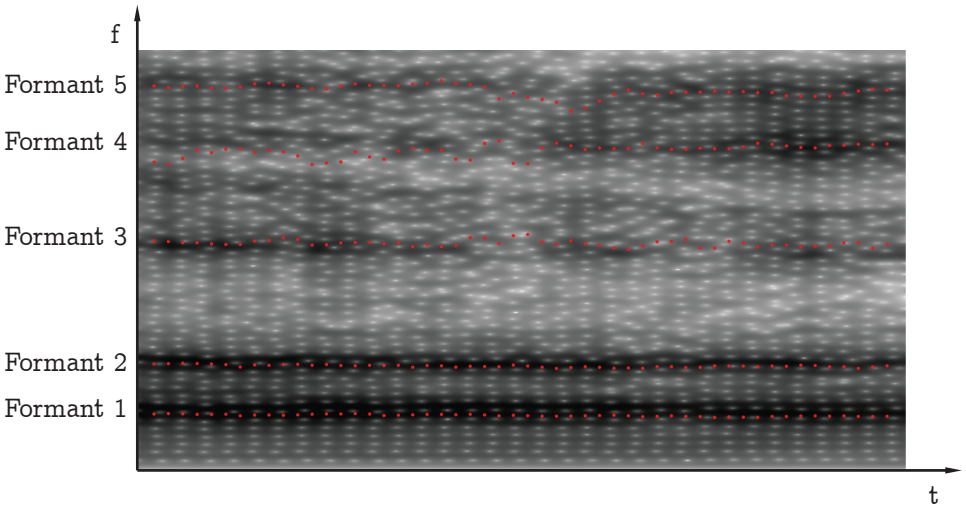



**Abbildung 3.6:** Spektrogramm eines gesprochenen Vokals „a“ mit eingezeichneten Formanten.

### 3.3.4 Eigenschaften von Vokalen

Zu den Vokalen zählen nicht nur die Hauptvokale „a“, „e“, „i“, „o“, „u“, sondern auch die Zwie- und Umlaute. Die Hauptvokale und Umlaute bestehen nur aus einem Laut und werden Monophthonge genannt. Zwielaute bestehen aus zwei aneinandergereihten Vokalen und werden Diphthonge genannt [3]. Alle in der deutschen Sprache enthaltenen Vokale sind in der Tabelle 3.2 aufgelistet.

Allen Vokalen liegt die Grundfrequenz des Sprechers zu Grunde. Durch Anpassung von Zungenposition, Lippenrundung und Zungenhöhe kann der Resonanzraum verändert werden. Diese Veränderungen ermöglichen es, die unterschiedlichen Vokale zu bilden.

#### Formanten

Die Frequenzen, welche die Maxima der Energiedichte im Spektrogramm aufweisen, werden Formanten genannt. Diese werden, beginnend bei der niedrigsten Frequenz, durchnummeriert. Besonders die ersten beiden Formanten sind für die Identifikation von Vokalen sehr wichtig [9]. Die Abbildung 3.6 zeigt, wie gut Formanten von Vokalen im Spektrum erkennbar sind.

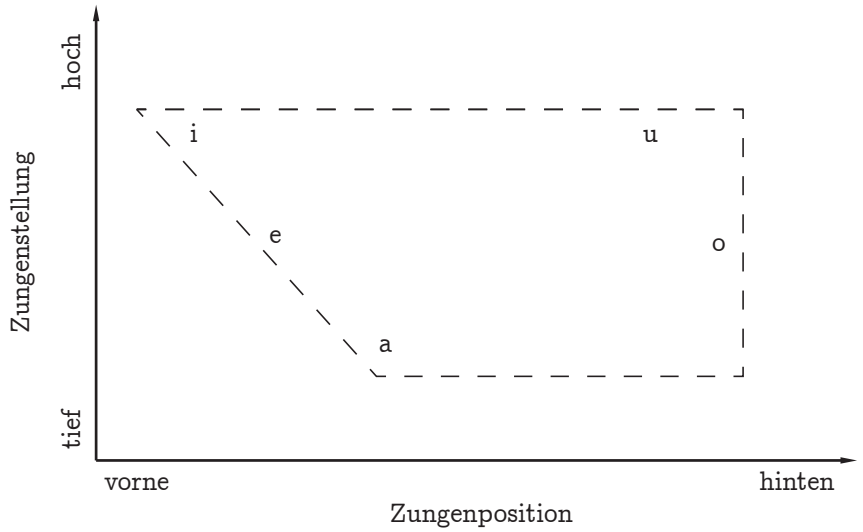



**Abbildung 3.7:** Anordnung der Vokale im Vokalviereck.

### Vokalviereck

Unter Berücksichtigung der anatomischen Merkmale eines gesprochenen Vokals lassen sich diese anhand des Vokalvierecks unterscheiden. Dabei wird auf der X-Achse des Vokalvierecks die Zungenposition, von vorne bis hinten, und auf der Y-Achse die Zungenstellung, von tief bis hoch, aufgetragen [18]. Das daraus resultierende Trapez kann auf Abbildung 3.7 betrachtet werden.

## 3.4 Machine Learning

Machine Learning ist ein Teilbereich der künstlichen Intelligenz, der Wissen aus vorhandenen Daten extrahiert [16]. Obwohl es in den meisten Fällen nicht bemerkt wird, ist beinahe jeder im täglichen Leben damit konfrontiert. Zu den vielseitigen Einsatzbereichen von Machine Learning zählen unter anderem die Erkennung von SPAM E-Mails, das Finden der besten Suchergebnisse, Empfehlungsdienste von Webshops, das Erstellen von Diagnosen in der Medizin oder die optimale Steuerung des Verkehrsflusses. Ziel von Machine Learning ist es, mit Hilfe eines Algorithmus von vorhandenen Daten zu lernen, um dann möglichst genaue Vorhersagen für bekannte oder unbekannte Probleme zu liefern.

Es gibt viele Wege, die zu einem Machine Learning-Modell führen. Die Entscheidungen, welche Art von Machine Learning oder welchen Algorithmus man verwendet, hängen zum größten Teil von den vorhandenen Daten ab und sind essenziell. Die wichtigsten Aspekte, Algorithmen und Vorgänge werden in diesem Abschnitt beschrieben.

### 3.4.1 Arten von Machine Learning

Bei Machine Learning unterscheidet man anhand der vorhandenen Information über die Trainingsdaten zwischen *Supervised* und *Unsupervised* Machine Learning.

#### Supervised Learning

Beim Trainieren eines *Supervised Learning* Modells werden ihm Trainingsdaten mit den dazugehörigen Ergebnissen zugeführt. Das Modell versucht dann einen Weg zu finden aufgrund der erhaltenen Daten auf das erwartete Ergebnis zu schließen. Ein fertiges funktionierendes Modell ist schlussendlich eigenständig im Stande, durch Zufuhr von neuen, nie zuvor gesehenen Daten, die dieselben Strukturen aufweisen wie die Trainingsdaten, das richtige Ergebnis zu errechnen [16]. Als Beispiel für *Supervised Learning* kann die Identifikation der Sprache, in der ein Text verfasst wurde, genannt werden. Übergibt man dem Modell einen neuen Text, retourniert es die dazu passende Sprache.

#### Unsupervised Learning

Bei *Unsupervised Learning* ist im Gegensatz zu *Supervised Learning* das zu den eingegebenen Daten dazugehörige Ergebnis nicht bekannt. Durch Einsatz von *Unsupervised Learning* können Daten strukturiert oder bedeutende Informationen extrahiert werden [21]. Ein Beispiel dafür ist die automatische Gesichtserkennung bei Fotos. Dabei werden alle Fotos nach Gemeinsamkeiten durchsucht und gruppiert.

Zusammenfassend kann gesagt werden: Kennt man im Vorfeld die richtigen Ergebnisse der Trainingsdaten, handelt es sich um *Supervised Learning*, kennt man sie nicht, spricht man von *Unsupervised Learning*. Da für diese Arbeit die richtigen Ergebnisse bekannt sind und somit nur *Supervised Machine Learning* relevant ist, wird im weiteren Verlauf der Arbeit der Fokus auf diese Form von Machine Learning gelegt.

### 3.4.2 Arten von Supervised Learning

Bei *Supervised Learning* Problemen unterscheidet man wiederum zwischen *Klassifikations* und *Regressions* Problemen. Der Unterschied steckt im Ergebnis des Machine Learning Modells.

#### Klassifikation

Bei einem *Klassifikations*-Problem ist das Ergebnis eine Auswahl aus einer Liste bekannter Bezeichnungen [16]. Soll beispielsweise die dominante Farbe eines Bildes im RGB-Farbraum ermittelt werden, so ist das Ergebnis entweder rot, grün oder blau und liefert damit ein eindeutig Ergebnis. Dieser Arbeit liegt somit mit der Klassifizierung der Vokale ein *Klassifikations*-Problem zugrunde.

#### Regression

Bei einem *Regressions*-Problem besteht das Ziel darin, eine kontinuierliche Zahl zu erhalten [16]. Das Modell hat daher keine Liste von möglichen Antworten, sondern gibt

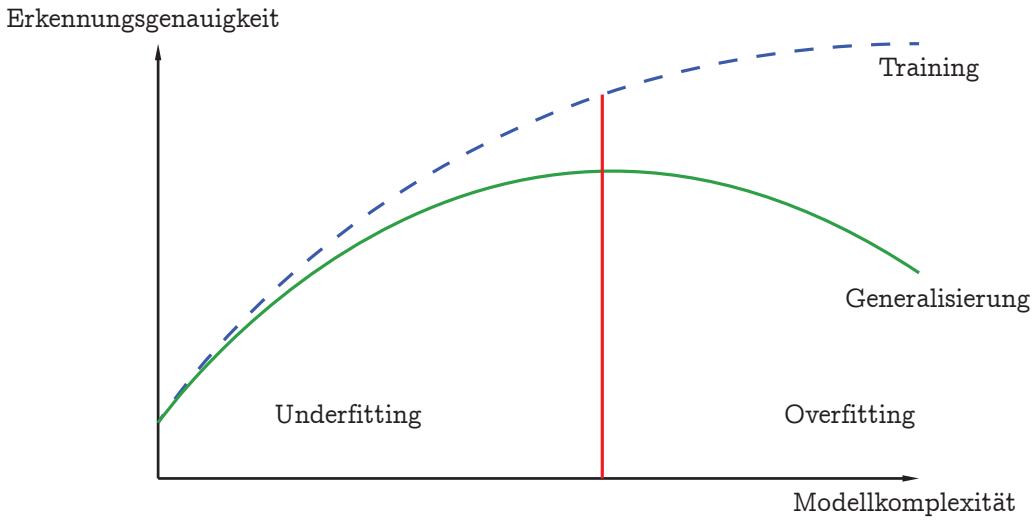



**Abbildung 3.8:** Verlauf der Generalisierung eines Modells bei steigender Anpassung an die Testdaten. Die vertikale rote Linie markiert jenen Bereich, wo das Modell am besten generalisiert.

stattdessen einen kalkulierten Wert zurück. Ein Beispiel eines *Regressions*-Problems wäre die Errechnung der Lebenserwartung eines Menschen anhand dessen Eigenschaften, Hobbies und Lebensstandards, was eine Fließkommazahl als Ergebnis liefern würde.

### 3.4.3 Generalisierung, Overfitting und Underfitting

Wenn noch nie zuvor gesehene Daten einer Klasse eindeutig zugewiesen werden können, sagt man, dass ein Modell in der Lage ist, zu generalisieren [16]. Dies ist das Ziel eines jeden trainierten Machine Learning Modells.

Ein generalisierendes Modell zu erstellen, bedeutet jedoch meist eine große Herausforderung und ist oft nicht möglich. Deshalb wird versucht, das Modell so gut wie möglich zu trainieren, bzw. den Generalisierungsfehler so klein wie möglich zu halten, um einem perfekten Modell so nahe wie möglich zu kommen. Sehr viele Machine Learning Algorithmen arbeiten mit Parametern, die die Komplexität des Modells regulieren. Als Beispiel kann der (in Abschnitt 3.4.7 vorgestellte) *K-Nearest-Neighbors*-Algorithmus, wo die Anzahl an zu betrachtenden Nachbarseigenschaften gewählt wird, genannt werden. Die Erhöhung der Komplexität bringt jedoch auch eine Gefahr mit sich, die darin liegt, dass das Modell den Testdaten immer mehr auf den Leib geschneidert wird und somit, ab einem gewissen Grad der Komplexität, die Generalisierung wieder abnimmt [16]. Wählt man die Modellkomplexität zu gering, spricht man von *Underfitting*. Wird die Modellkomplexität zu hoch gewählt, resultiert das in *Overfitting*. Ziel ist es also, jenen Punkt zu finden, wo das Modell von *Underfitting* auf *Overfitting* übergeht und somit am besten generalisiert. Abbildung 3.8 stellt den Verlauf der Generalisierung bei steigender Modellkomplexität grafisch dar.

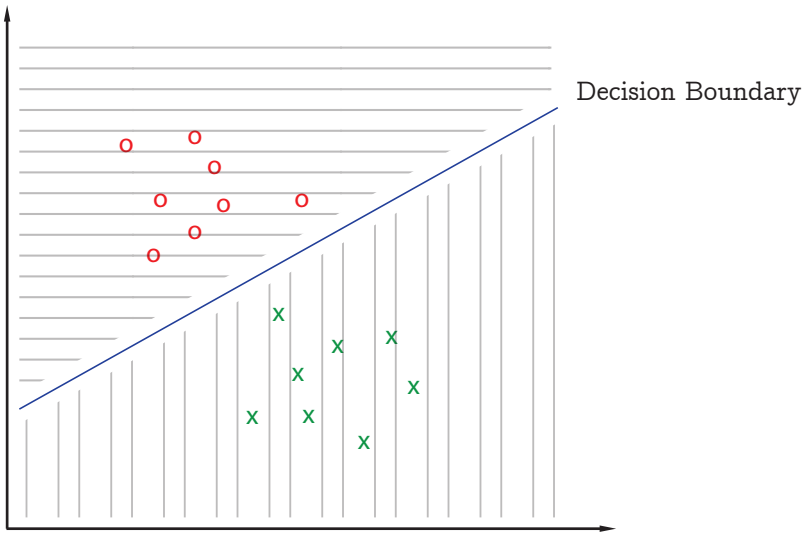



**Abbildung 3.9:** Darstellung der beiden Bereiche, die eine *Decision Boundary* im Falle eines binären, linearen Modells erzeugt.

### 3.4.4 Decision Boundary

Als *Decision Boundary* wird eine imaginäre Grenze bezeichnet, die von einem Algorithmus erzeugt wird, um neue Werte einer Klasse zuordnen zu können, bzw. die vorhandenen Klassen klar voneinander unterscheidbar zu machen. Wird ein neuer Wert in den Raum der Eigenschaften platziert, wird diesem die Klasse jenes Bereichs zugewiesen, den die *Decision Boundary* an diesem Punkt aufspannt. Eine *Decision Boundary* kann, je nach eingesetztem Algorithmus, verschiedene Formen annehmen. Abbildung 3.9 zeigt eine einfache *Decision Boundary* eines linearen Modells, wie es beispielsweise bei *Logistic Regression* (in Abschnitt 3.4.7) beschrieben wird.

### 3.4.5 Feature Selection

Um ein Modell möglichst optimal trainieren zu können, kommt oft eine gezielte Aufbereitung der vorhandenen Daten zum Einsatz. Die *Feature Selection* ist ein Bereich dieser Datenvorverarbeitung, der das Ziel verfolgt, die vorhandenen Daten auf das Wesentliche zu reduzieren bzw. die wertvollsten Eigenschaften herauszufiltern. Dafür gibt es laut Müller [16] die drei Basis-Technologien *Univariate Statistics*, *Model-Based Selection* und *Iterative selection*, die folgend erklärt werden.

#### Univariate Statistics

Bei der *Univariate Statistics* Methode wird jede Eigenschaft individuell betrachtet. Dabei wird geprüft, ob sich durch die jeweilige Eigenschaft Rückschlüsse auf das Ergebnis ziehen lassen. Nicht beachtet wird bei dieser Methode, ob die Eigenschaft in Kombination mit anderen Eigenschaften das Ergebnis gut oder schlecht repräsentiert. Dadurch ist *Univariate Statistics* zwar sehr performant, liefert jedoch nicht immer das optimale

Ergebnis.

#### Model-Based Selection

Die *Model-Based Selection* Methode betrachtet, im Gegensatz zur *Univariate Statistics*, alle Eigenschaften auf einmal, wodurch auch Zusammenhänge diverser Eigenschaften berücksichtigt werden können. Um dies zu gewährleisten, verwendet die *Model-Based Selection* Methode ein (in Abschnitt 3.4.7 beschriebenes) *Supervised Machine Learning Modell*, um die Wichtigkeit der einzelnen Features zu kalkulieren [16]. Das zur *Feature Selection* verwendete Modell kann dabei von dem verwendeten Hauptmodell abweichen.

#### Iterative Selection

Die *Iterative Selection* Methode verwendet wie auch die *Model-Based Selection* Methode ein *Supervised Machine Learning Modell*, um die Wichtigkeit der einzelnen Features zu kalkulieren. Anders als bei der *Model-Based Selection* wird hier jedoch nur jene eine Eigenschaft eliminiert, die den geringsten Einfluss auf das Ergebnis hat. Mit dem dadurch entstandenen, um eine Eigenschaft reduzierten, Modell, wird die Kalkulation erneut gestartet. Dieser Vorgang wird so lange wiederholt, bis die definierte Anzahl an Features erreicht ist [16]. Durch die große Anzahl an zu erzeugenden Modellen, ist diese Methode besonders rechenintensiv.

### 3.4.6 Feature Scaling

*Feature Scaling* ist ein weiterer Vorgang der Datenvorverarbeitung bei dem die einzelnen Eigenschaften in einen gemeinsamen Wertebereich gebracht und somit neu ausgerichtet werden. Beim *Feature Scaling* wird zwischen *Standardisierung* und *Normalisierung* unterschieden.

#### Standardisierung

Die *Standardisierung* führt dazu, dass alle Werte den Mittelpunkt 0 und eine Varianz von 1 haben. Dadurch rücken sie näher zusammen und haben einen einheitlichen Ursprung. Wie Raschka [21] erklärt, nehmen die Werte durch die *Standardisierung* die Form einer Normalverteilung an.

#### Normalisierung

Bei der *Normalisierung* werden die Werte in einen fix vorgegebenen Wertebereich skaliert, der üblicherweise zwischen 0 und 1 liegt. Die Werte rücken dabei noch näher zusammen als bei der *Standardisierung*, was Messfehler oder sogenannte Ausreißer entschärft [21]. Auf Abbildung 3.10 werden die Unterschiede von angewandter *Standardisierung* und *Normalisierung* aufgezeigt.

### 3.4.7 Supervised Machine Learning Algorithmen

Das Herzstück eines *Supervised Machine Learning* Modells ist der verwendete Klassifizierungs-Algorithmus, um die Daten zu verarbeiten und Rückschlüsse auf ein Ergebnis

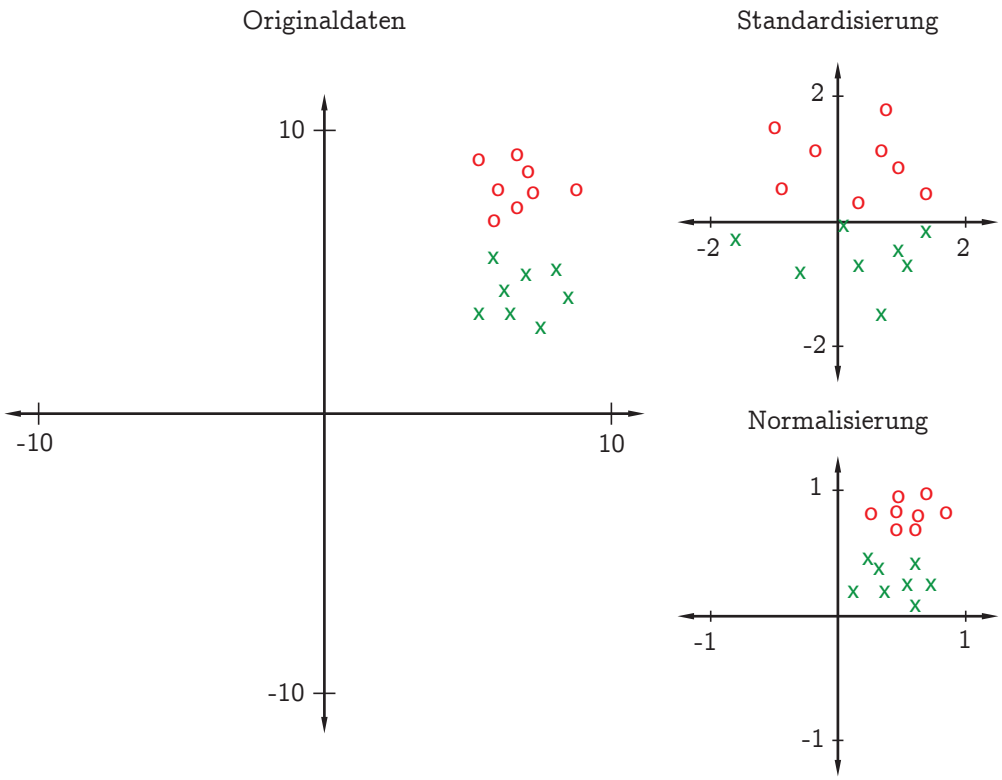



**Abbildung 3.10:** Die Effekte von *Feature Scaling* durch *Standardisierung* und *Normalisierung*.

zu ziehen. Da es unzählige verschiedene Problemstellungen gibt, die unterschiedliche Lösungsansätze erfordern, gibt es auch nicht *den* Klassifizierungsalgorithmus, der immer das beste Ergebnis liefert. Die Funktionsweisen, Eigenschaften und bevorzugten Einsatzgebiete bekannter Machine Learning Algorithmen werden in diesem Abschnitt aufgezeigt.

### K-Nearest-Neighbors

Der *K-Nearest-Neighbors*-Algorithmus, der während des Trainingsvorgangs ausschließlich alle Daten speichert, zählt zu den einfacheren Machine Learning Algorithmen. Ein solches Verfahren bezeichnet man auch als *Lazy Learning*, da der Großteil des Rechenaufwandes erst bei der Verwendung des Modells anfällt [5].

Um neue Daten klassifizieren zu können, wird der Abstand zu den umliegenden Daten gemessen und eine Mehrheitsentscheidung gefällt. Die Anzahl der zu berücksichtigenden Nachbarsdaten kann definiert werden und beeinflusst das Resultat des Algorithmus maßgeblich. Auf Abbildung 3.11 wird gezeigt, dass die Wahl mehrerer Nachbarn oft von Vorteil ist, da bei Beachtung nur eines Nachbarn Ausreißer oder Messfehler das Ergebnis verfälschen können. Da ein sehr naher Nachbar vermeintlich wichtiger ist als ein weit entfernter Nachbar, gibt es die Möglichkeit, die Entfernungen zu gewichten. Die

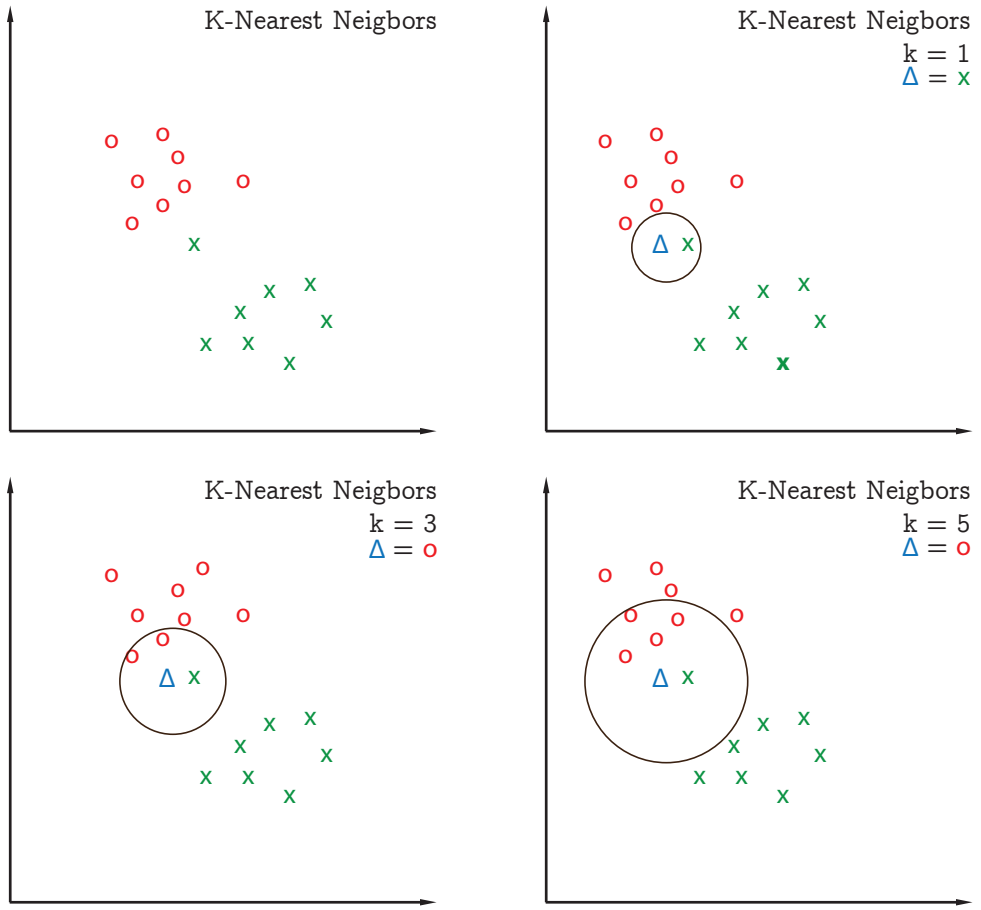



**Abbildung 3.11:** Klassifizierung einer Eigenschaft durch den *K-Nearest-Neighbors*-Algorithmus mit unterschiedlicher Anzahl an zu beachtenden Nachbarn. Die Anzahl der Nachbarn wird durch den Wert von „k“ definiert.

Stärke der Gewichtung kann über einen Gewichtungsparameter definiert werden [5].

Bei Anwendung des *K-Nearest-Neighbors*-Algorithmus muss darauf geachtet werden, dass bei einem, wie in Abbildung 3.11 gezeigten, Zwei-Klassen-Problem, die Anzahl an zu analysierenden Nachbarn ungerade gewählt wird. Grund dafür ist, dass es bei gerader Anzahl zu einem ausgeglichenen Verhältnis der Klassen kommen kann und der Algorithmus dadurch keine Entscheidung fällen kann. Aus demselben Grund darf die Anzahl an zu berücksichtigenden Nachbarn bei einem Multiplen-Klassen-Problem auch nicht der Anzahl der möglichen Klassen und keinem Vielfachen dieser entsprechen.

Ein großer Nachteil des *K-Nearest-Neighbors*-Algorithmus ist laut Theodoridis [24] die Geschwindigkeit. Da der Algorithmus bei der Vorhersage jede Entfernung zu jedem Nachbarn errechnen muss, wird der Rechenaufwand, gerade bei größeren Datenmengen, problematisch.

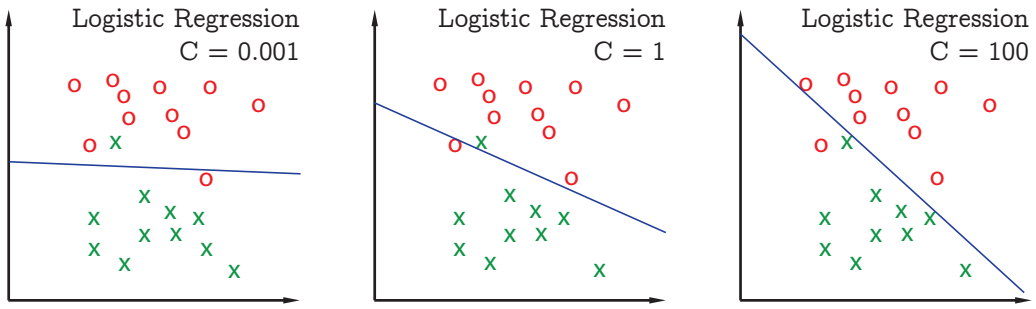



**Abbildung 3.12:** Effekt der *Regularisation* beim *Logistic Regression*-Algorithmus, definiert durch den Wert von „C“. Je höher die *Regularisation*, desto genauer wird die *Decision Boundary* den Daten angepasst. Eine hohe *Regularisation* geht somit mit dem Risiko des *Overfitting* einher.

### Logistic Regression

Der *Logistic Regression*-Algorithmus zählt zur Familie der Algorithmen für lineare Modelle. Um die Klassen unterscheiden zu können, wird anhand der gegebenen Eigenschaften eine Funktion gebildet, die die *Decision Boundary*, je nach Dimension, in Form einer Linie, Ebene oder Hyperebene erzeugt [16].

Bei *Logistic Regression* handelt es sich, im Gegensatz zu seinem Namen, um einen (wie in Abschnitt 3.4.2 beschrieben) *Klassifikations* und nicht um einen *Regressions*-Algorithmus. Ein wichtiger Parameter bei *Logistic Regression* bestimmt die *Regularisation* des Modells. *Regularisation* bedeutet, dass das (in Abschnitt 3.4.3 beschriebene) *Overfitting* bei einem Modell explizit eingeschränkt wird [16]. Ein hoher Regularisationswert bedeutet, dass der Algorithmus wenig *regularisiert* und die *Decision Boundary* genauer an die Trainingsdaten angepasst wird. Abbildung 3.12 zeigt diesen Effekt auf die *Decision Boundary*. Laut Müller [16] ist dieser Algorithmus zu bevorzugen, wenn die Anzahl der Eigenschaften sehr hoch ist, verglichen zu der Anzahl an Datensätzen.

### Decision Tree

Der *Decision Tree*-Algorithmus stellt die vorhandenen Daten in Form eines Entscheidungsbaumes dar. Ein großer Vorteil dieses Algorithmus ist, dass die resultierenden *if-else* Verzweigungen auch von Menschen leicht zu verstehen sind und daher nachgeprüft werden können.

Die Wahl des Wurzelknotens bzw. des zu verwendenden nächsten Knotens ist sehr wichtig, weshalb jene Eigenschaft ermittelt und ausgewählt wird, die sich als besonders aussagekräftig erweist und die Trainingsdaten am besten unterteilt [16]. Dieser Vorgang wird für jede Ebene wiederholt.

Beim Erlernen eines *Decision Trees* verzweigt sich der Entscheidungsbaum so weit, bis jede Eigenschaft eindeutig einer Klasse zugeordnet werden kann. Diese hohe Tiefe macht den Algorithmus nicht nur sehr rechenintensiv, sondern resultiert auch fast immer in *Overfitting*. Um diesem Problem entgegenzuwirken, kann die maximale Tiefe des

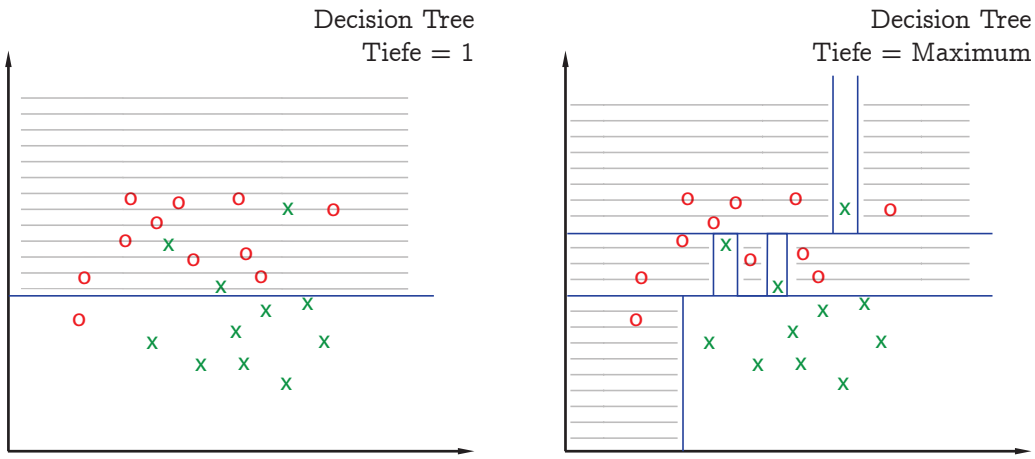



**Abbildung 3.13:** Effekt der erlaubten Tiefe eines durch den *Decision Tree*-Algorithmus erstellten Entscheidungsbaumes. Rechts kann das Auftreten von *Overfitting* sehr gut beobachtet werden, da das Modell zu exakt an die Testeigenschaften angepasst wurde.

Baumes eingeschränkt werden [11]. Diesen Vorgang nennt man *Pruning*. Auf Abbildung 3.13 ist deutlich zu sehen, dass das Modell mit hoher Tiefe den Testdaten zu genau angepasst wird, wodurch *Overfitting* auftritt.

Ein Vorteil von *Decision Tree*-Algorithmen ist, dass sie keine (in Abschnitt 3.4.5 erklärte) Vorverarbeitung der Daten benötigen [16].

### Naive Bayes Klassifikator

*Naive Bayes Klassifikatoren* sind den (in Abschnitt 3.4.7 erwähnten) linearen Modellen sehr ähnlich, benötigen jedoch einen geringeren Trainingsaufwand [16]. Anders als bei den linearen Systemen, wird bei den *Naive Bayes Klassifikatoren* von einer Unabhängigkeit der einzelnen Eigenschaften eines Datensatzes ausgegangen, was das Wort „naive“ im Namen erklärt. Es wird für jede Eigenschaft eine individuelle, einfache Klassenzugehörigkeit errechnet, was den Algorithmus sehr performant macht. Auch wenn die Annahme, dass die Eigenschaften keinen Zusammenhang besitzen, vor allem in Hinblick auf Praxiseinsätze, leichtsinnig klingen mag, liefern die *Naive Bayes Klassifikatoren* laut Domingos [4] oft sehr gute Ergebnisse.

Vorteile bietet der Einsatz von *Naive Bayes Klassifikatoren* vor allem bei sehr großen Datensätzen, wo der Einsatz von linearen Modellen einen zu hohen Rechenaufwand erfordern würde.

### Support Vektor Maschinen

Bei *Support Vektor Maschinen* unterscheidet man zwischen linearen und nicht-linearen *Support Vektor Maschinen*. Durch die nicht-linearen können komplexere Modelle erstellt werden, die oft auch jene Daten unterscheidbar machen, die auf den ersten Blick untrennbar wirken.

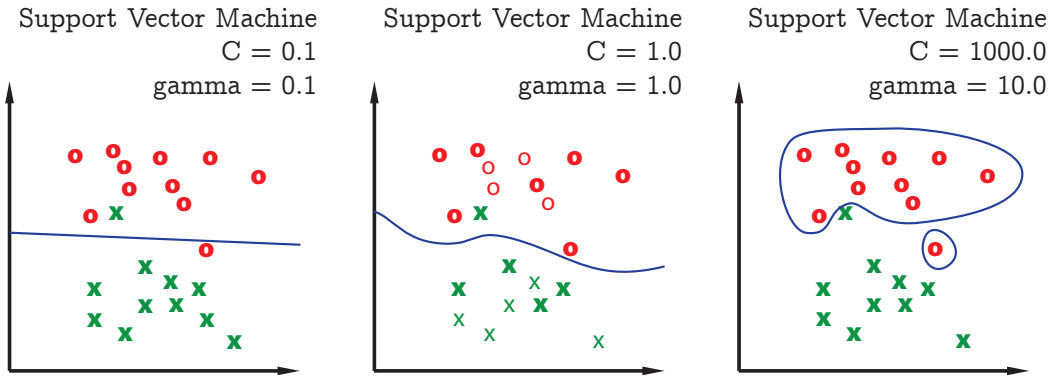



**Abbildung 3.14:** Effekt der Regularisation, definiert durch den „C"- Wert, und der Weite des Kernels, definiert durch den „gamma"-Wert, auf die *Decision Boundary* der *Support Vektor Maschinen*. Die Support Vektoren, welche die *Decision Boundary* maßgeblich prägen, werden fett und etwas größer dargestellt.

Zum Erzeugen der *Decision Boundary* zwischen den Klassen werden nur jene Datensätze berücksichtigt, die sich im Grenzbereich befinden bzw. die für die *Decision Boundary* die größte Wichtigkeit aufweisen. Diese definieren die Grenze eindeutig, haben exakt den gleichen Abstand zu ihr und werden Support-Vektoren genannt [5]. Diese Support-Vektoren sind auch die Namensgeber für die *Support Vektor Maschinen*.

Da Eigenschaften oft nicht linear voneinander unterscheidbar sind, wenden *Support Vektor Maschinen* einen Trick an. Dabei werden die Eigenschaften anhand einer sogenannten *Kernel-Funktion*, gegebenenfalls wiederholt, in einen höherdimensionalen Raum überführt, in dem sie linear trennbar werden. Laut Ertel [5] sind Daten immer durch Transformation des Vektorraums voneinander trennbar, sofern sie keine Widersprüche enthalten.

Auch bei den *Support Vektor Maschinen* gibt es Schrauben, an denen gedreht werden kann, um den Algorithmus anzupassen und das erzielte Ergebnis zu verbessern. Hierbei sind vor allem die Werte für die (in Abschnitt 3.4.7 beschriebene) *Regularisation* und für die Weite des Kernels wichtig [16]. Ein niedriger Wert für die Weite des Kernels bewirkt, dass der Radius der Kernelfunktion groß ist und daher sehr viele Punkte zur Erstellung der *Decision Boundary* herangezogen werden. Mit der Erhöhung des Wertes fokussiert sich die Kernelfunktion immer mehr auf die näheren Punkte. In Abbildung 3.14 wird der Effekt dieser beiden Parameter auf die *Decision Boundary* dargestellt.

Da *Support Vektor Maschinen* zu den rechenintensiven Algorithmen zählen, sind sie für besonders große Datensätze nicht geeignet. Die (in Abschnitt 3.4.5 erklärte) Vorverarbeitung ist für ein gut funktionierendes Modell auf Basis von *Support Vektor Maschinen* unbedingt notwendig [16]. Wurde diese Vorverarbeitung gründlich durchgeführt, liefern Modelle auf Basis von*Support Vektor Maschinen* jedoch sehr vielversprechende Ergebnisse.

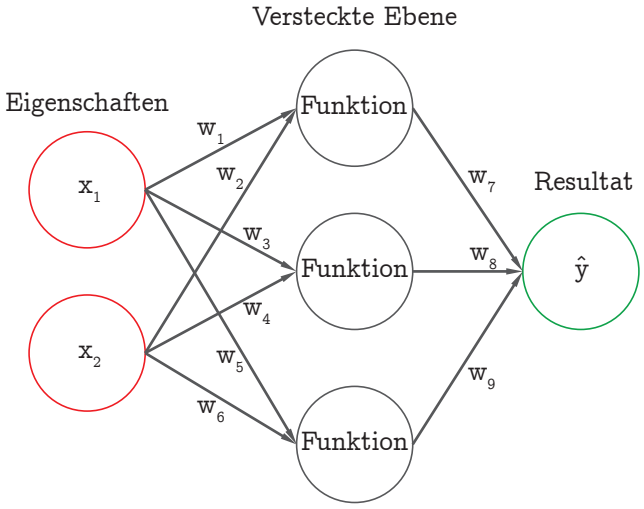



**Abbildung 3.15:** Aufbau eines *Neuronalen Netzwerks*. Jede Verbindung zwischen Eigenschaft und Unterebene hat eine berechnete Gewichtung „w“, mit der ihr Wert multipliziert wird. Bei den Unterebenen wird auf die summierten, gewichteten Eigenschaften eine Funktion angewandt, wodurch das Modell nicht-linear wird. Dieser Vorgang wird je nach Anzahl an Ebenen wiederholt.

### Neuronale Netzwerke

*Neuronale Netzwerke* besitzen die Fähigkeit, eine große Anzahl an Daten gut zu verarbeiten und extrem komplexe Modelle zu erstellen. Sie können als sich wiederholende lineare Modelle beschrieben werden [16]. Ein lineares Modell in Form der *Logistic Regression* wurde in Abschnitt 3.4.7 erklärt.

Bei *Neuronalen Netzwerken* werden zusätzliche Zwischenebenen eingeführt. Diese Ebenen werden, da sie von außen nicht sichtbar sind, versteckte Ebenen genannt und bestehen aus mehreren Unterebenen. Der Wert jeder Eigenschaft wird zu jeder angrenzenden Unterebene mit einer berechneten Gewichtung multipliziert und weitergereicht. Bei den Unterebenen werden die eingehenden, gewichteten Eigenschaften dann aufsummiert. Auf diese Summe wird eine Funktion angewandt, die das *Neuronale Netzwerk* nicht-linear macht und der resultierende Wert wird abschließend weitergereicht. Die Anzahl der versteckten Ebenen und Unterebenen ist dabei definierbar. Besteht ein Modell aus mehreren Ebenen, werden diese nacheinander durchlaufen. Der Aufbau eines *Neuronalen Netzwerks* wird auf Abbildung 3.15 dargestellt.

Modelle auf Basis von *Neuronalen Netzwerken* sind sehr empfindlich auf die (in Abschnitt 3.4.5 vorgestellte) Vorverarbeitung der Eigenschaften. Alle Eigenschaften sollten laut Müller [16] daher einen einheitlichen Mittelpunkt bei 0 und eine einheitliche Varianz aufweisen, was durch die *Standardisierung* der Daten bewerkstelligt werden kann. Die (in Abschnitt 3.4.7 beschriebene) *Regularisation* des Modells, die Anzahl an versteckten Ebenen und die Anzahl der darin enthaltenen Schichten kann definiert werden, um die Komplexität des Modells zu steuern.

Obwohl eine genaue Vorverarbeitung der Daten notwendig ist und Modelle basierend auf *Neuronalen Netzwerken* oft eine sehr lange Trainingsdauer mit sich bringen, sind sie

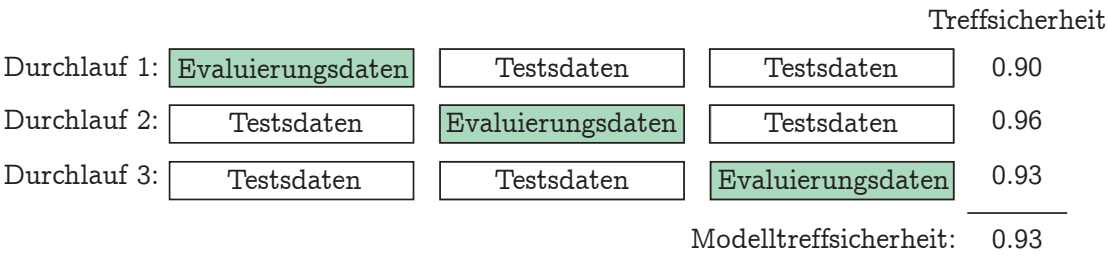



**Abbildung 3.16:** Ablauf und Aufteilung der Daten bei einer dreifachen *Cross Validation*. Die Modelltreffsicherheit errechnet sich durch den Mittelwert der Teiltreffsicherheiten.

sehr weit verbreitet und liefern, sorgfältig konfiguriert, oft bessere Ergebnisse als ihre Konkurrenzalgorithmen [16].

### 3.4.8 Modell-Evaluierung

Da es viele verschiedene Machine Learning Algorithmen gibt, die alle auf ihre eigene Art und Weise auf zugeführte Daten reagieren, ist eine möglichst gute Einschätzung der Erfolgschancen beim Einsatz eines bestimmten Algorithmus wichtig. Viele Algorithmen reagieren sensibel auf die übergebenen Parameter, weshalb es sehr erstrebenswert ist, diese so zu wählen, dass ein möglichst perfektes Modell daraus resultiert. Auch wäre oft ein tieferer Einblick in die Zusammenstellung der resultierenden Treffsicherheit des trainierten Modells interessant. Die Methoden der *Modell-Evaluierung*, die diese Informationen liefern, werden folgend erklärt.

#### Cross Validation

Durch den Einsatz von *Cross Validation* kann herausgefunden werden, wie gut ein Algorithmus generalisiert, wenn ihm gewisse Daten zugeführt werden. Standardmäßig werden bei Supervised Klassifiaktions Problemen die Trainingsdaten zum Ermitteln der Treffsicherheit eines Modells in zwei Teile zu 75% und 25% aufgeteilt. Der größere Teil wird zum Trainieren und der kleinere Teil zum Evaluieren des Modells verwendet. Um bei dieser Evaluierung einen aussagekräftigeren Wert zu erlangen, wird die *Cross Validation* eingesetzt. Dabei kann definiert werden, in wie viele Teile die vorhandenen Daten unterteilt werden sollen. Anschließend wird die Evaluierung so oft ausgeführt, wie die Daten unterteilt wurden. Bei jeder Durchführung wird ein anderer Teilbereich als Evaluierungsdatensatz verwendet [1]. Dadurch erhält man mehrere Treffsicherheiten, von denen dann der Mittelwert als allgemeine Treffsicherheit für den gewählten Algorithmus herangezogen wird. Auf Abbildung 3.16 werden dieser Ablauf und die Aufteilung der Daten grafisch dargestellt. Die einzelnen Teiltreffsicherheiten verraten außerdem, wie stabil das trainierte Modell ist. Stark variierende Treffsicherheiten lassen auf ein instabiles Modell schließen.

Ein Vorteil der *Cross Validation* ist die bessere Ausnutzung der vorhandenen Trainingsdaten. Während beim Standardvorgang nur 75% der Daten zum Trainieren des Modells herangezogen werden, können beispielsweise bei einer Unterteilung in zehn Blö-

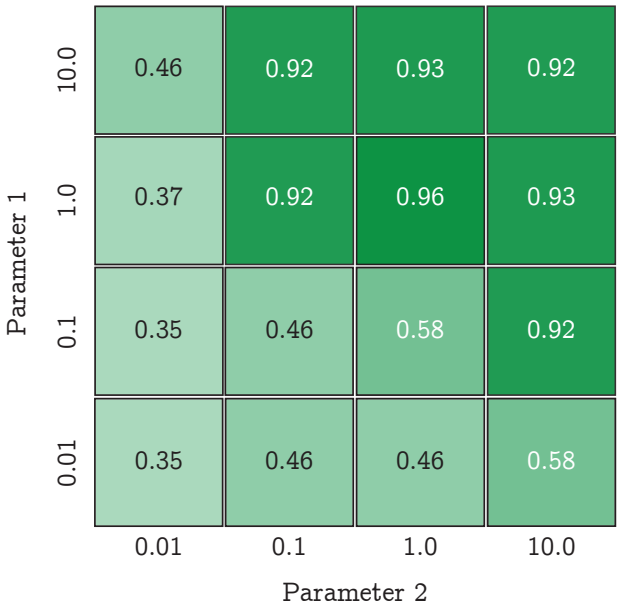



**Abbildung 3.17:** *Heatmap* der gemittelten Werte des *Grid Search* mit *Cross Validation*.

cke 90% der Daten zum Training verwendet werden. Da bei *Cross Validation* mehrere Modelle erzeugt werden müssen, wird auch der Rechenaufwand erhöht [16].

### Grid Search

Viele Algorithmen, die bei Machine Learning zum Einsatz kommen, können durch diverse Anpassungen die erzielten Ergebnisse verbessern. Durch die übergebenen Parameter wird die Arbeitsweise der Algorithmen definiert. Herauszufinden welche Parameter, bzw. welche Parameterkombinationen das beste Ergebnis liefern, ist daher ein wichtiges Ziel der Modell-Evaluierung. Ein einfaches, händisches Ausprobieren der unterschiedlichen Parameterwerte würde einen hohen Aufwand darstellen. Dank der *Grid Search* wird eine Evaluierung der Algorithmen, abhängig von ihren Parametern, automatisiert ermöglicht. Sehr verbreitet ist laut Müller [16] die Kombination von *Grid Search* mit *Cross Validation*, da dadurch eine noch aussagekräftigere Treffsicherheit ermittelt werden kann.

Auch das *Grid Search* Verfahren ist sehr rechenintensiv, weshalb es sich empfiehlt, zuerst mit wenigen Parametern zu starten und sich die gemittelten *Cross Validation* Treffsicherheiten anzusehen [16]. Diese Darstellung kann in Form einer *Heatmap* erfolgen, wie sie beispielsweise auf Abbildung 3.17 dargestellt wird. Jede Kachel repräsentiert dabei das gemittelte Ergebnis einer *Cross Validation* mit den auf der X- und Y-Achse angegebenen Parameterwerten. Anzumerken ist bei dieser Abbildung, dass die Werte gut gewählt wurden, da der Optimalwert nicht am Rand der *Heatmap* positioniert ist. In diesem Fall muss der Wertebereich nicht erhöht werden, stattdessen könnte er noch feiner aufgelöst werden, um so ein genaueres Ergebnis zu erzielen.

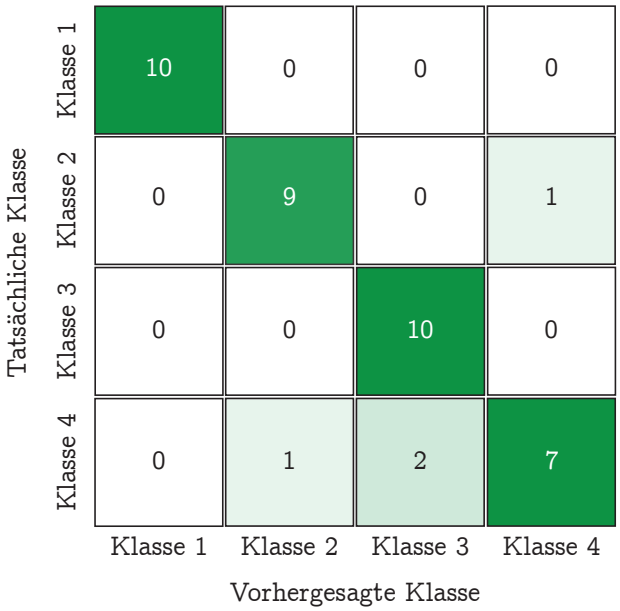



**Abbildung 3.18:** Darstellung einer *Confusion Matrix*, um die tatsächlich vergebenen Klassen zu visualisieren. Es wurden von jeder Klasse 10 Datensätze getestet.

### Confusion Matrix

Bei der Anwendung der vorangegangenen Methoden erhält man eine Treffsicherheit in Form einer Zahl, die einen Prozentsatz repräsentiert. Diese verrät zwar, wie gut das Modell neue Daten vorhersagen kann, zeigt jedoch nicht, wo sich die Probleme befinden, die eine perfekte Vorhersage verhindern. Um diese Information zu erhalten, kann eine *Confusion Matrix* erzeugt werden, die aufzeigt, welche Klasse vorhergesagt wurde und welche Klasse tatsächlich richtig gewesen wäre [16]. Ein Beispiel dafür ist in Abbildung 3.18 zu sehen. Aus der *Confusion Matrix* geht hervor, dass alle Datensätze, die die Klasse 1 und 3 repräsentieren, richtig vorhergesagt wurden. Ein Problem gab es bei der Klassifikation der Klasse 2, da diese einmal als Klasse 4 vorhergesagt wurde. Am meisten Probleme bereitete diesem Beispielmodell die Vorhersage der Klasse 4. Dabei wurde fälschlicherweise einmal die Klasse 2 und gleich zweimal die Klasse 3 vorhergesagt.

### Classification Report

Eine weitere Möglichkeit, jenen Bereich herauszufinden, der dem Modell Probleme bereitet, ist die Erstellung eines *Classification Reports*. Dabei wird eine Tabelle erzeugt, die für jede Klasse spezifische Werte errechnet. Der Wert in der ersten Spalte spiegelt die Präzision wider. Sie zeigt an, wie viel Prozent der vom Modell als diese Klasse vorhergesagten Datensätze auch tatsächlich dieser Klasse angehörten. Die zweite Spalte repräsentiert den Recall-Wert, der aussagt, wie viel Prozent der vom Modell als diese Klasse vorhergesagten Datensätze eigentlich einer anderen Klasse angehören und somit fälschlicherweise dieser Klasse zugeordnet wurden. Die dritte Spalte zeigt den *f1-score*,

**Tabelle 3.3:** Der *Classification Report* gibt noch genauere Einblicke in die Klassifizierung des Modells als die *Confusion Matrix*.

| | Präzision | Recall | f1-score | Anzahl |
|---|---|---|---|---|
| Klasse 1 | 1.00 | 1.00 | 1.00 | 10 |
| Klasse 2 | 0.90 | 0.90 | 0.90 | 10 |
| Klasse 3 | 0.83 | 1.00 | 0.92 | 10 |
| Klasse 4 | 0.88 | 0.70 | 0.79 | 10 |

der das harmonische Mittel aus den vorangegangenen Werten von Präzision und Recall darstellt. Abschließend wird noch angezeigt, wie oft die Klasse in diesem Test vorhanden war. Ein Beispiel für einen *Classification Report* ist in der Tabelle 3.3 zu sehen. Aus dieser Tabelle geht hervor, dass die Klasse 4 dem Modell die größten Probleme bereitet.

# Kapitel 4

# Umsetzung

Die Vokalerkennung lässt sich im Wesentlichen in drei Teile unterteilen. Begonnen wird mit der Digitalisierung von gesprochenen Vokalen. Daraus werden anschließend aussagekräftige Merkmale extrahiert. Diese Merkmale werden dann in ein Machine Learning Modell eingelernt, das schlussendlich durch Zufuhr von dem Modell unbekannten Eigenschaften, Rückschlüsse auf den gesprochenen Vokal ziehen kann.

Wie in Abschnitt 3.3.4 beschrieben, zählen nicht nur „a“, „e“, „i“, „o“ und „u“, sondern auch die Zwie- und Umlaute zu den Vokalen der deutschen Sprache. Diese Umsetzung konzentriert sich auf die Hauptvokale, weshalb die Zwie- und Umlaute nicht berücksichtigt werden.

## 4.1 Technologie

### 4.1.1 Praat

Das seit 1992 von Paul Börsma und David Weenink am *Institute of Phonetic Sciences* an der Universität von Amsterdam entwickelte Audioanalyseprogramm *Praat* wurde zur Digitalisierung und Verarbeitung der Sprachdaten verwendet. *Praat* ist laut Börsma [2] das umfangreichste verfügbare Audioanalyseprogramm und steht zur kostenlosen Verwendung zur Verfügung. Mit Hilfe von *Praat* können Audiosignale unter anderem grafisch dargestellt, analysiert und konvertiert werden. Abbildung 4.1 zeigt die grafische Darstellung einer Vokalreihe inklusive Markierungen wichtiger Eigenschaften durch *Praat*.

Ein großer Vorteil von *Praat* ist die vorhandene Skript-Schnittstelle. Dadurch können durch die eigene *Praat*-Skriptsprache programmatisch und automatisiert Operationen ausgeführt werden.

### 4.1.2 Python

Wie Müller [16] erklärt, hat sich Python zu *der* Programmiersprache für Datenverarbeitungsanwendungen entwickelt, da sie die Stärken von Universalprogrammiersprachen mit der einfachen Verwendung von Skriptsprachen vereint. Die klare Syntax macht Python-Programme leicht verständlich und dank der großen Verbreitung sind viele Beispiele und gute Dokumentationen verfügbar. Für die meisten Anwendungsbereiche exis-

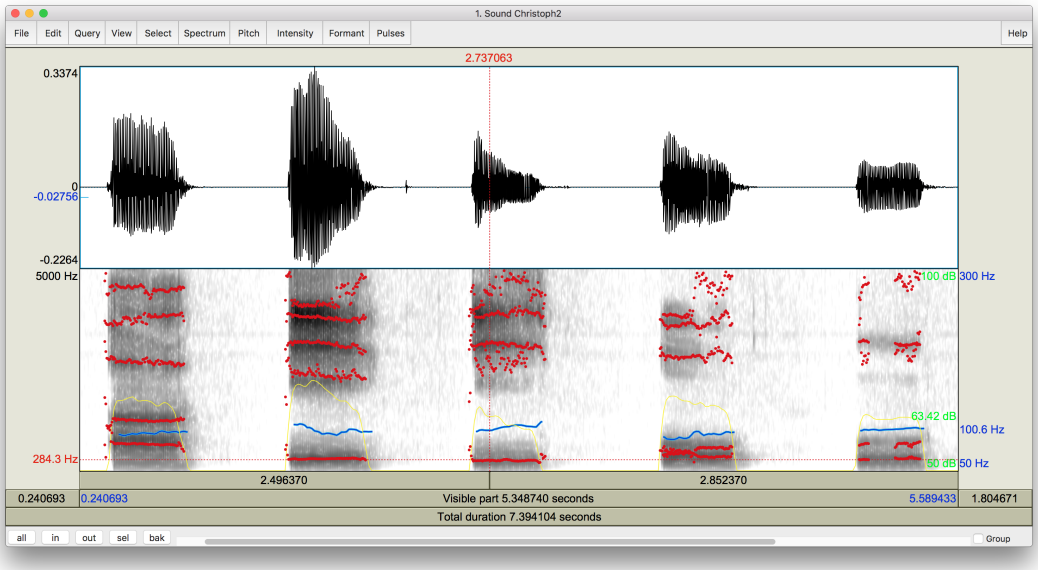



**Abbildung 4.1:** Grafische Darstellung einer Waveform und eines Spektrogramms einer Vokalreihe mit eingezeichneten Formanten, Intensitäten und Grundfrequenzen, durch das Audioanalyseprogramm *Praat*.

tieren außerdem umfangreiche Bibliotheken. Ein Hauptvorteil von Python ist die direkte Interaktionsmöglichkeit mit dem Code über das Terminal, was ein schnelles und einfaches Testen ermöglicht. Laut Harrington [7] ist der einzige Nachteil von Python gegenüber den Alternativprogrammiersprachen Java oder C die Geschwindigkeit. Folgend werden die verwendeten Python-Bibliotheken kurz erklärt.

### `scikit-learn`

Laut Pedregosa [17] integriert die `scikit-learn` Bibliothek eine Vielzahl an aktuellen Machine Learning Algorithmen, wobei besonders auf die einfache Verwendung, Geschwindigkeit, Dokumentation und API-Konsistenz geachtet wurde. Auch Müller [16] nennt `scikit-learn` die am meisten verwendete Python Bibliothek für Machine Learning. Diese Bibliothek ermöglicht es, alle notwendigen Vorverarbeitungsmethoden, Machine Learning Algorithmen und Evaluierungsmethoden, die für diese Umsetzung notwendig sind, auszuführen.

### `matplotlib`

Die `matplotlib` ist eine sehr umfangreiche Bibliothek, die zur grafischen Aufbereitung von Daten dient. Sie unterstützt eine Vielzahl an unterschiedlichen Diagrammen und ermöglicht es daher auf einfache Weise, 2D- oder 3D-Darstellungen der vorhandenen Eigenschaften zu generieren.

`statistics`

Für die Berechnung der Eigenschaften sind statistische Funktionen, wie unter anderem die Ermittlung der Varianz, notwendig. Hierfür kommt die `statistics` Bibliothek zum Einsatz, die alle benötigten Funktionen mit sich bringt [22].

`json`

Um die Eigenschaften persistieren zu können, werden sie im JSON-Format exportiert. Die `json` Bibliothek bietet Möglichkeiten, das vorhandene Datenkonstrukt sehr komfortabel in eine JSON-Datei zu exportieren, bzw. diese wieder zu importieren.

## 4.2 Digitalisierung

Ausgangspunkt der Umsetzung ist die Digitalisierung der gesprochenen Laute. Die Aufnahmen erfolgten über ein internes Mikrofon eines *Apple MacBookPro Retina* mit einer Abtastfrequenz von 44100 Hertz und dem Aufnahmemodus Mono. Zur softwareseitigen Aufnahme und Digitalisierung wurde das (in Abschnitt 4.1.1 beschriebene) Audioanalyseprogramm *Praat* verwendet. Abbildung 4.2 zeigt das Aufnahmefenster von *Praat*.

### 4.2.1 Testdaten

Die gesammelten Testdaten bestehen aus isoliert gesprochenen Vokalreihen. Die einzelnen Vokale wurden durch eine kurze Sprechpause getrennt. Insgesamt wurden 105 Vokale von weiblichen und männlichen Sprechern im Alter zwischen 24 und 60 Jahren aufgezeichnet. Das Aufnahmeumfeld war immer ein Raum mit wenig Nebengeräuschen.

## 4.3 Eigenschaften der Vokale

Um wichtige Eigenschaften und deren zeitliche Veränderungen aus dem Sprachsignal zu extrahieren, wurden *Praat*-Skripte erstellt, die dies möglichst automatisiert bewerkstelligen.

### 4.3.1 Kennzeichnung von Vokalen

Damit das System weiß, welcher Vokal welchem Bereich des Audiosignals entspricht, müssen die Vokale gekennzeichnet werden. In *Praat* werden daher der Vokalname sowie Start- und End-Zeitpunkt in einer *TextGrid* Datei gespeichert, die denselben Dateinamen wie die Audiodatei trägt und durch die Dateiendung *.textgrid* von der Audiodatei unterscheidbar ist. Das *Praat*-Skript lädt alle Audiodateien und generiert, nach manueller Kennzeichnung, automatisiert die dazugehörigen *TextGrid* Dateien. Abbildung 4.3 zeigt ein Bearbeitungsfenster mit bereits gekennzeichneten Vokalbereichen.

Bei den Markierungen wurde darauf geachtet, dass jene Bereiche des Vokals markiert werden, bei denen die Formanten eine möglichst hohe Konstanz aufweisen, da diese den Vokal am besten charakterisieren. Um den Einfluss der gesamten Vokalreihe auf einzelne Vokale zu verhindern, wurden sie automatisiert aus dem Gesamtsignal herausgeschnitten und als eigene isolierte Dateien abgespeichert.

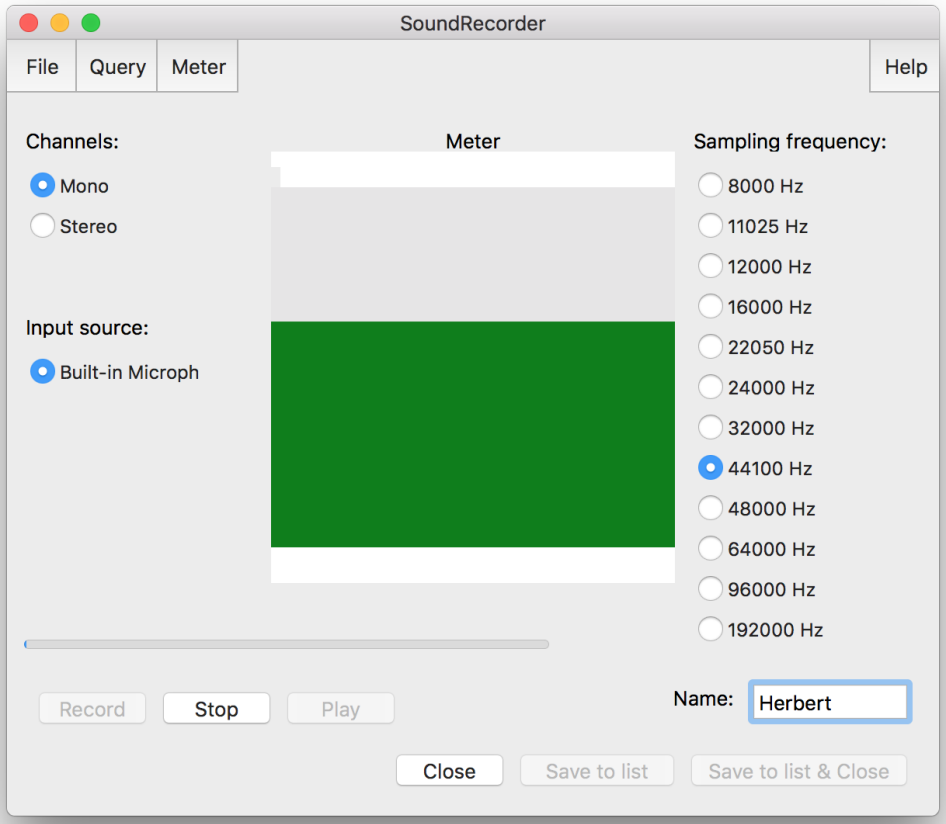



**Abbildung 4.2:** Aufnahmevorgang durch das Audioanalyseprogramm *Praat* mit den gewählten Einstellungen. Der grüne Balken in der Mitte repräsentiert die momentane Intensität der Aufnahme.

### 4.3.2 Extraktion von Merkmalen

Als Basis für die Extraktion von Merkmalen dienen die zuvor erstellten isolierten Audiodateien der Vokale. Das erstellte Praat-Skript extrahiert daraus mehrere Eigenschaften, welche folgend aufgelistet werden.

#### Grundfrequenz

Aus jedem Vokalbereich wurde die durchschnittliche Grundfrequenz ermittelt. Obwohl, wie Kumar [12] schreibt, die Grundfrequenz in erster Linie der Sprecheridentifikation dient, wurde sie dennoch als Eigenschaft extrahiert, da sie eine grundlegende Eigenschaft des Vokals darstellt.

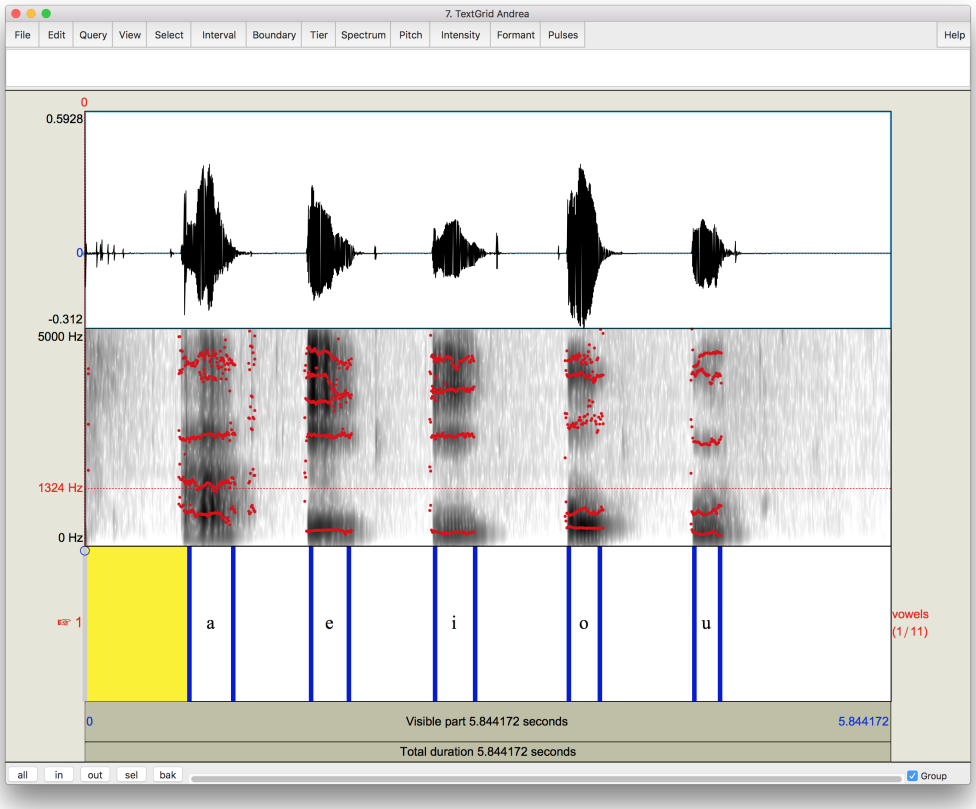



**Abbildung 4.3:** Kennzeichnung der Vokale in der Waveform-Darstellung bzw. im Spektrogramm im Audioanalyseprogramm *Praat*.

### Intensität

Um die Unterschiede der Intensität zwischen den Vokalen analysieren zu können, wurde der Leistungspegel des gesamten Vokals ermittelt. Dies kann in *Praat* durch die Transformation des Signals in das logarithmische Leistungsdichtespektrum vollzogen werden.

### Formanteneigenschaften

Da die Formanten zu den wichtigsten Charakteristiken der Vokale zählen [23], wurden diese besonders genau betrachtet. In 0,01 Sekunden-Schritten wurden die Werte der Formanten ermittelt. Für jedes Zeitfenster wurden die Frequenz und die Bandbreite der ersten drei Formanten extrahiert. Des Weiteren wurde in jedem Zeitfenster die Intensität ermittelt, welche das Signal bei den jeweiligen Formantenfrequenzen aufweist.

## 4.4 Aufbereitung der Daten

Die Grundfrequenz sowie die durchschnittliche Intensität des Vokals konnten direkt als Eigenschaften übernommen werden, während andere erst aus den vorhandenen Werten

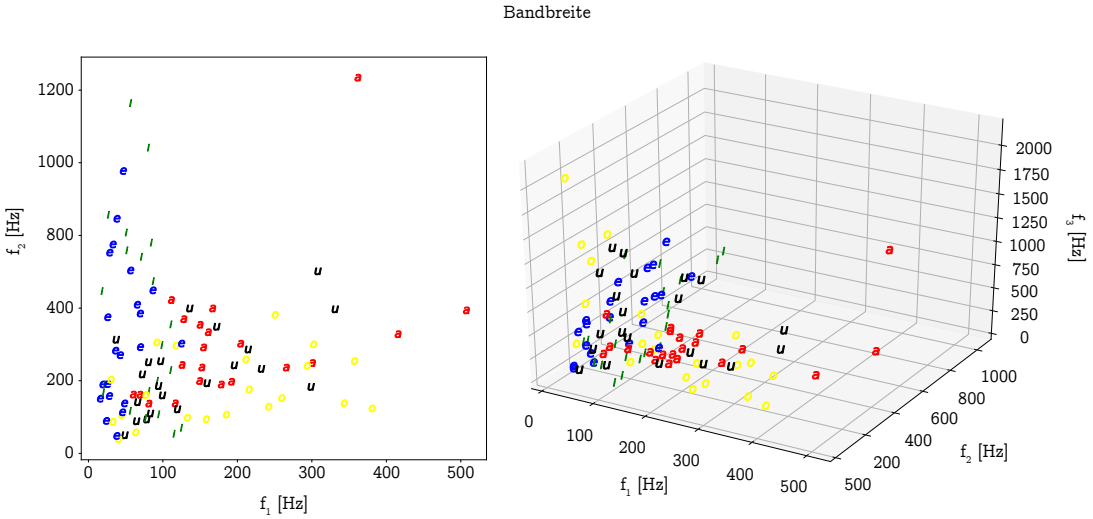



**Abbildung 4.4:** 2D- und 3D-Ansicht des arithmetischen Mittelwertes der Bandbreite der Formantenfrequenzen.

kombiniert bzw. errechnet werden mussten.

### 4.4.1 Berechnung der Formanteneigenschaften

Da das Hauptaugenmerk auf die Formanten gelegt wurde, waren deren extrahierte Eigenschaften Ausgangspunkt für mehrere Berechnungen, welche folgend aufgelistet werden.

#### Bandbreite der Frequenzen

Die Bandbreite wurde für jeden Formanten eines jeden Vokals im 0,01 Sekunden-Takt extrahiert. Der arithmetische Mittelwert dieser Werte wurde für jeden Formanten berechnet und als Eigenschaft gespeichert. Abbildung 4.4 zeigt ein 2D- und ein 3D-Diagramm mit den erhaltenen Werten. Eine klare Trennung der Vokale mit freiem Auge ist dabei nicht möglich. Wie beispielsweise in Abschnitt 3.4.7 beschrieben, kann man dadurch aber nicht darauf schließen, dass die Eigenschaft keine wichtigen Informationen zur Vokalklassifikation enthält. Es ist nicht ausgeschlossen, dass beispielsweise durch Überführen der Daten in einen höherdimensionalen Raum klare Grenzen entstehen.

#### Mittelwert der Frequenzen

Auch die Frequenzen der Formanten wurden im selben Intervall extrahiert, weshalb auch davon der arithmetische Mittelwert errechnet und gespeichert wurde. Eine grafische Aufbereitung dieser Werte ist auf Abbildung 4.5 zu betrachten. Dabei ist mit freiem Auge ersichtlich, dass sich die Vokale alle in ähnlichen Bereichen positionieren. Lediglich „e“ und „i“, sowie „o“ und „u“ können hier nicht klar voneinander abgegrenzt werden.

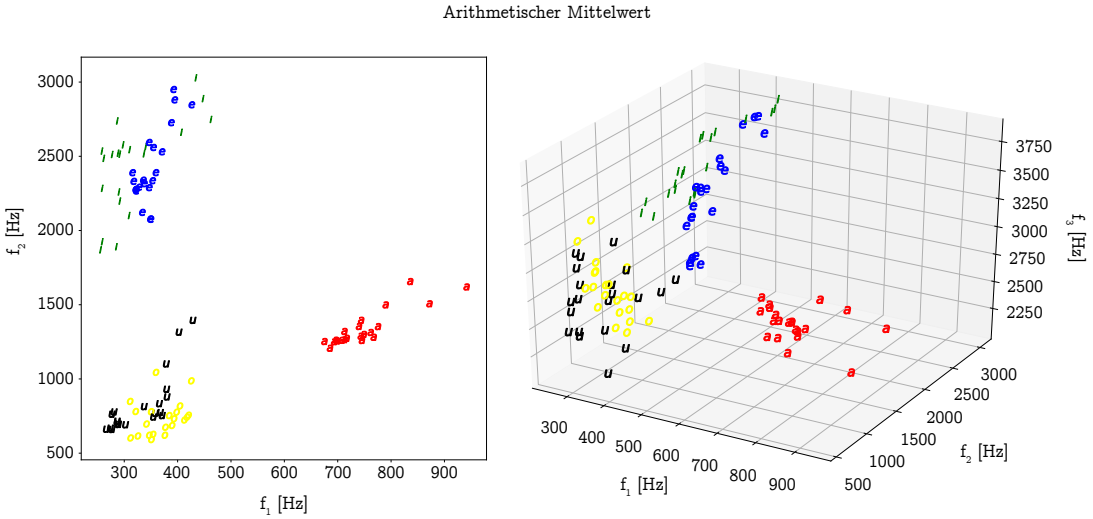



**Abbildung 4.5:** 2D- und 3D-Ansicht des arithmetischen Mittelwertes der Formantenfrequenzen.

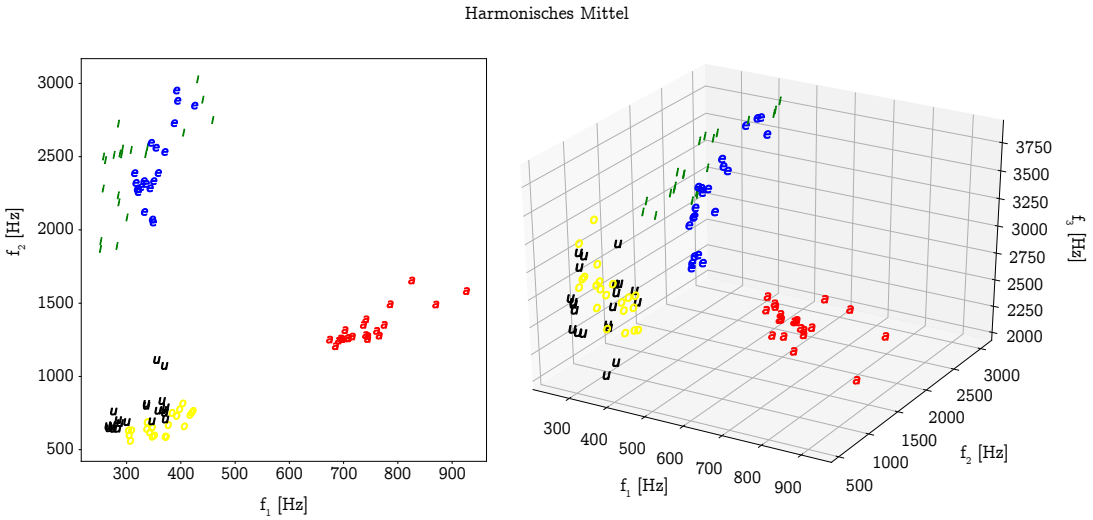



**Abbildung 4.6:** 2D- und 3D-Ansicht des harmonischen Mittelwertes der Formantenfrequenzen.

### Harmonisches Mittel der Frequenzen

Zusätzlich zum arithmetischen Mittelwert wurde auch das harmonische Mittel berechnet, um die Anzahl an repräsentativen Eigenschaften zu erhöhen. Wie in Abbildung 4.6 zu sehen ist, scheint dadurch die Grenze zwischen „o“ und „u“ etwas klarer zu ziehen zu sein.

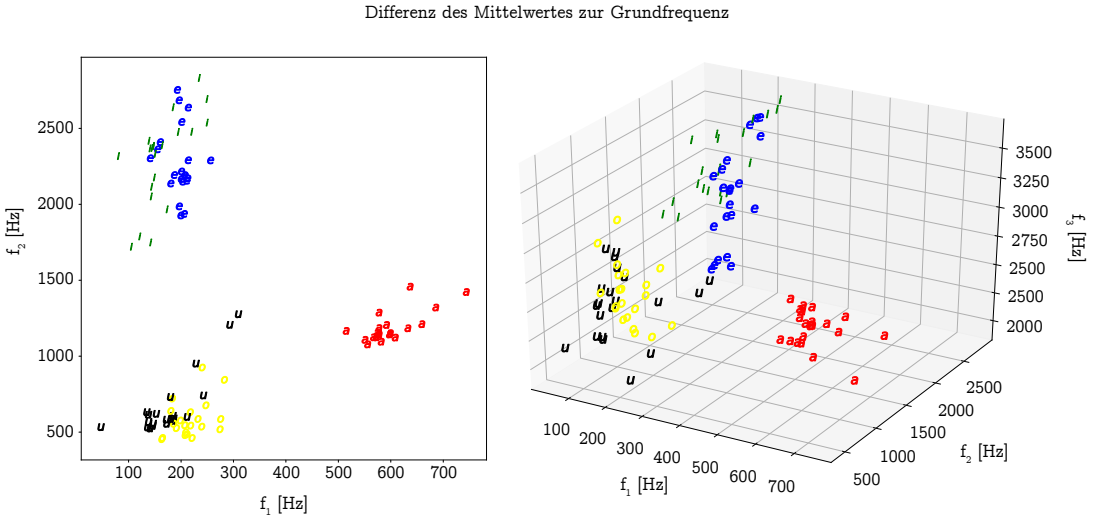



**Abbildung 4.7:** 2D- und 3D-Ansicht der Differenz des Mittelwertes der Formanten zur Grundfrequenz.

### Differenz des Mittelwertes zur Grundfrequenz

Um die Abhängigkeit der durchschnittlichen Formantenfrequenzen zur Grundfrequenz des gesprochenen Vokals ermitteln zu können, wurde die Differenz des arithmetischen Mittelwertes zur Grundfrequenz kalkuliert. Abbildung 4.7 zeigt, dass auch hier die Aufteilung ähnlich wie bei den vorangegangenen Mittelwerten ist.

### Standardabweichung

Zur Erlangung der durchschnittlichen Entfernung der einzelnen Frequenzwerte zum Mittelwert wurde die Standardabweichung berechnet. Wie in Abbildung 4.8 zu sehen ist, ist keine offensichtliche Trennung gegeben. Auch hier gilt jedoch wieder, dass dies nicht bedeuten muss, dass die Eigenschaften nutzlos sind.

### Varianz der Formantenfrequenzen

Wie stark die einzelnen Frequenzwerte um den Mittelwert streuen, wird durch die Varianz berechnet. Auf Abbildung 4.9 wird gezeigt, dass die Formantenfrequenzen teilweise sehr stark streuen, wodurch einige Extremwerte entstehen. Dennoch können die Eigenschaften wertvolle Informationen enthalten. Wie in Abschnitt 3.4.6 beschrieben wurde, gibt es Möglichkeiten, die Daten zu skalieren, wodurch die Extremwerte keine so große Rolle mehr spielen.

### Durchschnittliche Intensität der Formantenfrequenzen

Aus der Gesamtintensität des Vokals wurde der Mittelwert jener Intensität berechnet, welche bei den Formantenfrequenzen auftritt. Die grafische Darstellung auf Abbildung 4.10 zeigt, dass sich die Bereiche der Vokale sehr stark überlagern, weshalb mit freiem Auge keine eindeutige Trennung möglich ist.

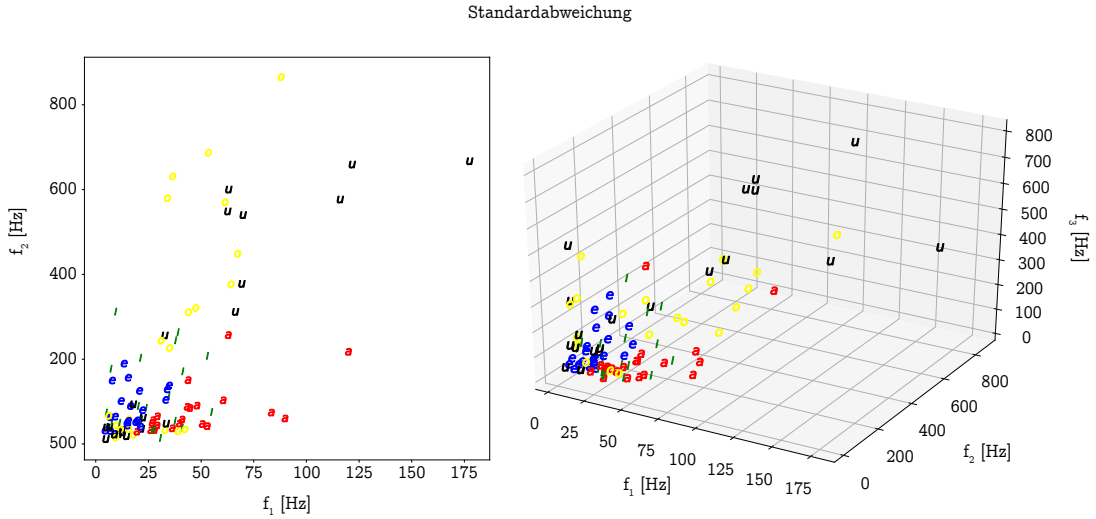



**Abbildung 4.8:** 2D- und 3D-Ansicht der Standardabweichung der Formantenfrequenzen.

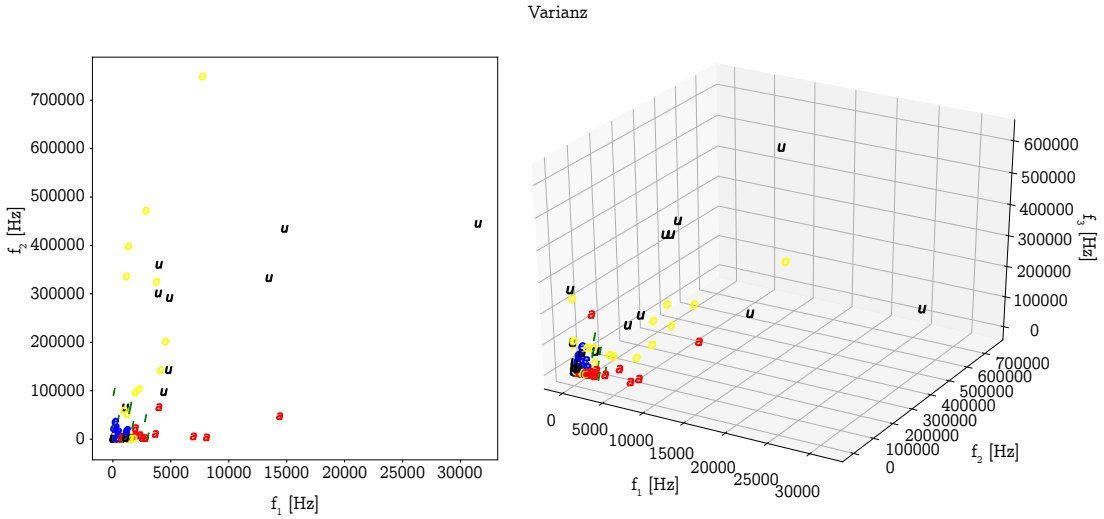



**Abbildung 4.9:** 2D- und 3D-Ansicht der Varianz der Formantenfrequenzen.

### Differenz der Formantenintensität zur Gesamtintensität

Um das Verhältnis der Formantenintensität zur durchschnittlichen Intensität des gesamten Vokals zu erhalten, wurde deren Differenz ermittelt. Durch den Bezug zur durchschnittlichen Intensität des Signals werden die sprecher- und aufnahmespezifischen Intensitätsunterschiede entschärft. Abbildung 4.11 zeigt, dass dadurch die Vokale auch mit freiem Auge in Bereiche einzugliedern sind, auch wenn sich, ähnlich wie bei den Mittelwerten, einige Bereiche überlagern.

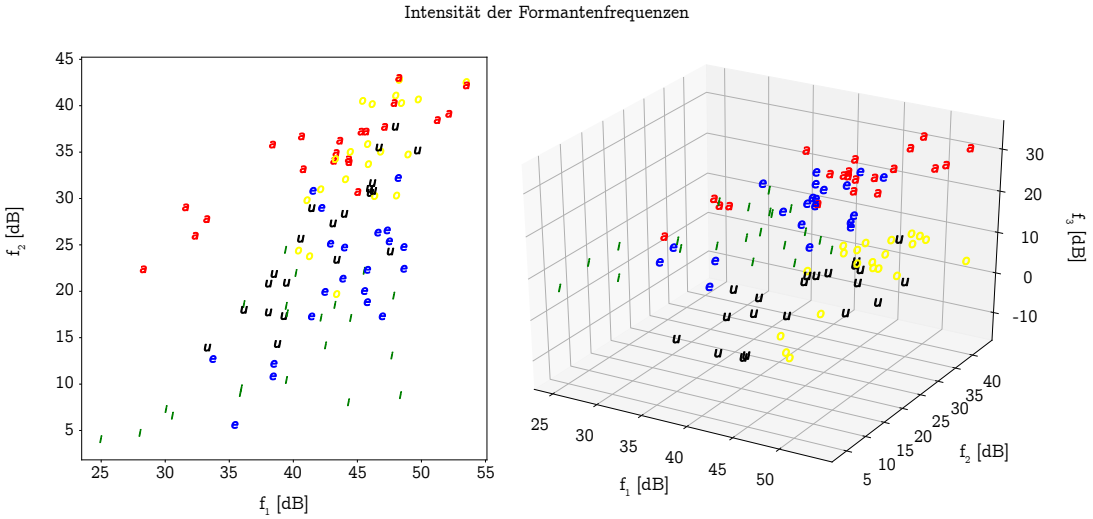



**Abbildung 4.10:** 2D- und 3D-Ansicht der durchschnittlichen Intensität der Formantenfrequenzen.

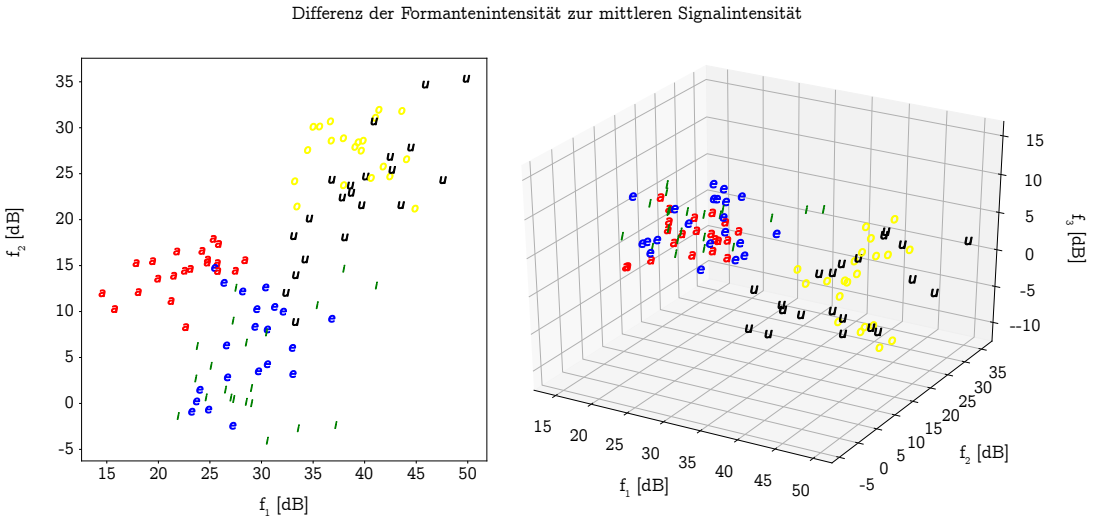



**Abbildung 4.11:** 2D- und 3D-Ansicht der Differenz der durchschnittlichen Formantenintensität zur durchschnittlichen Intensität des gesamten Vokals.

### 4.4.2 Differenzen der Formantenfrequenzen

Die ersten beiden Frequenzen der Formanten werden durch den Vokaltrakt gesteuert. Wie Brenner [3] beschreibt, reslutiert aus einer vorderen Zungenposition und somit einem kleinen Resonanzraum, ein hoher zweiter Formant, wie es beispielsweise bei „e“ und „i“ der Fall ist. Die Zungenhöhe wirkt sich dagegen auf den ersten Formanten aus. Je tiefer sie liegt, desto höher wird der erste Formant, was beispielsweise bei dem Vokal „a“ gut erkennbar ist.

In Abbildung 4.12 wurden die mittleren Frequenzwerte der ersten drei Formanten

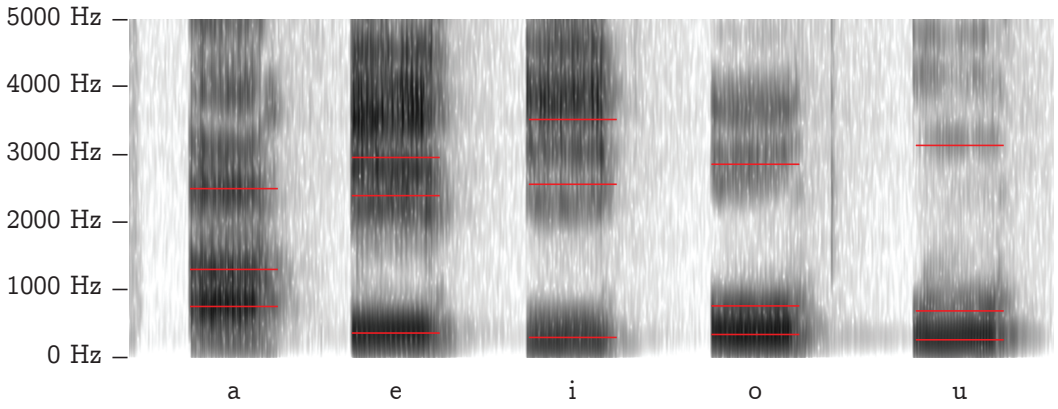



**Abbildung 4.12:** Spektrogramm einer Vokalreihe. Die roten Linien markieren jeweils den Mittelwert der ersten drei Formanten der Vokale.

einer Vokalreihe in Form einer roten Linie in das Spektrogramm eingezeichnet. Dabei lassen sich die Abhängigkeiten der Zungenposition gut nachvollziehen. Jeder Vokal hat in der Sprachproduktion seine eigene Zungenposition, weshalb die Differenzen der Formanten berechnet und als Eigenschaften aufgenommen wurden.

### 4.4.3 Resultierendes Datenkonstrukt

Durch die Aufbereitung und Berechnung der Eigenschaften der Vokale wurden schlussendlich über 3000 Werte generiert. Eine schematische Übersicht aller resultierenden Eigenschaften, in Form eines Datenkonstruktes im JSON-Format, zeigt der Code in Programm 4.1.

## 4.5 Wahl des Machine Learning Algorithmus

Nachdem die Eigenschaften aus dem Signal extrahiert wurden, mussten sie in ein Machine Learning Modell eingearbeitet werden. Ziel der Umsetzung war es, das für die vorhandenen Daten optimale Machine Learning Modell zu finden und zu erstellen. Ausschlaggebend für die Treffsicherheit des Modells ist der gewählte Algorithmus, der dem Modell zugrunde liegt. Um tatsächlich den besten Algorithmus zu ermitteln, wurde für jeden der in Abschnitt 3.4.7 vorgestellten Supervised Machine Learning Algorithmen ein Modell erstellt, um sie direkt miteinander vergleichen zu können.

Damit die gesammelten Daten für das Training und die Evaluierung der Algorithmen verwendet werden können, mussten sie in eine andere Form gebracht werden. Dabei wurden alle Eigenschaften eines gesprochenen Vokals auf eine Ebene gebracht und als aufeinander folgende Einträge in einem Array namens `X_train_raw` gespeichert. Jeder Vokal wird somit als eine, immer in der gleichen Reihenfolge auftretende, Reihe an Werten repräsentiert. Damit die Information, um welchen Vokal es sich bei den Eigenschaften handelt, nicht verloren geht, wurde ein zweites Array namens `Y_train_raw` erstellt, welches diese beinhaltet. Anhand der Indizes kann der Zusammenhang der bei-

**Programm 4.1:** JSON Schema des resultierenden Datenkonstrukts für einen Vokal.

```
'Sprecher': {
  'Vokal': {
    'Allgemein': {
      'Differenz_F1_F2': Wert,
      'Differenz_F1_F3': Wert,
      'Differenz_F2_F3': Wert,
      'Durchschnittliche_Intensitaet': Wert,
      'Grundfrequenz': Wert
    },
    'Formant1': {
      'Bandbreite': Wert,
      'Differenz_Grundfrequenz': Wert,
      'Formanten_Intensitaet': Wert,
      'Harmonisches_Mittel': Wert,
      'Mittelwert': Wert,
      'Standardabweichung': Wert,
      'Varianz': Wert,
      'Verhaeltnis_Formanten_Intensitaet': Wert
    },
    'Formant2': {
      'Bandbreite': Wert,
      'Differenz_Grundfrequenz': Wert,
      'Formanten_Intensitaet': Wert,
      'Harmonisches_Mittel': Wert,
      'Mittelwert': Wert,
      'Standardabweichung': Wert,
      'Varianz': Wert,
      'Verhaeltnis_Formanten_Intensitaet': Wert
    },
    'Formant3': {
      'Bandbreite': Wert,
      'Differenz_Grundfrequenz': Wert,
      'Formanten_Intensitaet': Wert,
      'Harmonisches_Mittel': Wert,
      'Mittelwert': Wert,
      'Standardabweichung': Wert,
      'Varianz': Wert,
      'Verhaeltnis_Formanten_Intensitaet': Wert
    }
}
```

den Arrays hergestellt werden.

Ein weiterer Schritt, der vor der Erstellung der Modelle notwendig war, ist die Aufteilung der vorhandenen Daten in einen Test- und einen Trainingsteil. Der folgende Code zeigt, wie dies standardmäßig mit `scikit-learn` gemacht werden kann:

```
X_train, X_test, Y_train, Y_test = train_test_split(X_train_raw, Y_train_raw,
random_state=0)
```

Dabei werden die Daten in einen Trainingsteil zu 75% und einen Testteil zu 25% aufgeteilt.

Darauf folgte die Erstellung der Modelle, deren Training anhand der erstellten Arrays

**Tabelle 4.1:** Treffsicherheiten der Machine Learning Modelle nach Training mit den Rohdaten.

| Algorithmus des Modells | Treffsicherheit |
|---|---|
| K-Nearest-Neighbors | 0,444 |
| Logistic Regression | 0,889 |
| Decision Tree | 0,852 |
| Naive Bayes Klassifikator | 0,778 |
| Support Vektor Maschine | 0,074 |
| Neuronales Netzwerk | 0,259 |

**Programm 4.2:** Standardisieren der Daten durch den `StandardScaler`.

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

und die Ermittlung ihrer Treffsicherheiten. Der folgende Code zeigt diesen Vorgang bei einem Modell auf Basis des *K-Nearest-Neighbors*-Algorithmus:

```
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
```

Die Tabelle 4.1 zeigt alle Treffsicherheiten der Modelle. Wie daraus hervorgeht, variieren diese sehr stark. Aufgrund der zahlreichen Möglichkeiten, die Treffsicherheit zu verbessern, wäre eine finale Wahl eines Algorithmus an dieser Stelle jedoch noch zu voreilig.

### 4.5.1 Feature Scaling

Manche Algorithmen reagieren auf das (in Abschnitt 3.4.6 beschriebene) *Feature Scaling* sehr empfindlich, weshalb es auch in der Umsetzung berücksichtigt wurde. Die Auswirkungen der Methoden auf die Modelle werden folgend aufgezeigt.

#### Standardisierung der Daten

In `scikit-learn` kann die Standardisierung der Daten unter Verwendung des `StandardScaler` vollzogen werden. Dabei werden alle Daten so neu ausgerichtet, dass sie einen gemeinsamen Mittelpunkt bei 0 und eine Varianz von 1 aufweisen. Die Durchführung der Skalierung wird in Programm 4.2 gezeigt. Die Treffsicherheiten der Modelle durch Training mit den standardisierten Daten werden in Tabelle 4.2 gezeigt. Auffällig ist dabei, dass sich die Treffsicherheit beinahe aller Modelle verbessert hat. Lediglich bei *Logistic Regression* ist die Treffsicherheit gesunken. Bestätigt wurde auch, dass (wie in Abschnitt 3.4.7 beschrieben) die Skalierung der Daten auf Modelle auf Basis des *Decision Tree*-Algorithmus keinen Einfluss hat.

**Tabelle 4.2:** Treffsicherheiten der Machine Learning Modelle nach Training mit den standardisierten Daten.

| Algorithmus des Modells | Treffsicherheit |
|---|---|
| K-Nearest-Neighbors | 0,704 |
| Logistic Regression | 0,852 |
| Decision Tree | 0,852 |
| Naive Bayes Klassifikator | 0,815 |
| Support Vektor Maschine | 0,778 |
| Neuronales Netzwerk | 0,889 |

**Tabelle 4.3:** Treffsicherheiten der Machine Learning Modelle nach Training mit den normalisierten Daten.

| Algorithmus des Modells | Treffsicherheit |
|---|---|
| K-Nearest-Neighbors | 0,630 |
| Logistic Regression | 0,852 |
| Decision Tree | 0,852 |
| Naive Bayes Klassifikator | 0,815 |
| Support Vektor Maschine | 0,481 |
| Neuronales Netzwerk | 0,852 |

#### Normalisierung der Daten

Ähnlich wie bei der Standardisierung können die Daten auch normalisiert werden. Dabei werden alle Daten in einen Wertebereich zwischen 0 und 1 skaliert. In `scikit-learn` kann dies mit dem `MinMaxScaler` umgesetzt werden. Die Treffsicherheiten der Modelle nach dem Trainieren mit den normalisierten Daten werden in Tabelle 4.3 aufgezeigt. Daraus geht hervor, dass die Treffsicherheit zwar auch bei beinahe allen Modellen besser wurde, jedoch nicht so deutlich wie bei den standardisierten Daten.

### 4.5.2 Feature Selection

Jeder Algorithmus verfolgt einen eigenen Ansatz, um ein Ergebnis vorhersagen zu können. Die einzelnen Eigenschaften sind deswegen auch für jedes Modell unterschiedlich wichtig, weshalb es sinnvoll erscheint, sich auf die für das Modell aussagekräftigen Eigenschaften zu konzentrieren. Genau dieses Ziel verfolgen die diversen Ansätze der Feature Selection. Abbildung 4.13 zeigt beispielsweise die jeweilige Wichtigkeit der unvorverarbeiteten Eigenschaften für ein Modell auf Basis des *Decision Tree*-Algorithmus. Alle *Feature Selection* Methoden wurden jeweils auf die Trainingsdaten angewandt, bevor die Modelle damit trainiert wurden. Die Effekte auf die Treffsicherheit können in Tabelle 4.4 verglichen werden. Da das Modell auf Basis der *Support Vektor Maschine* (wie in Abschnitt 3.4.7 beschrieben) ohnehin nur die wichtigsten Merkmale als sogenannte Support Vektoren verwendet, bringt in diesem Fall die *Feature Selection* keinen Vorteil mit sich. Bei den meisten Modellen hingegen sind durchaus Verbesserungen zu beobachten.

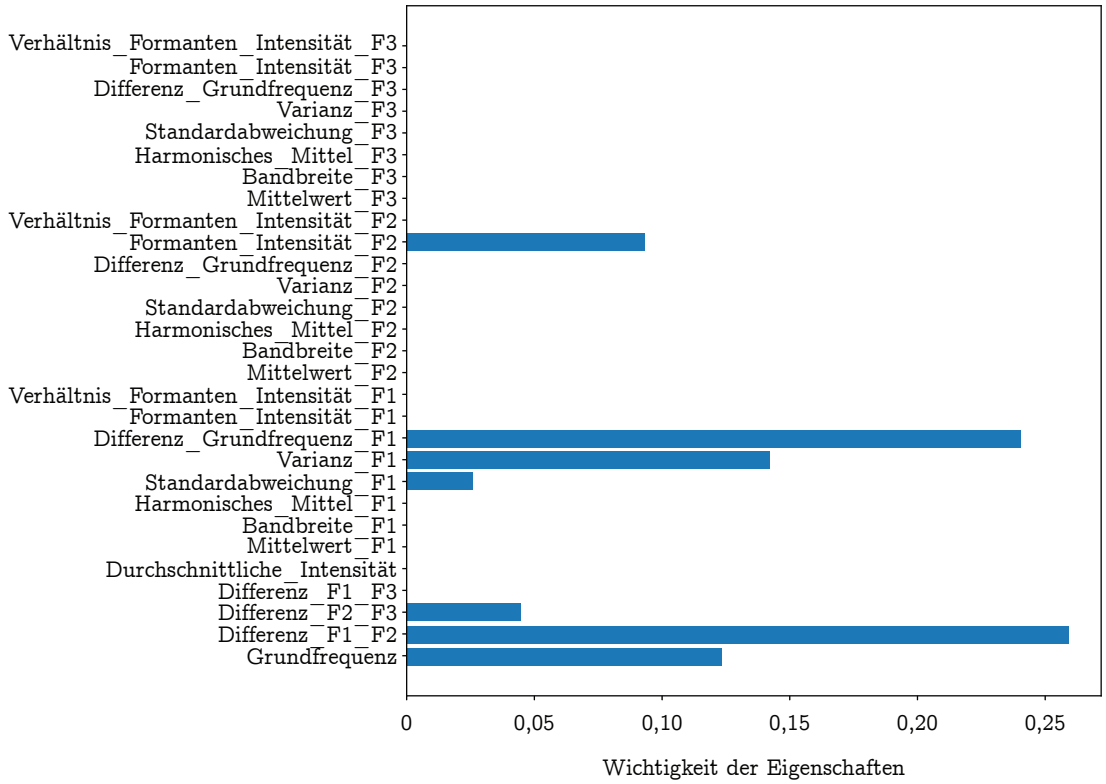



**Abbildung 4.13:** Wichtigkeit der unvorverarbeiteten Eigenschaften eines Modells basierend auf dem *Decision Tree*-Algorithmus.

**Tabelle 4.4:** Liste der Treffsicherheiten der Machine Learning Modelle nach Training mit den Daten ohne *Feature Selection* (OFS), mit *Univariate Statistics* (US), mit *Model Based Selection* (MBS) und mit *Iterative Selection* (IS).

| | Treffsicherheit | | | |
|---|---|---|---|---|
| Algorithmus des Modells | OFS | US | MBS | IS |
| K-Nearest-Neighbors | 0,444 | 0,741 | 0,704 | 0,704 |
| Logistic Regression | 0,889 | 0,906 | 0,778 | 0,906 |
| Decision Tree | 0,852 | 0,889 | 0,815 | 0,889 |
| Naive Bayes Klassifikator | 0,778 | 0,815 | 0,815 | 0,852 |
| Support Vektor Maschine | 0,074 | 0,074 | 0,074 | 0,074 |
| Neuronales Netzwerk | 0,259 | 0,037 | 0,259 | 0,407 |

### 4.5.3 Modell-Evaluierung

Nachdem die Vorverarbeitung der Daten umgesetzt wurde, war es wichtig, eine möglichst richtige Einschätzung der Treffsicherheit anhand der Testdaten zu erlangen und jene Parameter herauszufinden, durch die die jeweiligen Algorithmen das beste Ergebnis liefern. Diese Vorgänge fallen unter anderem in den Bereich der Modell-Evaluierung

**Programm 4.3:** Durchführung einer *Cross Validation* an einem Modell auf Basis des *K-Nearest-Neighbors*-Algorithmus. Die Ausgabe der Zwischenresultate und der resultierenden Durchschnittstreffsicherheit wird darunter dargestellt.

```
model = KNeighborsClassifier()
scores = cross_val_score(model, X_train_raw, Y_train_raw, cv=10)
print("Treffsicherheiten: {}".format(scores))
print("Durchschnittliche Treffsicherheit: {:.2f}".format(scores.mean()))
```

```
Treffsicherheiten: [0.3 0.5 0.4 0.6 0.5 0.4 0.6 0.6 0.8 0.5]
Durchschnittliche Treffsicherheit: 0.52
```

**Tabelle 4.5:** Treffsicherheiten der Machine Learning Modelle ohne und mit Einsatz von *Cross Validation*.

| | Treffsicherheit | |
|---|---|---|
| Algorithmus des Modells | ohne CV | mit CV |
| K-Nearest-Neighbors | 0,444 | 0,523 |
| Logistic Regression | 0,889 | 0,763 |
| Decision Tree | 0,852 | 0,810 |
| Naive Bayes Klassifikator | 0,778 | 0,800 |
| Support Vektor Maschine | 0,074 | 0,200 |
| Neuronales Netzwerk | 0,259 | 0,260 |

und wurden wie folgt durchgeführt.

### Cross Validation

Um eine authentischere Treffsicherheit der Modelle zu erhalten, wurde die (in Abschnitt 3.4.8 erklärte) emphCross Validation eingesetzt. Um beispielsweise 90% der Trainingsdaten zur Erlangung der Treffsicherheit nutzen zu können, müssen diese in Zehnerblöcke unterteilt werden. Dies geschieht in `scikit-learn` durch Übergabe des Modells, der Trainings- und Testdaten sowie der Anzahl der *Cross Validation* Schritte an die `cross_val_score` Methode. Der Code in Programm 4.3 zeigt die Ermittlung der Untertreffsicherheiten sowie die gemittelte Gesamttreffsicherheit der *Cross Validation* eines Modells auf Basis des *K-Nearest-Neighbors*-Algorithmus bei einer Aufteilung der Daten in zehn Blöcke. In Tabelle 4.5 können die Treffsicherheiten aller Modelle mit und ohne *Cross Validation* verglichen werden.

### Grid Search

Die *Grid Search* hat (wie in Abschnitt 3.4.8 erklärt) das Finden der besten Parameter für den dem Modell zugrunde liegenden Algorithmus zum Ziel. Damit das gemessene Ergebnis möglichst aussagekräftig ist, wurde auch die *Cross Validation* in die *Grid Search* inkludiert. In `scikit-learn` kann eine *Grid Search*, wie im Code in Programm 4.4 gezeigt, durchgeführt werden. Alle zu testenden Parameter und Parameterwerte

**Programm 4.4:** Durchführung einer *Grid Search* mit inkludierter *Cross Validation* an einem Modell auf Basis des *Support Vektor Maschine*-Algorithmus.

```
param_grid = {'C': [0.1, 1, 10, 100, 1000, 10000, 100000], 'gamma': [0.00001,
    0.0001, 0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=10)
grid.fit(X_train_scaled, Y_train)
```


**Abbildung 4.14:** *Heatmap* der Ergebnisse einer *Grid Search*.

werden dabei in der Variable `param_grid` gespeichert und der Methode `GridSearchCV` als Parameter übergeben. Die Ergebnisse dieser Evaluierung können gut anhand einer *Heatmap* dargestellt werden, was für das im Code in Programm 4.4 beschriebene Beispiel in Abbildung 4.14 veranschaulicht wird. Daraus geht hervor, dass für diesen Algorithmus ein *Gamma*-Wert von 0,01 in Kombination mit einem *C*-Wert von 10 das beste Ergebnis liefert. Alle in der *Heatmap* angezeigten Ergebnisse sind dabei Durchschnittswerte, die jeweils eine *Cross Validation* hinter sich haben.

### Confusion Matrix und Classification Report

Nachdem die optimalen Parameter für die Modelle eruiert wurden, sollte ein tieferer Einblick in den Klassifizierungsvorgang der Modelle gegeben werden. Welche Klassifi-

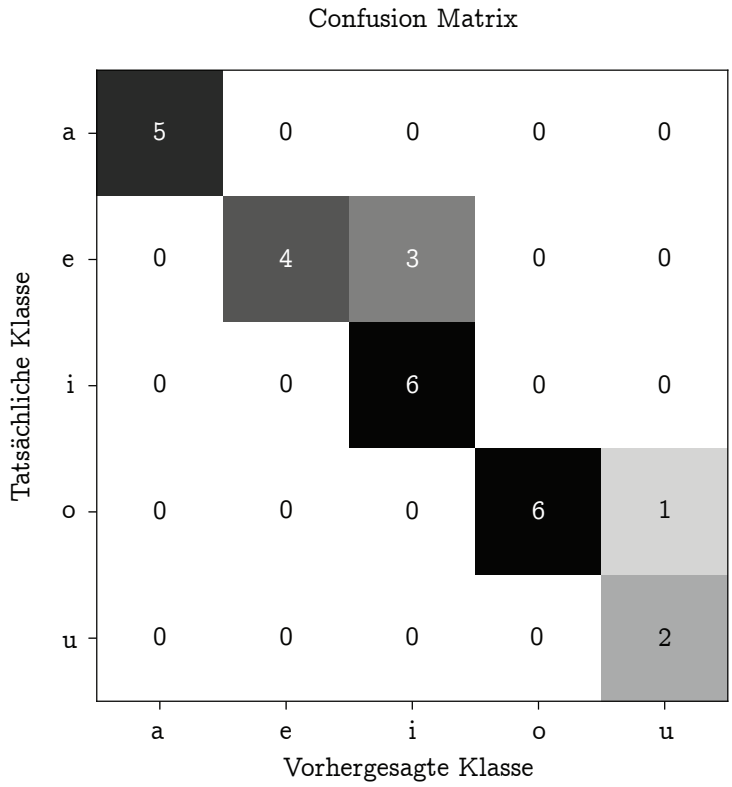



**Abbildung 4.15:** *Confusion Matrix* eines Modells auf Basis einer *Support Vektor Maschine*.

**Tabelle 4.6:** *Classification Report* eines Modells auf Basis einer *Support Vektor Maschine*.

| | Präzision | Recall | f1-score | Anzahl |
|---|---|---|---|---|
| a | 1,00 | 1,00 | 1,00 | 5 |
| e | 1,00 | 0,57 | 0,73 | 7 |
| i | 0,67 | 1,00 | 0,80 | 6 |
| o | 1,00 | 0,86 | 0,92 | 7 |
| u | 0,67 | 1,00 | 0,80 | 2 |
| avg / total | 0,90 | 0,85 | 0,85 | 27 |

zierungen welcher Vokale aus den Testdaten einem Modell Probleme bereiteten, wurde anhand der *Confusion Matrix* und des *Classification Report* ermittelt. Die Abbildung 4.15 zeigt die *Confusion Matrix* jenes Modells auf Basis eines *Support Vektor Maschine*-Algorithmus, deren Parameter zuvor durch die *Grid Search* ermittelt wurden. Daraus geht hervor, dass nur der Vokal „a“ ohne Probleme klassifiziert werden konnte. Auch der (in Abschnitt 3.4.8 beschriebene) *Classification Report* wurde erstellt, um die Problembereiche herauszufinden. Dieser kann in Tabelle 4.6 betrachtet werden. Dabei wird ein Unterscheidungsproblem zwischen „e“ und „i“ sowie zwischen „o“ und „u“ deutlich.

**Tabelle 4.7:** Treffsicherheiten aller Algorithmen mit unterschiedlich vorverarbeiteten Daten. Die jeweils optimalen Parameter wurden durch eine *Grid Search* und das Ergebnis durch eine *Cross Validation* ermittelt.

| Algorithmus des Modells | Ohne | SS | NO | US | MBS | IS | SS+US | SS+MBS | SS+IS | NO+US | NO+MBS | NO+IS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K-Nearest-Neighbors | 0,60 | 0,80 | 0,84 | 0,74 | 0,74 | 0,75 | 0,88 | 0,90 | 0,89 | 0,90 | 0,90 | 0,89 |
| Logistic Regression | 0,81 | 0,90 | 0,88 | 0,91 | 0,91 | 0,90 | 0,92 | 0,91 | 0,89 | 0,92 | 0,91 | 0,89 |
| Decision Tree | 0,88 | 0,88 | 0,88 | 0,85 | 0,89 | 0,85 | 0,85 | 0,89 | 0,88 | 0,85 | 0,89 | 0,88 |
| Naive Bayes Klassifikator | 0,76 | 0,85 | 0,76 | 0,78 | 0,81 | 0,76 | 0,78 | 0,81 | 0,76 | 0,78 | 0,81 | 0,76 |
| Support Vektor Maschine | 0,84 | 0,91 | 0,92 | 0,90 | 0,90 | 0,90 | 0,91 | 0,92 | 0,91 | 0,92 | 0,90 | 0,90 |
| Neuronales Netzwerk | 0,68 | 0,91 | 0,91 | 0,90 | 0,91 | 0,90 | 0,92 | 0,92 | 0,92 | 0,91 | 0,91 | 0,91 |

### 4.5.4 Finale Auswahl des Algorithmus

In einem sehr rechenaufwändigen Prozess wurde die *Grid Search* für alle Algorithmen mit den unterschiedlich vorverarbeiteten Daten durchgeführt. Die Daten wurden dabei durch alle möglichen *Feature Scaling* und *Feature Selection* Kombinationen mehrfach separat aufbereitet, um damit jeweils ein Modell zu trainieren. Für jedes Modell mit jeder Datenkombination wurden anschließend jene Parameter ermittelt, durch die das Modell die beste Treffsicherheit erlangt. Um ein genaueres Ergebnis zu liefern, wurden diese Treffsicherheiten durch eine *Cross Validation* ermittelt. Eine Auflistung aller erhaltenen Werte ist in Tabelle 4.7 auf Seite 48 zu sehen. Aus diesem aussagekräftigen Vergleich geht schlussendlich hervor, dass Modelle auf Basis von *Support Vektor Maschinen* mit dem geringsten Vorverarbeitungsaufwand die beste Treffsicherheit für die vorhandenen Daten liefern.

# Kapitel 5

# Resultat

Auf Basis der im Kapitel 4 gewonnenen Erkenntnisse wurde schlussendlich das Vokalerkennungssystem erstellt. Als finaler Algorithmus wurde der in `scikit-learn` implementierte `SVC` gewählt, der zu den *Support Vektor Maschinen* zählt. Das Modell wurde mit allen durch den `MinMaxScaler` normalisierten Daten trainiert und die Treffsicherheit mit Hilfe einer *Cross Validation* ermittelt. Um das System auch mit neuen, unbekannten Daten testen zu können, wurde eine Vokalreihe gesondert aufgenommen und gleich den Trainingsdaten verarbeitet. Die daraus extrahierten Eigenschaften wurden in ein Array namens `X_predict_raw` gespeichert. Anhand dieser Daten wurde die Vorhersage dann durchgeführt. Der gesamte Ablauf und das Ergebnis werden in Programm 5.1 dargestellt.

**Programm 5.1:** Erstellen, Trainieren und Testen des finalen Machine Learning Modells. Die Ausgabe wird darunter dargestellt.

```
# Skalieren der Trainings und Testdaten
scaler = MinMaxScaler()
scaler.fit(X_train_raw)
X_train_scaled = scaler.transform(X_train_raw)
X_predict_scaled = scaler.transform(X_predict_raw)

# Erstellen des Modells mit den optimalen Parametern
model = SVC(C=1000, gamma=0.1)
# Trainieren des Modells mit allen vorhandenen Trainingsdaten
model.fit(X_train_scaled, Y_train_raw)

# Ermittlung der Treffsicherheit des Modells durch die Cross Validation
scores = cross_val_score(model, X_train_scaled, Y_train_raw, cv=5)

# Ausgabe der Treffsicherheit
print("Cross Validation Treffsicherheit: {:.3f}".format(scores.mean()))
# Ausgabe des vorhergesagten Resultats
print("Vorhersage der Test-Vokalreihe: {}".format(model.predict(X_predict_scaled)))
```

```
Cross Validation Treffsicherheit: 0.922
Vorhersage der Test-Vokalreihe: ['a' 'e' 'i' 'u' 'u']
```

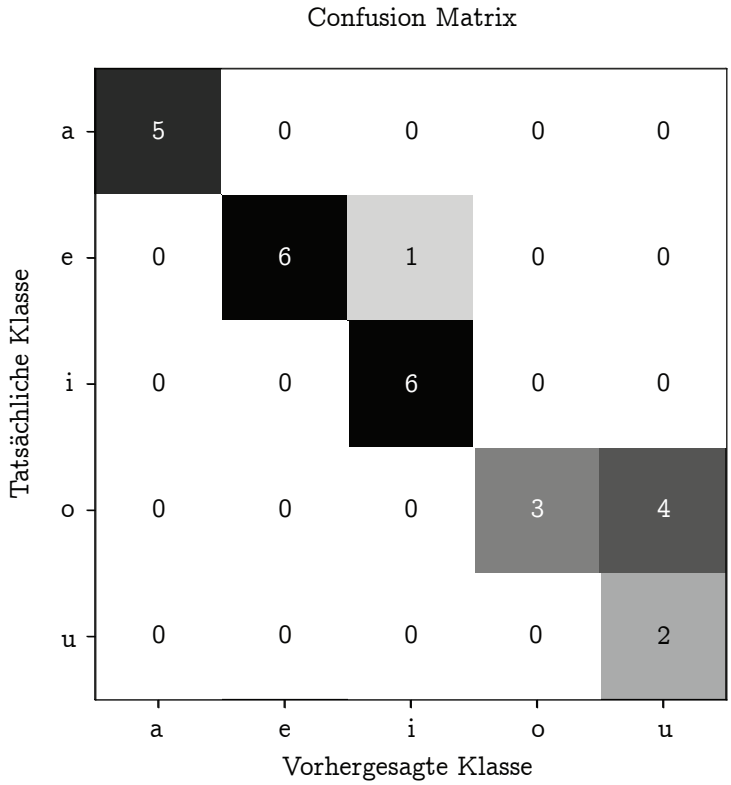



**Abbildung 5.1:** *Confusion Matrix* der *Grid Search* des finalen Machine Learning Modells.

**Tabelle 5.1:** *Classification Report* des finalen Machine Learning Modells.

| | Präzision | Recall | f1-score | Anzahl |
|---|---|---|---|---|
| a | 1,00 | 1,00 | 1,00 | 5 |
| e | 1,00 | 0,86 | 0,92 | 7 |
| i | 0,86 | 1,00 | 0,92 | 6 |
| o | 1,00 | 0,43 | 0,60 | 7 |
| u | 0,33 | 1,00 | 0,50 | 2 |
| avg / total | 0,92 | 0,81 | 0,82 | 27 |

Die Treffsicherheit des erstellten Modells lag am Ende bei 92%. Zum Testen des Modells wurden Vokale in der Reihenfolge „a“, „e“, „i“, „o“ und „u“ aufgenommen. Das System war somit in der Lage alle Vokale, bis auf einen, richtig zu erkennen, womit das grundlegende Ziel, alle Vokale nur anhand ihrer spektralen Eigenschaften zu erkennen, nicht ganz erreicht wurde. Um das finale Modell noch etwas besser evaluieren und die Problemstellen herausfinden zu können, wurden die vorhandenen Daten aufgeteilt und 25% davon verwendet, um eine *Confusion Matrix* und einen *Classification Report* zu erstellen, welche auf Abbildung 5.1 bzw. in der Tabelle 5.1 zu sehen sind.

Wie anhand der *Confiusion Matrix* zu erkennen ist, liegen die Probleme bei der Klassifizierung zwischen den Vokalen „e“ und „i“, sowie zwischen „o“ und „u“. Somit hat sich der durch Betrachtung der 2D- und 3D-Diagramme der Eigenschaften entstandene Verdacht, dass genau diese Klassen schwer voneinander unterscheidbar sind, bestätigt. Der *Classification Report* gibt ebenfalls tiefe Einblicke in die getroffenen Entscheidungen des Modells und legt dar, dass alle Vokale „a“, „e“ und „o“, die als solche klassifiziert wurden, auch tatsächlich dieser Klasse angehörten. Falsch klassifiziert wurden ein „i“, das eigentlich ein „e“ und vier „u“, die richtigerweise ein „o“ sein hätten müssen. Dies macht deutlich, dass das Hauptproblem des Modells die Unterscheidung zwischen „o“ und „u“ ist und die Unterscheidung zwischen „e“ und „i“ ebenfalls nicht eindeutig möglich ist.

# Kapitel 6

# Schlussbemerkung

## 6.1 Zusammenfassung

Im Zuge dieser Arbeit wurde eine Vorgangsweise zur Erlangung von spektralen Eigenschaften von gesprochenen Vokalen präsentiert, die diese möglichst gut charakterisieren. Die dominanten Frequenzen, auch Formanten genannt, waren dabei im Fokus und Ausgangspunkt für die Berechnung von weiteren Eigenschaften. Auch die Differenzen der Formantenfrequenzen erwiesen sich als vielversprechende Eigenschaft. Damit die Daten vorweg grob eingeschätzt werden konnten, wurden sie in Form von 2D- bzw. 3D-Diagrammen dargestellt. Um von den extrahierten Eigenschaften auf einen Vokal schließen zu können, wurden viele verschiedene Machine Learning Ansätze durchexerziert. Auffällig war, dass eine sorgfältige Datenvorverarbeitung bei den meisten eingesetzten Algorithmen große Verbesserungen mit sich brachte. Auch die Auswahl der Parameter, die das Verhalten der Algorithmen steuern, war ein wichtiger Punkt. Dank der *GridSearch* Methode konnte die Ermittlung der optimalen Parameter in einem sehr rechenintensiven Vorgang für alle Algorithmen automatisiert durchgeführt werden. Aus dieser Evaluierung der Algorithmen ging hervor, dass bei den vorhandenen Trainingsdaten ein Modell auf Basis eines *Support Vektor Maschinen*-Algorithmus (mit 92%) die beste Treffsicherheit mit dem wenigsten Vorverarbeitungsaufwand liefert. Bei genauerer Betrachtung konnte festgestellt werden, dass der Vokal „a“ immer sehr eindeutig klassifiziert werden konnte, während die Unterscheidung zwischen den Vokalen „e“ und „i“ sowie speziell zwischen „o“ und „u“ dem Modell Probleme bereitet.

## 6.2 Ausblick

Der Bereich der Spracherkennung wird bereits lange intensiv erforscht. Auch wenn die Hardwarekomponenten immer kleiner und leistungsstärker werden, steigt die Individualität des Einsatzgebietes stetig an, weshalb die benötigte Leistung einer Spracherkennung stets ein wichtiger Aspekt bleiben wird. Auch wenn das Ergebnis dieser Arbeit für einen Einsatz in einem produktiven System noch nicht ganz geeignet ist, zeigt es dennoch auf, dass Vokale anhand ihrer spektralen Eigenschaften prinzipiell voneinander unterschieden werden können.

Um das entstandene System zu verbessern, bzw. zu erweitern, wäre eine Aufstockung

der Trainingsdaten ein interessanter Ansatz. Damit könnte das Modell noch besser trainiert werden, was vermutlich zu einer Steigerung der Treffsicherheit führen würde. Auch im Bereich der Signalverarbeitung könnten noch weitere spektrale Eigenschaften extrahiert und mit den vorhandenen experimentiert werden. Eine Erweiterung des Systems von den Hauptvokalen auf alle Vokale wäre ebenfalls ein spannender Schritt. Dies wäre ohne großen Aufwand bereits möglich, da alle bei der Extraktion der Merkmale auftretenden Kennzeichnungen den Klassen hinzugefügt werden. Somit müsste lediglich mit der Digitalisierung und entsprechenden Kennzeichnung begonnen werden.

# Anhang A

# Installationsanleitung der Umsetzung

Installationsanleitung aller benötigten Komponenten zur Ausführung des Projekts unter *macOS*[1].

## A.1 Praat

Zur Extraktion der Eigenschaften eines Sprachsignals wurden *Praat*-Skripte erstellt. Um diese ausführen zu können, muss das Audioanalyseprogramm *Praat* installiert sein. Dieses ist frei verfügbar und kann auf der offiziellen Website[2] heruntergeladen werden. Die Skripte müssen in der nummerierten Reihenfolge ausgeführt werden. Die Pfade zu den zu analysierenden Audiodateien können in einer Dialogbox bei der Ausführung der Skripte gewählt werden. Resultierend aus der Ausführung aller Skripte, wird eine Datei namens `calculated.json` erzeugt, die alle extrahierten Eigenschaften enthält.

## A.2 Python

Die Verarbeitung der Daten sowie die Machine Learning Vorgänge wurden in *Python* realisiert. Um diese ausführen zu können, muss *Python*[3] mindestens in der Version 3.6.1 installiert sein. Zusätzlich, werden die *Python*-Bibliotheken `scikit-learn`, `matplotlib`, `statistics` und `json` benötigt. Um die *Python*-Programme ausführen zu können, muss der `load_from_file` Funktion der Pfad der zuvor erstellten `calculated.json` übergeben werden. Das finale Machine Learning Modell kann durch Ausführen der Datei `05_final_pipeline.py` erstellt werden.

[1] https://www.apple.com/lae/macos/sierra/
[2] http://www.fon.hum.uva.nl/praat/download_mac.html
[3] https://www.python.org/downloads/

# Anhang B

# Inhalt der CD-ROM

## B.1 PDF-Dateien

Pfad: /

Ennser_Christoph_2017.pdf Masterarbeit

Installationsanleitung.pdf Installationsanleitung für die Ausführung der Umsetzung

Pfad: /Onlinequellen/

AppleReportQ42016.pdf Apple Reports 4Q 2016 Results [25]

## B.2 Projektdaten

Pfad: /Projekt/

Praat/ . . . . . Skripte und Dateien der Signalverarbeitung

Python/ . . . . . Python Dateien für die Erstellung und Evaluierung der Machine Learning Modelle

## B.3 Abbildungen

Pfad: /Abbildungen/

*.pdf . . . . . Vektorgrafiken

*.png . . . . . Screenshots

# Quellenverzeichnis

## Literatur


[1] Sylvain Arlot, Alain Celisse u. a. „A Survey of Cross-Validation Procedures for Model Selection“. *Statistics Surveys* 4 (2010), S. 40–79 (siehe S. 26).

[2] Paul Boersma und Vincent Van Heuven. „Speak and unSpeak with PRAAT“. *Glot International* 5.9/10 (2001), S. 341–347 (siehe S. 30).

[3] Koloman Brenner, Balázs Huszka und Csaba Werk-Marinkás. *Deutsche Phonetik*. Budapest, Hungary: Bölcsész Konzorcium, 2006 (siehe S. 12, 14, 39).

[4] Pedro Domingos und Michael Pazzani. „On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss“. *Machine Learning* 29.2 (1997), S. 103–130 (siehe S. 23).

[5] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz. Eine Praxisorientierte Einführung*. 4. Aufl. Wiesbaden, DE: Springer, 2016 (siehe S. 20, 21, 24).

[6] Klaus Fellbaum. *Sprachverarbeitung und Sprachübertragung*. 2. Aufl. Berlin Heidelberg, DE: Springer, 2013 (siehe S. 11–13).

[7] Peter Harrington. *Machine Learning in Action*. Bd. 5. Manning Greenwich, CT, 2012 (siehe S. 31).

[8] Hynek Hermansky. „Perceptual Linear Predictive (PLP) Analysis of Speech“. *The Journal of the Acoustical Society of America* 87.4 (1990), S. 1738–1752 (siehe S. 5).

[9] Xuedong Huang, Alex Acero und Hsiao-Wuen Hon. *Spoken Language Processing. A Guide to Theory, Algorithm, and System Development*. Upper Saddle River, NJ, USA: Prentice Hall, 2001 (siehe S. 3, 9, 13, 14).

[10] Veton Z. Këpuska und Hussien A. Elharati. „Robust Speech Recognition System Using Conventional and Hybrid Features of MFCC, LPCC, PLP, RASTA-PLP and Hidden Markov Model Classifier in Noisy Conditions“. *Journal of Computer and Communications* 3.06 (2015) (siehe S. 5, 6).

[11] S. B. Kotsiantis. „Supervised Machine Learning. A Review of Classification Techniques“. In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Amsterdam, The Netherlands: IOS Press, 2007, S. 249–268 (siehe S. 23).

[12] Jeet Kumar, Om Prakash Prabhakar und Navneet Kumar Sahu. „Comparative Analysis of Different Feature Extraction and Classifier Techniques for Speaker Identification Systems. A Review“. *International Journal of Innovative Research in Computer and Communication Engineering* 2.1 (2014), S. 2760–2269 (siehe S. 5, 33).

[13] Paul Lamere u. a. „The CMU SPHINX-4 Speech Recognition System“. In: *Proceedings of the IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. Bd. 1. Hong Kong, China, 2003, S. 2–5 (siehe S. 6).

[14] Beth Logan u. a. „Mel Frequency Cepstral Coefficients for Music Modeling“. In: *Proceedings of the International Symposium on Music Information Retrieval*. Plymouth, Mass., US, 2000, S. 3. URL: http://ismir2000.ismir.net/papers/logan_paper.pdf (siehe S. 4).

[15] Petr Motlicek. *Feature Extraction in Speech Coding and Recognition*. Techn. Ber. Portland, US: Oregon Graduate Institute of Science und Technology, ASP Group, OGI-OHSU, 2002. URL: http://www.fit.vutbr.cz/research/view_pub.php.cs?id=7069 (siehe S. 4).

[16] Andreas Müller und Sarah Guido. *Introduction to Machine Learning with Python. A Guide for Data Scientists*. Sebastopol, CA, USA: O'Reilly Media, 2017 (siehe S. 15–19, 22–28, 30, 31).

[17] Fabian Pedregosa u. a. „Scikit-learn. Machine Learning in Python“. *Journal of Machine Learning Research* 12 (2011), S. 2825–2830 (siehe S. 31).

[18] Ralf Pörings und Ulrich Schmitz. *Sprache und Sprachwissenschaft. Eine Kognitiv Orientierte Einführung*. Tübingen, DE: Gunter Narr Verlag, 2003 (siehe S. 15).

[19] Lawrence R. Rabiner und Ronald W. Schafer. *Digital Processing of Speech Signals*. Englewood Cliffs, New Jersey, US: Prentice Hall, 1978 (siehe S. 4).

[20] Milan Ramljak, Maja Stella und Matko Šarić. „High Performance Processing for Speech Recognition“. *International Journal of Circuits, Systems and Signal Processing* 8 (2014), S. 166–172 (siehe S. 4).

[21] Sebastian Raschka. *Python Machine Learning*. Birmingham, United Kingdom: Packt Publishing, 2015 (siehe S. 16, 19).

[22] Guido Rossum. *Python Library Reference*. Techn. Ber. Amsterdam, The Netherlands: Centre for Mathematics und Computer Science (CWI), University of Amsterdam, 1995 (siehe S. 32).

[23] Robert J. Summers, Peter J. Bailey und Brian Roberts. „Effects of the Rate of Formant-Frequency Variation on the Grouping of Formants in Speech Perception“. *Journal of the Association for Research in Otolaryngology* 13.2 (2012), S. 269–280 (siehe S. 34).

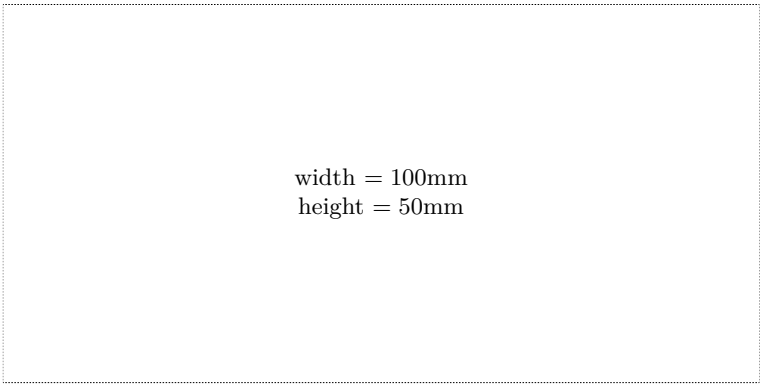
[24] Sergios Theodoridis und Konstantinos Koutroumbas. *Pattern Recognition*. 3. Aufl. Orlando, FL, USA: Academic Press, 2006 (siehe S. 21).

## Online-Quellen

[25] Joe Rossignol. *Apple Reports 4Q 2016 Results*. Juni 2017. URL: https://www.macrumors.com/2016/10/25/earnings-4q-2016/ (besucht am 04. 06. 2017) (siehe S. 7, 55).

# Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —




— Diese Seite nach dem Druck entfernen! —