BOSCH
Technik fürs Leben

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Analyse von Audiosignalen unter der Verwendung von Linear Predictive Coding

Projektarbeit T3000

über die Praxisphase des dritten Studienjahrs

an der Fakultät für Technik
im Studiengang Informationstechnik

an der DHBW Ravensburg
Campus Friedrichshafen

von
Henry Schuler

20. März 2023

Bearbeitungszeitraum:	02.01.2023 - 17.04.2023
Matrikelnummer, Kurs:	5220542, TIT20
Dualer Partner:	Robert Bosch GmbH
Betreuer des Dualen Partners:	Maximilian Main

Selbstständigkeitserklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017.
Ich versichere hiermit, dass ich meine Bachelorarbeit (bzw. Projektarbeit oder Studienarbeit bzw. Hausarbeit) mit dem Thema:

Analyse von Audiosignalen unter der Verwendung von Linear Predictive Coding

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Blaichach, 20. März 2023	TEF-EAT31,
Ort, Datum	Abteilung, Unterschrift

Zusammenfassung

Um Sprecher anhand von Stimmaufzeichnungen in einem System zu authentifizieren, muss das analoge Audiosignal in digitale Parameter umgewandelt werden, die einen Bezug zu der sprechenden Person ermöglichen. Diese Arbeit beschäftigt sich mit der Berechnung von Linear Predictive Coding Koeffizienten, welche Eigenschaften der Stimmerzeugung im Vokaltrakt modellieren. Der Zusammenhang zwischen Sprecher und berechneten Koeffizienten wird unter der Verwendung eines Neuronalen Netzes überprüft. In der Auswertung mit einem kleinen Datensatz von 10 Personen zeigt sich eine Vorhersagegenauigkeit von 70,54 Prozent, wodurch der grundsätzliche Zusammenhang gezeigt ist.

Schlüsselwörter Linear Predictive Coding, Sprecherauthentifizierung, Framing, Fensterfunktion

Abstract

To authenticate speakers via recordings of their voices, the analog audio recording has to be converted into digital parameters related to the speaker. For this reason, this study deals with calculating linear predictive coding coefficients, which are used to model human voice production in the vocal tract. The connection between speakers and calculated coefficients is checked using a neural network. Using a small data set consisting of ten different speakers, a prediction accuracy of 70.54 percent is achieved. Therefore the connection between the speaker and the coefficients is proven.

Keywords Linear Predictive Coding, Speaker Authentication, Framing, Windowing

Inhaltsverzeichnis

Abkürzungsverzeichnis

DHBW Duale Hochschule Baden-Württemberg 1

LPC Linear Predicitve Coding 1

LPCC Linear Prediction Cepstral Coefficient 8

MFCC Mel-frequency Cepstral Coefficients 1

AR Autoregression 4

NN Neuronales Netz 1

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Im Rahmen des Informatikstudiums an der Dualen Hochschule Baden-Württemberg (DHBW) Ravensburg muss im dritten Studienjahr eine Studienarbeit abgelegt werden. Die Hochschule stellt dafür eine Auswahl an Themen zur Verfügung. Eines dieser Themen beschäftigt sich mit der Problematik der Sprecherauthentifizierung, wobei es Nutzern ermöglicht werden soll, sich über ihre Stimme zu authentifizieren. In dieser Arbeit sollen die Grundlagen für die Bearbeitung dieser Studienarbeit behandelt werden.

1.1 Kontext

Damit ein Zusammenhang zwischen Stimme und Audioaufzeichnung hergestellt werden kann, müssen stimmspezifische Merkmale aus dem aufgezeichneten Stimmsignal extrahiert werden. Im Bereich der Sprecherauthentifizierung haben sich zwei Verfahren zur Berechnung stimmspezifischer Merkmale etabliert: Mel-frequency Cepstral Coefficients (MFCC) und Linear Predicitve Coding (LPC) (vgl. Zulfiqar u. a. 2010, S. 116) (vgl. Chelali und Djeradi 2017, S. 726). Während mittels des MFCC-Verfahrens versucht wird, die Funktionsweise des menschlichen Ohrs abzubilden, versucht das LPC-Verfahren die Eigenschaften des menschlichen Vokaltrakts aus dem Audiosignal zu extrahieren (vgl. Zulfiqar u. a. 2010, S. 117). Die erhaltenen Werte können anschließend für das Training eines Neuronalen Netzes (NN) verwendet werden, welches die Klassifizierung neuer Datensätze während des Authentifizierungsprozesses übernimmt.

1.2 Ziel der Arbeit

Im Rahmen dieser Arbeit soll das LPC-Verfahren genauer untersucht werden. Dazu soll ein Programm erstellt werden, welches ein gegebenes Audiosignal mittels LPC in eine vordefinierte Anzahl an Koeffizienten umwandelt. In einem weiteren Schritt soll der Zusammenhang zwischen den berechneten Koeffizienten und der sprechenden Person unter Verwendung eines vereinfachten NN aufgezeigt werden.

1.3 Vorgehensweise

Die Arbeit unterteilt sich in fünf Kapitel. Im Anschluss an die Einleitung stellt Kapitel 2 die für diese Arbeit relevanten Grundlagen vor. Kapitel 3 kombiniert die vorgestellten Verfahren zu einem ausführbaren Programm. Die Ergebnisse des erstellten Programms werden in Kapitel 4 validiert. Abschließend werden die Erkenntnisse in Kapitel 5 interpretiert und die Arbeit wird mit einem Ausblick abgeschlossen.

2 Grundlagen

Der Schwerpunkt dieser Arbeit unterteilt sich in zwei Teile, die Signalvorverarbeitung und das LPC-Verfahren. Im Folgenden werden die theoretischen Grundlagen für beide Prozesse beschrieben.

2.1 Signalvorverarbeitung

Um ein gegebenes Audiosignal einheitlich verarbeiten zu können, muss dieses zunächst mittels verschiedener Verfahren vorbereitet werden. Ziel dieser Vorverarbeitung ist es, die Effizienz und Effektivität des anschließenden Verarbeitungsprozesses zu erhöhen und somit ein verbessertes Ergebnis zu erzielen (vgl. Lokesh und Devi 2019, S. 11672). Die Vorverarbeitung im Rahmen dieser Arbeit beinhaltet die vier Schritte Rauschreduzierung, Pausen entfernen, Framing und Windowing, welche in den folgenden Unterkapiteln genauer erläutert werden.

2.1.1 Rauschreduzierung

Um störende Frequenzen aus dem Audiosignal zu entfernen wird eine Rauschreduzierungsfunktion verwendet. Die in dieser Arbeit verwendete Funktion nutzt den sogenannten Spectral Noise Gate Algorithmus. Dabei wird zunächst die Signatur des Rauschens ermittelt. Basierend darauf kann das Rauschen anschließend verringert werden (vgl. Kiapuchinski, Lima und Kaestner 2012, S. 25).

2.1.2 Pausen entfernen

Die für die Sprecherauthentifizierung relevanten Daten stecken in dem aufgezeichneten Signal der Stimme. Sprechpausen innerhalb des Audiosignals enthalten somit keine brauchbaren Informationen, weshalb diese herausgefiltert werden müssen. Durch den vorangehenden Schritt der Rauschreduzierung kann hier ein stark vereinfachtes Verfahren gewählt werden. Liegt das Signal für einen definierten Zeitraum unterhalb einer definierten Lautstärke, werden die entsprechenden Signalwerte aus dem Gesamtsignal entfernt.

2.1.3 Framing

Für eine detaillierte Analyse des Audiosignals muss dieses in kleinere Blöcke unterteilt werden. Dieser Prozess wird als Framing bezeichnet. Dabei muss zunächst eine einheitliche Blockgröße festgelegt werden. Da Stimmsignale aufgrund der Eigenschaften des Vokaltrakts über eine Periode von 10-30 ms stationär sind, wird eine Blockgröße in dieser Zeitordnung verwendet.

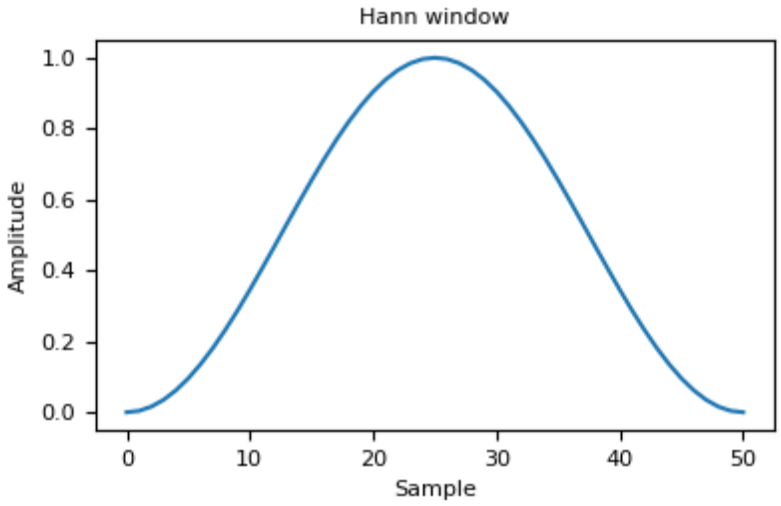

Abb. 2.1: Von Hann Fensterfunktion (*numpy.hanning — NumPy v1.24 Manual* o. D.)

Zusätzlich wird eine Überlagerungszeit definiert, welche eine Überlappung der einzelnen Blöcke verursacht. Durch die Überlappung wird ein Zusammenhang zwischen zwei benachbarten Frames und damit auch den anschließend berechneten Koeffizienten hergestellt (vgl. Richter u. a. 2022, S. 457).

2.1.4 Windowing

Um die bei der Unterteilung des Audiosignals entstandenen Diskontinuitäten aufzulösen, wird eine Fensterfunktion auf die einzelnen Blöcke angewendet. Abbildung 2.1 zeigt die von Hann Fensterfunktion, welche neben dem Hamming Fenster zu den typischen Fensterfunktionen in der Audiosignalverarbeitung zählt. Durch den Nulldurchgang am Anfang und Ende der Fensterfunktion werden die Amplituden des Blocksignals nach Anwenden der Funktion an den Grenzen auf Null gezogen, wodurch sich ein kontinuierlicher, periodischer Signalverlauf ergibt (vgl. Richter u. a. 2022, S. 462).

Wird der Schritt des Windowing nicht durchgeführt, führt dies zu einem Phänomen namens spectral leakage. Bei der Transformation des Signals von dem Zeitbereich in den Frequenzbereich resultiert der Amplitudensprung an den Blockenden in der Registrierung einer Vielzahl von Frequenzen. Wie der Name bereits beschreibt, wird aus einer eindeutigen Frequenz, ein Spektrum aus Frequenzen, die nicht Teil des Signals sind (vgl. Wu, Chen und Chen 2012, S. 1296).

2.2 Linear Predictive Coding Koeffizientenberechnung

Ausgehend von dem in Frames unterteilten Audiosignal, müssen nun für jeden Frame LPC-Koeffizienten berechnet werden, welche anschließend für die Zuordnung des Audiosignals zu einer spezifischen Stimme genutzt werden können. Die Grundlage von LPC bildet das Autoregression (AR) Modell, welches zunächst beschrieben wird. Anschließend wird der theoretische Zusammenhang zwischen AR, LPC und der menschlichen Stimme dargestellt.

2.2.1 Autoregression Modell

Die AR basiert auf dem Konzept der multiplen Regression und wird auf zeitlich veränderliche Prozesse angewandt. Dabei wird eine Kriteriumsvariable unter Betrachtung von einer beliebigen Anzahl an Prädiktorvariablen vorhergesagt (vgl. Canela, Alegre und Ibarra 2019, S. 37-38). Im speziellen Fall der AR handelt es sich bei den Prädiktorvariablen um vorhergehende Werte des Prozesses. Ein AR Modell sagt somit den Wert zu einem Zeitpunkt n, basierend auf p Vorgängerwerten des Prozesses voraus. Es gilt somit der in Formel 1 dargestellte Zusammenhang, wobei $\hat{s}_n$ den vorausgesagten Wert, s_{n-k} die vorhergehenden Werte, a_k die Regressionsgewichte und p die Anzahl an verwendeten Vorgängerwerten darstellt (Atal 1974, S. 1304).

$$\hat{s}_n = \sum_{k=1}^{p} s_{n-k} a_k \tag{1}$$

Zur Bestimmung der Regressionsgewichte wurden verschiedene rekursive Verfahren entwickelt. Neben der Yule-Walker Methode stellt der Burg Algorithmus eine beliebte Alternative dar, welcher in Marple, S. 443 beschrieben ist.

2.2.2 Linear Predictive Coding

Wie bereits zu Beginn der Arbeit erwähnt, wird bei dem Verfahren LPC der Ansatz verfolgt, Rückschlüsse von dem akustischen Signal auf die Stimmerzeugung zu ziehen. Dazu wird ein AR-Filter verwendet um ein vereinfachtes Modell des menschlichen Stimmtrakts zu erstellen. Die Regressionsgewichte a_k entsprechen dabei den LPC-Koeffizienten.

Bei der Stimmerzeugung spielen die sogenannten Formanten eine Rolle. Diese beschreiben die akustische Energie in einem unveränderlichen Frequenzbereich, welche wiederum von den Resonanz- und Interferenzeigenschaften des Artikulationsraums abhängen. Dadurch werden bestimmte Frequenzen verstärkt, während andere gedämpft werden (vgl. Fitch 2000, S. 259). Das durch die LPC-Koeffizienten erstellte Modell erfasst die Resonanzeigenschaften des Signals, wodurch Rückschlüsse auf die Formanten geschlossen werden können. Da die Struktur der For-

manten sprecherspezifisch ist, kann der Sprecher somit über die LPC-Koeffizienten identifiziert werden (vgl. Zulfiqar u. a. 2010, S. 117).

Zur Berechnung der LPC-Koeffizienten wird zunächst die selbe Annahme wie in Kapitel 2.1.3 getroffen, dass sich die Form des Vokaltrakts und das in den Stimmritzen erzeugte Signal über den betrachteten Zeitraum nicht verändert (vgl. Atal 1974, S. 1304). Somit lassen sich die Koeffizienten des AR-Filters mittels des Burg-Algorithmus berechnen.

3 Technische Umsetzung

Da die Zuordnung der erzeugten LPC-Koeffizienten zu einem spezifischen Sprecher mittels eines NN umgesetzt wird, wird auf die Programmiersprache Python zurückgegriffen. Diese ermöglicht die Verwendung des von Google entwickelten Machine Learning Frameworks TensorFlow. Folglich findet auch die Implementierung der Signalvorverarbeitung, sowie die LPC-Berechnung mit Hilfe der Sprache Python statt.

Um Programmierfehler zu vermeiden, sowie die Effizienz des Codes zu erhöhen, werden Funktionen aus verschiedenen Bibliotheken verwendet. Als Basis wird die Bibliothek `numpy` verwendet, welche Funktionen für die Bearbeitung von Arrays und Matrizen bereitstellt, sowie die Bibliothek `librosa` für audiospezifische Funktionen wie das Laden von WAV-Dateien.

3.1 Klasse AudioPreprocessor

Die Klasse `AudioPreprocessor` (vgl. Quellcode A.1) beinhaltet die Funktionen für die Schritte der Signalvorverarbeitung (vgl. Kapitel 2.1). Die Funktion `remove_noise` implementiert die Rauschreduzierung unter Verwendung der Bibliothek `noisereduce`. Für die Funktion `remove_silence` wurde wie bereits erwähnt ein eigener Algorithmus entwickelt (vgl. Zeile 67-89), der in Kapitel 2.1.1 genauer erläutert ist. Die abschließende Unterteilung des Audiosignals in Frames, sowie das Windowing der Frames findet mit Hilfe von `numpy`-Operationen in den Funktionen `create_frames` und `window_frames` statt. Die passende Fensterfunktion wird dabei ebenfalls durch die `numpy`-Bibliothek bereitgestellt (vgl. Zeile 107).

3.2 Klasse FeatureExtractor

Mit Blick auf die an diese Arbeit folgende Studienarbeit wird für die Implementierung der Koeffizientenberechnung ein Ansatz gewählt, der eine einfache Erweiterung des Programms um verschiedene andere Verfahren wie etwa MFCC ermöglicht. Dazu wird das Designmuster Strategie in abgewandelter Form verwendet, wobei zunächst ein Interface für die Berechnungsverfahren

erstellt werden muss (vgl. Quellcode A.2). Dieses definiert die Funktion `calculate_features`, welche in den abgeleiteten Klassen implementiert wird. Die Klasse `LPCExtractor` (vgl. Quellcode A.3) nutzt hierfür die von der Bibliothek `librosa` bereitgestellt Funktion `lpc` um für die übergebenen Frames die zugehörigen LPC Koeffizienten zu berechnen und anschließend zurückzugeben. Der LPC-Koeffizient nullter Ordnung wird dabei von der Funktion standardmäßig mit der Zahl Eins befüllt und ist kein Teil der berechneten LPC-Ordnung, weshalb dieser manuell entfernt werden muss (vgl. Z. 10).

Die Klasse `FeatureExtractor` (vgl. Quellcode A.4) implementiert die Funktion `extract_features`, welcher über den Parameter `feature_list` eine genaue Anweisung über die zu berechnenden Koeffizienten übergeben werden kann (vgl. Z. 19). Dabei kann im Speziellen eine Angabe zu der Art der Koeffizienten, der Anzahl an zu berechnenden Koeffizienten, sowie der zusätzlich zu berechnenden Ableitungs-Ordnungen übergeben werden (vgl. Z. 23).

4 Validierung

Damit sichergestellt werden kann, dass die in dieser Arbeit erarbeitete Parametrisierung von Stimmaufzeichnungen eine Aussage über die jeweilige sprechende Person ermöglicht, muss der Zusammenhang zwischen den berechneten LPC-Koeffizienten und der sprechenden Person gezeigt werden. Dies erfolgt in der Klasse `FeatureEvaluator` (vgl. Quellcode A.5).

Hierfür wird wie bereits eingangs erwähnt ein einfaches NN mit Hilfe der Bibliothek `tensorflow` trainiert. Das NN besteht dabei aus einem Input-Layer mit 30 * 12 Features = 360 Neuronen, zwei Hidden-Layern mit je 16 Neuronen, sowie einem Output-Layer (vgl. Z. 106-111). Da im Output-Layer für jede Sprecher-ID ein Neuron erstellt wird, passt sich die Anzahl der Neuronen an die höchste verwendete Sprecher-ID an (vgl. Z. 100).

Als Datengrundlage kommt ein von Vibhor Jain erstellter Datensatz zum Einsatz, welcher auf der Internetseite kaggle.com zur Verfügung steht und Audio-Datensätze zu 50 unterschiedlichen Sprechern bereitstellt (vgl. Vibhor Jain 2019). Für jeden Sprecher existieren dabei Aufzeichnungen mit einer Dauer von bis zu einer Stunde, welche in einminütige WAV Dateien heruntergebrochen wurden. Die Dateien mit Index Null bis einschließlich 14 jedes Sprechers werden zum Training des NN verwendet. Alle Dateien ab Index 15 können somit zum Testen des NN verwendet werden.

Für die Generierung der Trainingsdaten für das NN wird der in Kapitel 3 beschriebene Ablauf durchgeführt. Dabei wird eine Blockgröße von 500 Samples mit einer Überlappung von 100 Samples gewählt. Für die Personen 21 bis 30 werden je 1000 Chunks bestehend aus jeweils 30 aufeinanderfolgenden Frames generiert. Da für jeden Frame zwölf Koeffizienten berechnet werden, enthält jeder Chunk somit 360 LPC-Werte. Für eine einfach skalierbare Erstellung des

		Testdaten Sprecher-ID									
		21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Zugeordnete Sprecher-ID	**21**	**437**	3	233	229	31	0	0	0	1	0
	22	2	**560**	0	0	0	11	22	74	22	39
	23	258	0	**701**	15	24	0	0	0	0	0
	24	257	0	33	**743**	33	0	0	0	0	0
	25	46	12	33	13	**912**	0	0	0	0	0
	26	0	47	0	0	0	**535**	193	25	37	40
	27	0	64	0	0	0	142	**771**	1	10	3
	28	0	146	0	0	0	136	5	**779**	59	83
	29	0	67	0	0	0	154	6	37	**796**	15
	30	0	101	0	0	0	22	5	84	75	**820**
Abstand zu 2		179	414	468	514	879	381	578	695	721	737

Tab. 4.1: Modellvorhersagen für 1000 Testdaten pro Sprecher

Datensatzes wird die Funktion `create_dataset` (vgl. Z. 32-85) verwendet, welche neben dem erstellten Datensatz eine weitere Liste, die die Zuordnung des Datensatzes zu der Sprecher-ID enthält, zurückgibt.

Bevor die Trainingsdaten nun für das Training des NN verwendet werden, werden diese gemischt, um ein besseres Trainingsergebnis zu erzielen (vgl. Z. 103).

Für die Evaluation des trainierten Modells, wird ein Testdatensatz nach dem selben Verfahren aus den Dateien ab Index 15 für die Personen 21 bis 30 erstellt. Somit wird zunächst sichergestellt, dass es sich bei den Testdaten um unbekannte Werte für das NN handelt.

Mit der Funktion `evaluate_model` (vgl. Z. 119-127) kann nun die Genauigkeit, sowie die Fehlerrate des NN ermittelt werden. Der in dieser Arbeit verwendete Testdatensatz wurde von dem Modell zu 70,54 Prozent korrekt vorhergesagt, bei einer Fehlerrate von 5,47. Die Fehlerrate berechnet sich dabei nach dem categorical-crossentropy-Verfahren.

Betrachtet man die Vorhersagen des Modells mittels der Funktion `predict` (vgl. Z. 129-139) genauer, ergibt sich die in Tabelle 4.1 dargestellte Verteilung. Es ist zu erkennen, dass das Modell jeden Sprecher korrekt vorhergesagt hat, wobei die Zuversichtlichkeit im schlechtesten Fall 43,7 und im besten Fall 91,2 Prozent beträgt. Zwischen dem vorhergesagten Sprecher (fett) sowie dem Sprecher mit den zweitmeisten Vorhersagen (unterstrichen) liegt dabei im Durchschnitt ein Abstand von 55,7 Prozentpunkten. Relativ betrachtet wird der korrekte Sprecher im Durchschnitt 4,7 Mal so oft wie der Sprecher mit den zweitmeisten Vorhersagen zugeordnet.

5 Kritische Reflexion und Ausblick

In dieser Arbeit wurde ein lineares Verfahren entwickelt, welches Audiodateien zunächst in vier sequenziellen Schritten vorverarbeitet und anschließend in LPC-Koeffizienten umrechnet. Der theoretische Ansatz hinter der LPC-Berechnung zeigt dabei bereits ein hohes Potenzial der Koeffizienten für die Verwendung im Kontext Sprecherauthentifizierung.

Im Rahmen der Implementierung der vorgestellten Verfahren wird auf einen modularen Ansatz gesetzt, der eine Erweiterung des entwickelten Programms um verschiedene Verfahren der Koeffizientenberechnung ermöglicht, wodurch dieses als Basis für die anschließende Studienarbeit verwendet werden kann. Gleichzeitig können relevante Größen wie die Länge der zu erstellenden Frames oder die Anzahl zu berechnender Koeffizienten als Parameter den entsprechenden Funktionen übergeben werden, wodurch eine hohe Flexibilität erreicht wird.

Die abschließende Validierung der Ergebnisse dieser Arbeit bestätigen die in der Einleitung getroffene These. Mit einer Genauigkeit von 70,54 Prozent kann das trainierte NN neue Stimmaufzeichnungen den korrekten Sprechern zuordnen. Es besteht somit ein klarer Zusammenhang zwischen den berechneten Koeffizienten und der sprechenden Person. Dabei kann mit Blick auf die begrenzten Testdaten festgestellt werden, dass der korrekte Sprecher im Durchschnitt 4,7 Mal so oft gegenüber dem Sprecher mit den zweitmeisten Vorhersagen zugeordnet wird, was die Effektivität der Koeffizienten noch einmal verstärkt hervorhebt.

Im Kontext der anschließenden Studienarbeit zeigen die Ergebnisse, dass die LPC-Koeffizienten gewinnbringend für die Authentifizierung von Sprechern sind. Diese Arbeit bietet somit die Grundlage für verschiedene Ansätze, die im Rahmen der Studienarbeit aufgefasst und vertieft werden können.

Durch Anpassungen der Koeffizienten-Zusammensetzung kann untersucht werden, ob die Genauigkeit des NN verbessert werden kann. Dies bezieht sich insbesondere auf die Faktoren Vorhersagegenauigkeit und Fehlerrate des NN.

Neben der Koeffizienten-Zusammensetzung kann der Fokus ebenfalls auf den Aufbau des NN gelegt werden. Das in dieser Arbeit verwendete Netz beschreibt einen Standardaufbau eines NN und ist somit nicht für die Sprecherauthentifizierung optimiert. Es kann untersucht werden, inwiefern eine Veränderung der Schichtgrößen, sowie der allgemeinen Struktur zu einer Verbesserung der Vorhersagegenauigkeit und Fehlerrate führt.

Als dritte Option besteht die Möglichkeit der Erweiterung des entwickelten LPC Verfahrens. Durch weitere Rechenschritte können LPC-Koeffizienten in Linear Prediction Cepstral Coefficient (LPCC) umgerechnet werden. Im Rahmen einer Anschlussarbeit kann evaluiert werden, ob die Umrechnung in LPCC zu einer Verbesserung der Sprecherauthentifizierung führt.

Literatur

Atal, B. S. (Juni 1974). „Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification“. en. In: *The Journal of the Acoustical Society of America* 55.6, S. 1304–1312. ISSN: 0001-4966. DOI: 10.1121/1.1914702. URL: http://asa.scitation.org/doi/10.1121/1.1914702 (besucht am 16. 02. 2023).

Canela, Miguel Ángel, Inés Alegre und Alberto Ibarra (2019). „Multiple Regression“. en. In: *Quantitative Methods for Management*. Cham: Springer International Publishing, S. 37–45. ISBN: 978-3-030-17553-5 978-3-030-17554-2. DOI: 10.1007/978-3-030-17554-2_4. URL: http://link.springer.com/10.1007/978-3-030-17554-2_4 (besucht am 17. 02. 2023).

Chelali, Fatma Zohra und Amar Djeradi (Sep. 2017). „Text dependant speaker recognition using MFCC, LPC and DWT“. en. In: *International Journal of Speech Technology* 20.3, S. 725–740. ISSN: 1381-2416, 1572-8110. DOI: 10.1007/s10772-017-9441-1. URL: http://link.springer.com/10.1007/s10772-017-9441-1 (besucht am 23. 01. 2023).

Fitch, W.Tecumseh (Juli 2000). „The evolution of speech: a comparative review“. en. In: *Trends in Cognitive Sciences* 4.7, S. 258–267. ISSN: 13646613. DOI: 10.1016/S1364-6613(00)01494-7. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364661300014947 (besucht am 07. 03. 2023).

Kiapuchinski, Davi Miara, Carlos Raimundo Erig Lima und Celso Antonio Alves Kaestner (Dez. 2012). „Spectral Noise Gate Technique Applied to Birdsong Preprocessing on Embedded Unit“. In: *2012 IEEE International Symposium on Multimedia*. Irvine, CA, USA: IEEE, S. 24–27. ISBN: 978-1-4673-4370-1 978-0-7695-4875-3. DOI: 10.1109/ISM.2012.12. URL: http://ieeexplore.ieee.org/document/6424625/ (besucht am 27. 02. 2023).

Lokesh, S. und M. Ramya Devi (Sep. 2019). „Speech recognition system using enhanced mel frequency cepstral coefficient with windowing and framing method“. en. In: *Cluster Computing* 22.S5, S. 11669–11679. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-017-1447-6. URL: http://link.springer.com/10.1007/s10586-017-1447-6 (besucht am 07. 02. 2023).

Marple, L. (Aug. 1980). „A new autoregressive spectrum analysis algorithm“. en. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4, S. 441–454. ISSN: 0096-3518. DOI: 10.1109/TASSP.1980.1163429. URL: http://ieeexplore.ieee.org/document/1163429/ (besucht am 17. 02. 2023).

numpy.hanning — NumPy v1.24 Manual (o. D.). URL: https://numpy.org/doc/stable/reference/generated/numpy.hanning.html (besucht am 07. 02. 2023).

Richter, Michael u. a. (2022). *Signal processing and machine learning with applications*. eng. OCLC: 1347386653. Cham: Springer. ISBN: 978-3-319-45372-9.

Vibhor Jain (Aug. 2019). *Speaker Recognition Audio Dataset*. en. URL: https://www.kaggle.com/datasets/vjcalling/speaker-recognition-audio-dataset (besucht am 12. 03. 2023).

Wu, Minshun, Degang Chen und Guican Chen (Mai 2012). „New Spectral Leakage-Removing Method for Spectral Testing of Approximate Sinusoidal Signals“. In: *IEEE Transactions on Instrumentation and Measurement* 61.5, S. 1296–1306. ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2011.2180971. URL: http://ieeexplore.ieee.org/document/6134664/ (besucht am 07. 03. 2023).

Zulfiqar, Ali u. a. (2010). „Text-Independent Speaker Identification Using VQ-HMM Model Based Multiple Classifier System“. In: *Advances in Soft Computing*. Hrsg. von Grigori Sidorov, Arturo Hernández Aguirre und Carlos Alberto Reyes García. Bd. 6438. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 116–125. ISBN: 978-3-642-16772-0 978-3-642-16773-7. DOI: 10.1007/978-3-642-16773-7_10. URL: http://link.springer.com/10.1007/978-3-642-16773-7_10 (besucht am 23. 01. 2023).

A Anhang

A.1 AudioPreprocessor Klasse

```
import librosa
import numpy as np
import noisereduce as nr

class AudioPreprocessor:
    @staticmethod
    def int_to_float(array, type=np.float32):
        """
        Change np.array int16 into np.float32
        Parameters
        ----------
        array: np.array
        type: np.float32
        Returns
        -------
        result : np.array
        """

        if array.dtype == type:
            return array

        if array.dtype not in [np.float16, np.float32, np.
           float64]:
            if np.max(np.abs(array)) == 0:
                array = array.astype(np.float32)
                array[:] = 0
            else:
                array = array.astype(np.float32) / np.max(np.
                   abs(array))

        return array

    @staticmethod
```

```
32      def float_to_int(array, type=np.int16, divide_max_abs=True)
            :
33          """
34          Change np.array float32 / float64 into np.int16
35          Parameters
36          ----------
37          array: np.array
38          type: np.int16
39          Returns
40          -------
41          result : np.array
42          """
43
44          if array.dtype == type:
45              return array
46
47          if array.dtype not in [np.int16, np.int32, np.int64]:
48              if np.max(np.abs(array)) == 0:
49                  array[:] = 0
50                  array = type(array * np.iinfo(type).max)
51              else:
52                  if divide_max_abs:
53                      array = type(array / np.max(np.abs(array))
                                * np.iinfo(type).max)
54                  else:
55                      array = type(array * np.iinfo(type).max)
56
57          return array
58
59      @staticmethod
60      def remove_noise(y, sr):
61          # prop_decrease 0.8 only reduces noise by 0.8 -> sound
                quality is better than at 1.0
62          y_ = nr.reduce_noise(y=y, sr=sr, prop_decrease=0.8)
63
64          return y_
65
```

```
66      @staticmethod
67      def remove_silence(y):
68          threshold = 0.005
69          pause_length_in_ms = 200
70          keep_at_start_and_end = 50
71          counter_below_threshold = 0
72          indices_to_remove = []
73
74          for i, amp in enumerate(y):
75              if abs(amp) < threshold:
76                  counter_below_threshold += 1
77              else:
78                  if counter_below_threshold > pause_length_in_ms
                        :
79                      for index in range(i-
                            counter_below_threshold+
                            keep_at_start_and_end, i-
                            keep_at_start_and_end):
80                          indices_to_remove.append(index)
81                  counter_below_threshold = 0
82
83          if counter_below_threshold > pause_length_in_ms:
84              for index in range(len(y)-counter_below_threshold+
                    keep_at_start_and_end, len(y)-
                    keep_at_start_and_end):
85                  indices_to_remove.append(index)
86
87          y_ = np.delete(y, indices_to_remove)
88
89          return y_
90
91      @staticmethod
92      def create_frames(y, frame_size, overlap):
93          frames = []
94
95          if overlap >= frame_size or frame_size <= 0 or overlap
                < 0:
```

```
96              return frames
97
98          index = 0
99
100         while index + frame_size < y.shape[0]:
101             frames.append(y[index: index + frame_size])
102             index = index + frame_size - overlap
103
104         return frames
105
106     @staticmethod
107     def window_frames(frames, window_function=np.hanning):
108         windowed_frames = []
109
110         for frame in frames:
111             windowed_frames.append(frame * window_function(
                    frame.shape[0]))
112
113         return windowed_frames
114
115     @staticmethod
116     def load_preprocessed_frames(filepath=None, y=None, sr=None
            ):
117         if filepath is None and (y is None or sr is None):
118             raise ValueError("Either␣filepath␣or␣y␣and␣sr␣must␣
                    be␣given.")
119
120         if y is None or sr is None:
121             y, sr = librosa.load(filepath)
122
123         y = AudioPreprocessor.remove_noise(y=y, sr=sr)
124         y = AudioPreprocessor.remove_silence(y=y)
125
126         frames = AudioPreprocessor.create_frames(y=y,
                frame_size=1000, overlap=100)
127         windowed_frames = AudioPreprocessor.window_frames(
                frames=frames)
```

```
128
129         return windowed_frames
```

Listing A.1: Signalvorverarbeitung - Klasse AudioPreprocessor

A.2 ExtractorInterface Klasse

```
1 class ExtractorInterface:
2     def calculate_features(self, frames, sr, order):
3         pass
```

Listing A.2: Koeffizientenberechnung Interface - Klasse ExtractorInterface

A.3 LPCExtractor Klasse

```
1 from FeatureExtractor.ExtractorInterface import
    ExtractorInterface
2
3 import librosa
4
5 class LPCExtractor(ExtractorInterface):
6     def calculate_features(self, frames, sr, order):
7         lpc_coefficients = []
8
9         for frame in frames:
10            lpc_coefficients.append(librosa.lpc(y=frame, order=
                order)[1:])
11
12        return lpc_coefficients
```

Listing A.3: Koeffizientenberechnung LPC - Klasse LPCExtractor

A.4 FeatureExtractor Klasse

```
1 from FeatureExtractor.LPCExtractor import LPCExtractor
2
3 import librosa
```

```
4  import numpy as np
5  from enum import Enum
6
7  class Feature(Enum):
8      LPC = 0
9
10 class FeatureExtractor:
11     def __init__(self, frames, sr):
12         self.frames = frames
13         self.sr = sr
14         self.extractors = [
15             LPCExtractor()
16         ]
17         self.last_feature_count = 0
18
19     def extract_features(self, feature_list):
20         """_summary_
21
22         Args:
23             feature_list ((Feature, int, int[])[]): 2D List of
                Features (enum) + order (int) + deltas (int[])
                lists to extract
24
25         Returns:
26             NDArray[]: Array of requested features for each
                frame
27         """
28         feature_set = None
29
30         for feature_info in feature_list:
31             features = self.extractors[feature_info[0].value].
                calculate_features(self.frames, self.sr,
                feature_info[1])
32             if feature_set is None:
33                 feature_set = np.array(features)
34             else:
```

```
35                  np.concatenate((feature_set, np.array(features)
                        ), axis=1)
36
37              for delta in feature_info[2]:
38                  delta_features = librosa.feature.delta(np.array
                        (features), order=delta, mode='nearest')
39                  np.concatenate((feature_set, delta_features),
                        axis=1)
40
41          feature_set = feature_set.tolist()
42          self.last_feature_count = len(feature_set[0])
43
44          return feature_set
45
46      def get_last_feature_count(self):
47          return self.last_feature_count
```

Listing A.4: Koeffizientenberechnung - Klasse FeatureExtractor

A.5 FeatureEvaluator Klasse

```
1  from DatasetHandler.DatasetHandler import DatasetHandler
2  from AudioPreprocessor.AudioPreprocessor import
       AudioPreprocessor
3  from FeatureExtractor.FeatureExtractor import FeatureExtractor
4
5  import numpy as np
6  import librosa
7  import tensorflow as tf
8
9  class FeatureEvaluator:
10     def __init__(self, dataset_base_path) -> None:
11         self.dataset = DatasetHandler(dataset_base_path)
12         self.X = None
13         self.y = None
14         self.X_evaluation = None
15         self.y_evaluation = None
```

```
16          self.model = None
17
18      def get_model_dataset(self):
19          return self.X, self.y
20
21      def set_model_dataset(self, X, y):
22          self.X = np.asarray(X)
23          self.y = np.asarray(y)
24
25      def get_evaluation_dataset(self):
26          return self.X_evaluation, self.y_evaluation
27
28      def set_evaluation_dataset(self, X, y):
29          self.X_evaluation = np.asarray(X)
30          self.y_evaluation = np.asarray(y)
31
32      def create_dataset(self, speaker_ids, extraction_pattern,
            frames_per_chunk, chunks_per_speaker, samples_per_frame,
             samples_overlap, window_function=np.hanning,
            start_at_file_index=0):
33          frames_per_speaker = frames_per_chunk *
                chunks_per_speaker
34
35          dataset = []
36          dataset_speaker_ids = []
37
38          print("Create dataset process started.")
39          print(f"{samples_per_frame} samples per frame, {
                samples_overlap} samples overlap, {
                frames_per_speaker} frames per speaker.")
40          print()
41
42          for speaker_id in speaker_ids:
43              print(f"Creating dataset for speaker {speaker_id
                    :02}:")
44              # get frames_per_speaker frames for speaker_id
45              feature_list = []
```

```
46                  file_index = start_at_file_index
47                  while (len(feature_list) < frames_per_speaker):
48                      file_path = self.dataset.get_speaker_file_path(
                            speaker_id, file_index)
49
50                      # Load audio file
51                      print(f"Loading␣dataset␣{file_index:04}␣...",
                            end="\r")
52                      y, sr = librosa.load(file_path)
53
54                      # Preprocess audio file
55                      y = AudioPreprocessor.remove_noise(y=y, sr=sr)
56                      y = AudioPreprocessor.remove_silence(y=y)
57                      # frame_size: frame_duration * sr, overlap:
                            frame_size * overlap
58                      frames = AudioPreprocessor.create_frames(y=y,
                            frame_size=samples_per_frame, overlap=
                            samples_overlap)
59                      windowed_frames = AudioPreprocessor.
                            window_frames(frames=frames, window_function
                            =window_function)
60
61                      # Extract features
62                      feature_extractor = FeatureExtractor(
                            windowed_frames, sr)
63                      features = feature_extractor.extract_features(
                            extraction_pattern)
64
65                      feature_list.extend(features)
66                      file_index += 1
67
68                  print(f"Extracted␣features␣for␣{len(feature_list)}␣
                        frames␣from␣{file_index␣-␣start_at_file_index}␣
                        files.")
69                  print(f"Concatenating␣frames␣to␣chunks␣of␣{
                        frames_per_chunk}␣frames␣...")
70
```

```
71              # concat features to chunks
72              chunks = []
73              for i in range(0, len(feature_list),
                    frames_per_chunk):
74                  print(f"Chunk: {int(i/20):06}", end="\r")
75                  chunks.append(np.concatenate(feature_list[i:i+
                        frames_per_chunk]))
76
77              print(f"Created {len(chunks)} chunks, using first {
                    chunks_per_speaker} chunks.")
78              print()
79
80              dataset.extend(chunks[:chunks_per_speaker])
81              # create numpy array that has chunks_per_speaker
                    times value speaker_id
82              dataset_speaker_ids.extend(np.full((
                    chunks_per_speaker), speaker_id))
83
84          print("Dataset created.")
85          return dataset, dataset_speaker_ids
86
87      def unison_shuffled_copies(self, X, y):
88          a = np.asarray(X)
89          b = np.asarray(y)
90          assert len(a) == len(b)
91          p = np.random.permutation(len(a))
92          return a[p], b[p]
93
94      def create_nn_model(self, epochs):
95          if (self.X is None or self.y is None or len(self.X) !=
                len(self.y)):
96              print("Model dataset is not set or not valid.")
97              return
98
99          input_layer_neurons = self.X[0].shape[0]
100         output_layer_neurons = np.max(self.y) + 1
101
```

```
102         # shuffle dataset
103         X, y = self.unison_shuffled_copies(self.X, self.y)
104 
105         # create model
106         model = tf.keras.models.Sequential([
107             tf.keras.layers.Flatten(input_shape=[
                input_layer_neurons]),
108             tf.keras.layers.Dense(16, activation=tf.nn.relu),
109             tf.keras.layers.Dense(16, activation=tf.nn.relu),
110             tf.keras.layers.Dense(output_layer_neurons,
                activation=tf.nn.softmax),
111         ])
112 
113         model.compile(optimizer=tf.optimizers.Adam(), loss='
            sparse_categorical_crossentropy', metrics=['accuracy
            '])
114 
115         model.fit(X, y, epochs=epochs, verbose=0)
116 
117         self.model = model
118 
119     def evaluate_model(self):
120         if (self.model is None or self.X_evaluation is None or
            self.y_evaluation is None):
121             print("Model or evaluation dataset is not set.")
122             return
123 
124         loss, accuracy = self.model.evaluate(self.X_evaluation,
            self.y_evaluation, verbose=0)
125 
126         print(f"Model loss: {loss}")
127         print(f"Model accuracy: {accuracy}")
128 
129     def predict(self, X):
130         if (self.model is None):
131             print("Model is not set.")
132             return
```

```
133
134             prediction = self.model.predict(X)
135
136             ids = np.unique(np.argmax(prediction, axis=1))
137
138             for predicted_id in ids:
139                 print(f"ID {predicted_id}: {np.count_nonzero(np.
                        argmax(prediction, axis=1) == predicted_id)}")
```

Listing A.5: Koeffizientenvalidierung - Klasse FeatureEvaluator