Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit (Informatik)

Entwicklung einer Android-App zur Erkennung von Redeanteilen im Unterricht

Autor	Arash Besadi
Hauptbetreuung	Dr. Thilo Stadelmann Dr. Mark Cieliebak
Datum	05.06.2015

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

.....

Unterschriften:

.....

.....

.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Eine Person kann automatisch anhand ihrer Stimme erkannt werden. Dieses Verfahren wird auch automatische Sprechererkennung genannt und kann mit dem heutigen Stand der Technik in einer Smartphone-Applikation implementiert werden. Der Benutzer kann mit einer Smartphone-App ausserdem seinen Redeanteil für ein aufgenommenes Gespräch berechnen lassen, was eine Vielzahl von Anwendungsmöglichkeiten ermöglicht. So lässt sich dann beispielsweise herausfinden, wie gross der Redeanteil eines Dozenten während einer Vorlesung ist beziehungsweise wie stark sich die Studenten am Unterricht beteiligen. Jedoch findet sich zurzeit keine entsprechende Smartphone-App auf dem Markt, welche so ein Verfahren ermöglichen würde. In dieser Bachelorarbeit wurde zur Berechnung des Redeanteils eine Android-App entwickelt.
Für die Implementierung der Android-App wurden verschiedene Bibliotheken getestet, wobei die Kombination von drei Bibliotheken zum Erfolgt geführt hat. Anhand der Funktionen der Sphinx4 Bibliothek wird das Sprachsignal der Stimme zuerst im Preprocessing-Vorgang verstärkt und für die Weiterverwendung in viele kleine Blöcke unterteilt. Aus diesen werden dann die Features extrahiert, welche die Merkmale der Stimme repräsentieren. Mit dem k-Means Algorithmus aus der Apache Commons Math Bibliothek werden im nächsten Schritt die Features in mehreren Gruppen unterteilt, woraus sich dann die Parameter bestimmen lassen. Diese Parameter dienen dann zum Initialisieren des Gaussian Mixture Models (GMM) aus der sclib Bibliothek, welches ein Sprechermodell der Stimme darstellt. Mit der Log-Likelihood Funktion eines GMM kann schlussendlich die Wahrscheinlichkeit einer Stimme zum Sprechermodell bestimmt werden. Zum Testen der Android-App wurden die TIMIT-Daten verwendet, welche hochqualitative Aufnahmen von 630 Sprecher enthalten. Die Testergebnisse zeigten bezüglich der 630 Sprecher eine 100% Sprechererkennungsrate. In weiteren Experimenten wurde die Android-App noch anhand von selbst aufgezeichneten Gesprächen getestet, welche mit dem Smartphone-Mikrophone aufgenommen wurden. Diese Experimente führten zur Erkenntnis, dass für eine hohe Sprechererkennungsrate ein Gespräch am besten alle 2 Sekunden analysiert werden sollte. Im Allgemeinen zeigen die Ergebnisse, dass die Android-App für Aufnahmen mit hoher Qualität eine sehr gute Sprechererkennungsrate erreicht.

Abstract

A person can be automatically recognized by his or her voice. This procedure is called automatic speaker recognition and can be implemented with state of the art development in a smartphone application. In addition, a user can use their smartphone to calculate the speech ratio for a recorded conversation. For example, one may find out how high the speech ratio of a professor during lecture is in respect to how high the involvement of the students is. However, there is currently no such smartphone app on the market that would enable such a procedure. In this thesis, an android app is developed to calculate the speech ratio.
Several libraries were used for the implementation of this project; the combination of three of these libraries is successful. The functions from the sphinx 4 library are used to enhance the speech signal during the preprocessing and split them into small pieces for the next step. From these, the features are extracted which represent the characteristic of the voice. With the k-Means algorithm form the Apache Commons Math library, the features are put into groups from which the parameters are calculated. These parameters are used to initialize the Gaussian Mixture Model (GMM) from the sclib library, which represents a voice model. Finally, the GMMs Log-Liklihood can be used to calculate the probability of the voice to the voice model. For the testing of the android app, the TIMIT-Data was used and contained high quality recordings of 630 speakers. The test results showed a speaker recognition of 100% regarding the 630 speakers. In further experiments the android-app was tested with its own recorded conversations that were recorded with the smartphone microphone. The results showed that for a high speaker recognition rate, the conversation should be analyzed all 2s. Overall, the results showed that the android achieved a great speaker recognition rate with high quality recordings.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Jedes Individuum hat eine einzigartige Stimme, sodass diese sich heutzutage mit verschiedensten Verfahren automatisch erkennen lässt. Dies ermöglicht eine Vielzahl von verschiedenen Anwendungszwecken. Vor allem für Organisationen, welche in Bereichen der Bekämpfung von Kriminalität und Terror arbeiten, spielt die automatische Sprechererkennung eine sehr wichtige Rolle. Die Europäische Kommission hat dafür das Speaker Identification Integrated Project (SIIP) gegründet. Polizeibehörden sollen somit die Möglichkeit erhalten, mit einer automatischen Sprechererkennungssoftware und einer gemeinsamen internationalen Datenbank von Stimmproben, kriminelle künftig anhand ihrer Stimme überführen zu können (A.Ziech, 2014). Auch Geheimdienste dürften an der automatischen Identifizierung der Stimme ein grosses Interesse haben.

In einem Artikel von Golem wird daher die Sprechererkennung als die neue biometrische Authentifizierung beschrieben (J.Wendt, 2015). Immer mehr Firmen verwenden dabei akustische Biometriesysteme, um ihre Daten besser vor Angreifern zu schützen. Wie in einem Beispiel im Artikel beschrieben, riefen die Angreifer bei der Hotline ihrer Zielperson an und gaben einfach erhaltende Daten an und erhielten so Zugang zum Account. Solche Angriffe könnten mit Sprechererkennung- Authentifizierung erschwert werden. Aus diesem Grund setzten vor allem Banken nun auf Sprechererkennung und ersparen dem Kunden die Eingabe eines Pins.

Es gibt aber noch andere Anwendungsbereiche. Durch die automatische Sprechererkennung lässt sich zusätzlich auch objektiv bestimmen, wie gross der Redeanteil einer Person am Gespräch ist. Dies beantwortet die Frage „wer spricht wann“ und wird als speaker diarisation bezeichnet (X.Anguera et al., 2010).

Um dies zu verdeutlichen folgen hier einige Szenarien.

- Während einer Dozenten-Evaluation möchte man gerne wissen, ob der Dozent die Studenten in den Unterricht miteinbezieht bzw. wie gross die Unterrichtsbeteiligung der Studenten ist.
- Bei einem Bewerbungsgespräch möchte der Gesprächsführer gerne wissen, wie stark der Kandidat am Gespräch teilgenommen hat.

In beiden Beispielen kann der der Redeanteil der Person bei der Gesamtbeurteilung als Indikator mitwirken.

1.2 Aufgabenstellung

Das Ziel dieser Bachelorarbeit ist eine App zur automatischen Sprechererkennung zu entwickeln, welche den Redeanteil einer Person an einem Gespräch berechnet. Die App soll auf einem Android-Smartphone laufen. Dabei soll der Benutzer die Möglichkeit haben, mittels seines Smartphone-Mikrophones eine Stimmprobe aufzunehmen. Durch die Stimmprobe wird die Stimme des Benutzers in einem Gespräch erkannt und entsprechend der Redeanteil am Gespräch angegeben. Der restliche Redeanteil wird als Gesamteinheit den anderen Personen zugeschrieben.

1.3 Umfeld

Diese Bachelorarbeit wird im Rahmen eines Forschungsprojekts zur automatischen Sprechererkennung am Institut für angewandte Informationstechnologie (InIT) durchgeführt. In einem Zusammenschluss zwischen Thilo Stadelmann, Mark Cieliebak und Hans-Peter Hutter möchte das InIT eine Applikation entwickeln, welche in den Vorlesungen den Redeanteil von Dozenten berechnet. Als erste Testlaufstelle sind die Pädagogische Hochschule Zürich und die ZHAW gedacht.

Am InIT wurde bereits für automatische Sprechererkennung in Matlab ein Algorithmus entwickelt, welcher erfolgreich getestet wurde. Dieser Algorithmus kann als Hilfsmittel in der Bachelorarbeit verwendet werden.

1.4 Weitere Gliederung

Der weitere Aufbau dieser Bachelorarbeit ist folgendermassen gegliedert: Kapitel 2 erklärt die Grundlagen der automatischen Sprechererkennung und befasst sich mit den Verfahren, welche bei der Implementierung eingesetzt werden. Kapitel 3 beschreibt wie die Verfahren im vorhergehenden Kapitel in Komponenten zusammengefasst werden. Die Auswahl der eingesetzten Bibliotheken wird entsprechend anhand der Komponenten bestimmt. Kapitel 4 beinhaltet die Konfiguration der Parameter zur Implementierung. Zudem wird erklärt, wie der Ablauf zum Testen mit den Trainings-Daten stattfindet. Kapitel 5 zeigt die Ergebnisse der eingesetzten Bibliotheken und den Experimenten mit den selbst aufgenommenen Gesprächen auf. Kapitel 6 fasst diese Bachelorarbeit zusammen und nennt in den kurzfristigen Folgearbeiten die weiteren Implementierungen. Für die langfristige Forschung werden weitere Experimente vorgeschlagen.

2 Grundlagen der automatischen Sprechererkennung

2.1 Ansatz

In dieser Bachelorarbeit wird für die automatische Sprechererkennung das in 1995 vorgestellte Verfahren von Douglas A. Reynolds und Richard C. Rose angewendet (A.Reynolds et al., 1995). Obwohl dieses Verfahren schon länger bekannt ist, gilt es für die Sprechererkennung immer noch als State of the Art.

Das Verfahren basiert auf der Annahme, dass eine Person anhand einer Stimmprobe erkennt werden kann. Wie Reynolds und Rose in ihrer Publikation erklären, wird zwischen den zwei Erkennungsarten verifizieren und identifizieren unterschieden. Bei der Verifikation wird mit einer Stimmprobe überprüft, ob es sich bei der Person um diejenige handelt, für die sie sich ausgibt. Die Identifizierung hingegen verfolgt das Ziel, anhand einer Stimmprobe aus einer Menge von Stimmen, die dazu passendste auszuwählen. In dieser Bachelorarbeit wird für die Sprechererkennung ein ähnliches Ziel wie bei der Identifizierung verfolgt. Aus einem Gespräch mit mehreren Stimmen wird das Gespräch in viele kleine Blöcke unterteilt. Anschliessend wird für jeden Block überprüft, ob sie zur Stimmprobe passt oder nicht. Dieser einfach erklärte Vorgang besteht aus vielen mathematischen Schritten. Das gesamte Verfahren wird in vier aufeinander aufbauenden Vorgängen unterteilt und entsprechend in vier Kapiteln beschrieben.

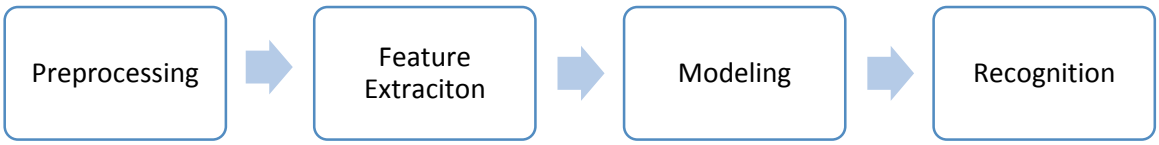

Abbildung 1: Verfahren der automatischen Sprechererkennung

In Kapitel Preprocessing 2.2 wird beschrieben, wie das Sprachsignal mit verschiedenen Filtern verstärkt und bereinigt wird und anschliessend für die Weiterverwendung in kleine Blöcke (Frames) aufgeteilt wird. Bei der Feature Extraction in Kapitel 2.3 werden danach aus den Frames eindeutige Merkmale der Stimme, sogenannte Features, extrahiert. In Kapitel Modeling 2.4 wird erklärt, wie aus den Features ein Sprechermodell erzeugt wird. Zum Schluss wird in Kapitel Recognition 2.5 aufgezeigt, wie eine Person mit ihrem Sprechermodell aus einer Menge von Stimmen identifiziert werden kann.

2.2 Preprocessing

Menschen erzeugen beim Reden ein Sprachsignal, welches aus einer kontinuierlichen Serie von Phonemen besteht. Dabei ist ein Phonem die kleinste akustische Einheit einer Sprache, welches zwei Wörter voneinander unterscheiden lässt (Wikipedia, 2015). Wie zum Beispiel Haus – Maus. So ist /h/ und /m/ ein Phonem, weil es einen Bedeutungsunterschied erzeugt.

Beim Preprocessing wird das Sprachsignal von unnötigen Informationen befreit und so aufbereitet, dass die Daten für die Feature Extraction brauchbar sind (T.Stadelmann, 2010). Dieser Vorgang besteht aus mehreren kleinen Schritten.

- Sampling
- Noise-, Silence- und Highpass-Filter
- Framing

2.2.1 Sampling

Beim Sampling wird das kontinuierliche Sprachsignal auf ein diskretes Signal reduziert. Ein Sample ist somit ein diskreter Wert in einem Zeitabschnitt des Signals. In der untenstehende Abbildung 2 repräsentiert die grüne Linie das kontinuierliche Signal und die blauen Punkte einen diskreten Sample.

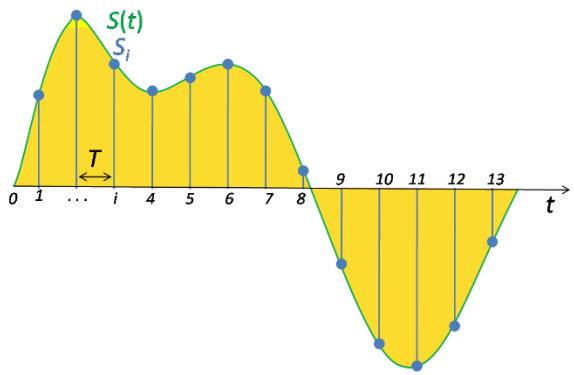

Abbildung 2: Sampling eines Signals (Binksternet, 2009)

Samplerate

Die Samplerate ist die Anzahl Samples pro Sekunde, wobei sie in der Einheit Herz angegeben wird. In unserem Fall wird das Sprachsignal mit einer Samplerate von 16000Hz abgetastet, sodass nach dem Theorem von Nyquist eine Frequenz von 8000Hz abgebildet werden kann. Die meisten spezifischen Informationen eines Sprechers in einem Sprachsignal befinden sich in einem Bereich von 500-5000 Hz (T.Stadelmann, 2010). Daher sollte eine Samplerate von 16kHzt ausreichend sein.

2.2.2 Filter

Um ein besseres Signal zu erhalten, werden beim Preprocessing verschiedene Varianten von Filtern angewendet. Beim Noise-Filter wird das Signal von Nebengeräuschen befreit, welche das Signal stören.

Der Silence-Filter, wie der Name es schon sagt, befreit das Signal von Stille. Dieser kann beim Preprocessing angewendet werden, indem die Tonhöhe bei einer Menge von Samples betrachtet wird und diese verworfen werden, falls diese eine bestimmte Grenze nicht erreichen. Die Stille kann aber auch nach der Feature Extraction (Kapitel 2.3) gefiltert werden, welches einfacher zum Implementieren ist.

Ein weiterer Filter ist der Highpass-Filter, welche hohen Frequenzen anhebt. Dadurch wird das Signal-to-noise ration (SNR) des Sprachsignals verbessert (T.Stadelmann 2010, zit. n. R. Vergin et al., 1995). Für den Highpass-Filter wird folgende rekursive Formel für n Samples verwendet:

$$s'_n = s_n - \alpha * s_{n-1}$$

Ein üblicher Wert für α ist 0.97 (CMUSphinx, 2015).

2.2.3 Framing

Im letzten Schritt des Preprocessing werden die Samples für die weitere Verarbeitung in Blöcke zusammengefasst, welche Frames genannt werden. Um spätere Berechnungen zu vereinfachen, wird eine Framegrösse von 20ms gewählt, sodass sich die Werte der Samples in diesem Intervall nicht stark verändern bzw. stationär sind (T.Stadelmann, 2010). Falls die Framegrösse kürzer ist, sind nicht genügend Samples für ein brauchbares Resultat vorhanden und falls sie länger ist, ändert sich das Signal zu stark in einem Frame.

Bei einer Samplerate von 16kHz und einer Framegrösse von 20ms sind das 16000 * 0.020 = 320 Samples pro Frame. Dabei überschneiden sich die Frames um 50% bzw. um 10ms, um auch Informationen am Rande eines Frames zu erfassen (CMUSphinx, 2015).

Aus jedem Frame werden anschliessend die Features extrahiert.

2.3 Feature Extraction

Bei der automatischen Sprechererkennung geht es darum, bestimmte Muster der Stimme automatisch zu erkennen. Das Problem ist, dass die Stimme Informationen enthält, welche für die Sprechererkennung nicht von Interesse sind. Zum Beispiel wird neben den Wörtern auch die Satzmelodie, die sogenannte Prosodie, übermittelt (M.Becker et al., 2010). Die Prosodie enthält unteranderem Informationen über die Gefühlslage und die körperliche Verfassung eines Sprechers. Das Ziel der Feature Extraction ist es daher, durch eine Kompression relevante Muster explizit hervorzuheben und nicht benötigte Informationen zu entfernen (T.Stadelmann, 2010). Features enthalten somit vereinfacht gesehen die Merkmale einer Stimme, welche Sprecher voneinander unterscheiden lassen. Es gibt verschiedene Arten von Features, wobei die am häufigsten verwendeten die Mel Frequency Cepstral Coefficients (MFCC) Features sind, welche im Jahre 1980 von Mermelstein vorgestellt wurden (T.Stadelmann, 2010).

2.3.1 Mel Frequency Cepstral Coefficients

Um die MFCC Features zu erhalten, müssen verschiedene Vorgänge durchgeführt werden.

- Die Transformation der Samplewerte vom zeitlichen Verlauf in das Frequenzspektrum mittels einer Diskreten Fourier-Transformation (DFT).
- Das Aufsummieren von Frequenzen durch eine Filterbank zu einer Filterbank Energy (FBE).
- Die Dekorrelation der FBE und die Glättung des Signals mittels einer Diskreten Cosinus-Transformation (DCT).

2.3.2 Diskrete Fourier-Transformation

Bei der DFT werden die Samplewerte durch eine Fast Fourier Transformation (FFT) vom zeitlichen Verlauf in einem Frequenzspektrum abgebildet. Dadurch lassen diese sich besser analysieren, wie zum Beispiel das explizite Rauslesen der Tonhöhe einer Frequenz. Wie in Stadelmanns These wird auch hier ein Frequenzspektrum zwischen 0 – 7600Hz verwendet.

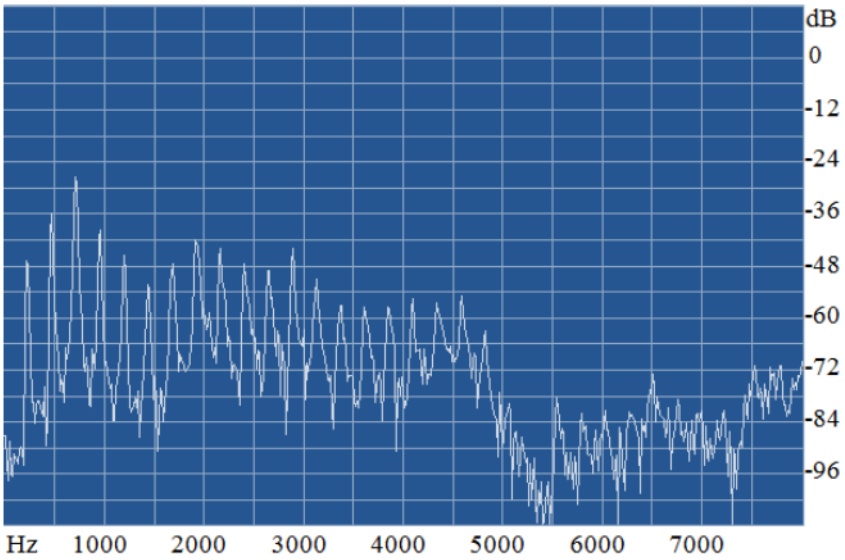

Abbildung 3: Frequenzspektrum des Satzes „she had“, (T.Stadelmann, 2010)

Da in unserem Fall sich in einem Frame von 20ms Länge ungefähr 320 Samples befinden, werden für dessen Abbildung im Frequenzspektrum entsprechend gleich viele Punkte gebraucht. Da aber die FFT die Punkteanzahl durch eine Konvention eine Potenz mit der Basis zwei sein muss und 256 nicht ausreichen, wird eine Punkteanzahl von 512 verwendet. Die Differenz zwischen den Anzahl Samples und den 512 Punkten wird durch Anfügen von Nullen ausgeglichen (zero-padding).

2.3.3 Mel Filterbank

Das menschliche Gehör kann nahe nebeneinander liegende Frequenzen nicht unterscheiden (J.Lyons, 2012). Aus diesem Grund werden mehrere Frequenzen durch Filter in Form von Dreiecken summiert und somit die Energie in den verschiedenen Bereichen des Frequenzspektrums bestimmt. Dies geschieht durch die Mel Filterbank. Normalerweise werden für eine Frequenz im Bereich von 0 - 8kHz eine Anzahl von 24 Filter verwendet (T.Stadelmann, 2010). Die Filter sind nahe 0 Hertz sehr schmal und werden linear mit höheren Frequenzen immer breiter, da das menschliche Gehör höhere Frequenzen weniger wahrnimmt.

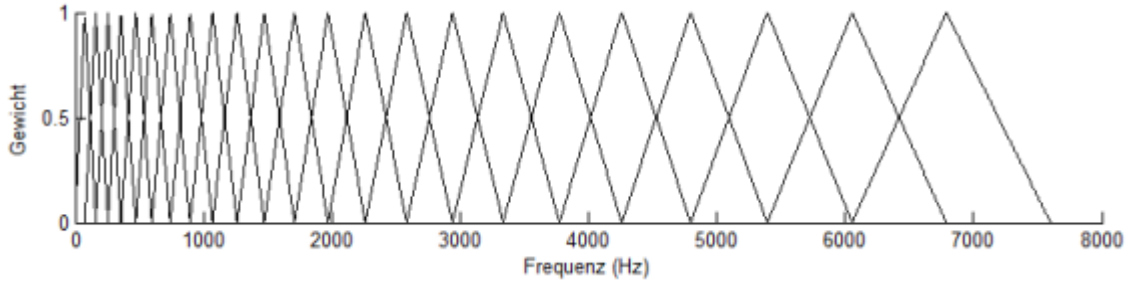

Abbildung 4: Mel Filterbank mit 24 Filtern zwischen 0 und 7600 Hz (J.Stampfli, 2014)

Nach der Berechnung der Filterbank Energien werden diese noch logarithmiert. Dies ist motiviert dadurch, dass der Mensch die Tonhöhe nicht linear wahrnimmt. Um eine doppelte Tonhöhe wahrzunehmen, braucht es normalerweise acht Mal mehr Energie (J.Lyons, 2012). Das Logarithmieren liefert dann die log-Filterbank Energy.

2.3.4 Diskrete Kosinus-Transformation

Im letzten Schritt wird noch eine DCT angewendet. Da sich die Filterbanken überschneiden, weisen die Filterbank Energien eine starke Korrelation auf (J.Lyons, 2012). Mit der DCT wird eine Dekorrelation der FBE ermöglicht. Zudem werden mit einer DCT starke Schwankungen entfernt, welches eine Glättung des Signals ermöglicht und somit die Sprechererkennungsrate verbessert. Das Resultat ist ein MFCC Feature Vektor, welcher in der ersten Dimension die Tonhöhe (Mel) und in den restlichen Dimensionen die Cepstral Koeffizienten enthält. In den Experimenten von Reynolds und Rose enthalten die Feature Vektoren insgesamt 20 Cepstral Koeffizienten. In unserem Fall wird jedoch die Tonhöhe für die Modellierung nicht benötigt, sodass der erste Koeffizient verworfen wird und daraus ein Feature Vektor mit 19 Dimensionen resultiert.

2.4 Modeling

Nachdem die Features extrahiert wurden, kann nun mit diesen die Stimme modelliert werden. Reynolds und Rose führen in ihrer Publikation zwei Hauptgründe auf, dafür ein Gaussian Mixture Model (GMM) zu verwenden.

1. Ein GMM besteht aus einer Anzahl von Normalverteilungen (Gaussian Mixture). Ihrer Ansicht nach kann aus verschiedenen Gründen vernünftig angenommen werden, dass jede Normalverteilung eine akustische Klasse von Phonemen repräsentiert. Diese reflektieren einige sprecherabhängige Artikulationen, welche einen Sprecher charakterisieren. Sie haben dabei verschiedene Anzahlen zweier Potenzen getestet, wobei 32 Normalverteilungen die beste Sprechererkennungsrate liefern. Diese Anzahl verträgt sich auch gut mit den ungefähren Anzahl von 40 Phonemen in der deutschen Sprache (Wikipedia: Phonem, 2015).

2. Ein GMM hat die besondere Eigenschaft durch Approximation eine willkürlich geformte Wahrscheinlichkeitsdichtefunktion (WDF) von Verteilungen bilden zu können.

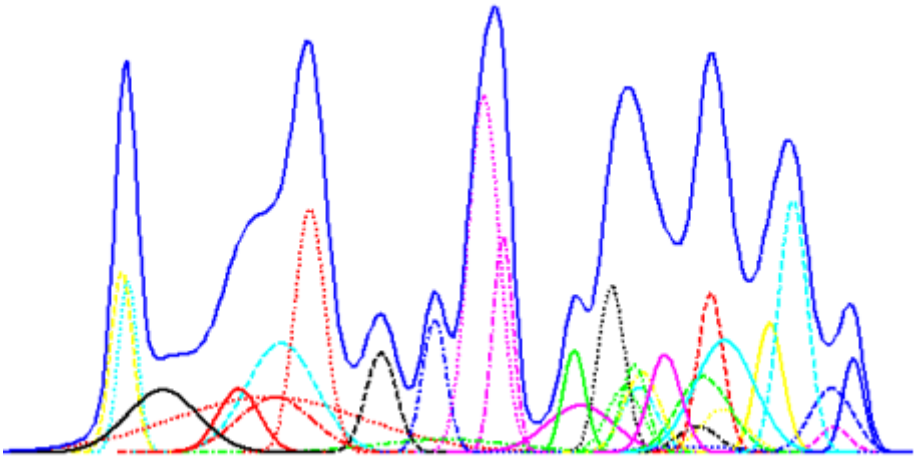

Abbildung 5: Gaussian Mixture Model (J.Stampfli, 2014)

Die nächsten Kapitel sind folgendermassen aufgebaut: Kapitel 2.4.1 beschreibt die Funktionen und Parameter eines GMM und Kapitel 2.4.2 macht auf einige Probleme der Parameter aufmerksam. Kapitel 2.4.3 erklärt wie der k-Means Algorithmus die Parameter für das Initialisieren eines GMM bildet und Kapitel 2.4.4 zeigt auf, wie die Parameter mit dem EM-Algorithmus optimiert werden.

2.4.1 Die Funktionen und Parameter eines GMM

Ein GMM wird durch folgende Parameter beschrieben:

- Gewicht w (Anteil Feature Vektoren pro Normalverteilung)
- Mittelwert μ
- Varianz σ^2

Diese werden zusammengefasst als λ:

$$\lambda = \{w_i, \mu_i, \sigma^2{}_i\};\ i = 1 \dots M \text{ (Anzahl Normalverteilungen)}$$

Die WDF eines GMM ist die gewichtete Summe aller Parameter. Die Zufallsvariable X wird hier einfachheitshalber durch die Variable x als Feature Vektor ersetzt.

$$p(x \mid \lambda) = \sum_{i=1}^{M} w_i * b_i(x)$$

$$b_i(x) = \frac{1}{\sqrt{2\pi\sigma^2{}_i}} * e^{\left(-\frac{(x-\mu_i)^2}{2\sigma^2{}_i}\right)}$$

$$\sum_{i=1}^{M} w_i = 1$$

Für weitere Informationen, wie zum Beispiel die Berechnung der Parameter λ, siehe die Publikation von Reynolds und Rose (A.Reynolds et al., 1995).

Log-Likelihood Funktion

Mit der Likelihood-Funktion lässt sich dann anhand der WDF des GMM die Wahrscheinlichkeit von T Feature Vektoren als Produkt angeben.

$$L(x \mid \lambda) = \prod_{i=1}^{T} p(x_t \mid \lambda)$$

Dabei wird wie bei Reynolds und Rose die logarithmierte Likelihood-Funktion verwendet.

2.4.2 Probleme der Parameter eines GMM

Da die Dimension eines Feature Vektors 19 beträgt, werden multivariate (mehrdimensionale) Normalverteilungen verwendet. Diese werden durch den Mittelwert Vektor μ und der Kovarianzmatrix Σ berechnet. (Wikipedia: Mehrdimensionale Normalverteilung, 2015).

Die Berechnung der vollen Kovarianzmatrix ist aber sehr rechenintensiv. Zudem muss für deren Berechnung das Inverse gebildet werden, welches bei einer singulären Matrix nicht möglich ist. Wie aber Reynolds and Rose erwähnen, ist die Berechnung der vollen Kovarianzmatrix nicht notwendig, weil dessen Hauptdiagonale ausreichend ist für ein GMM. Auf der Hauptdiagonale befinden sich nämlich die Varianzen (Wikipedia: Kovarianzmatrix, 2015).

Des Weiteren kann wegen der hohen Anzahl von Normalverteilungen und bei nicht genügend Trainingsdaten die Varianz sehr klein werden. Dies führt nach Angaben von Reynolds und Rose zu einer Singularität der Likelihood Funktion im GMM und verschlechtert die Sprechererkennungsrate. Aus diesem Grund wird empfohlen, ein Varianz-Limit von 0.1 oder 0.01 einzuführen. Falls eine Varianz das Limit nicht erreicht, wird sie durch das Limit ersetzt.

2.4.3 k-Means Algorithmus

Die Parameter des GMM werden mit dem k-Means Algorithmus initialisiert, welcher ein einfacher und weitverbreiteter Clustering Algorithmus ist. Er basiert auf dem Prinzip von machine learning, wobei zwischen unsupervised- and supervised learning unterschieden wird (T. Stadelmann zit. n. T.Mitchell, 1997). Beim k-Means handelt es sich um einen unsupervised-learning Algorithmus, bei welchem keine weiteren Informationen über die Daten vorhanden sind. Dieser Algorithmus muss daher nur anhand der Daten eine Struktur bilden können, welches Clustering genannt wird.

Der Buchstabe k in k-means steht für die Anzahl Cluster, in welche die Daten gruppiert werden. Da 32 Normalverteilungen für das GMM verwendet werden, wird k in diesem Fall auch mit 32 initialisiert. Zuerst werden zufällige Cluster Zentren (Centroid genannt) anhand der Feature Vektoren gebildet. Anschliessend werden zwischen 10-100 Iterationen die Distanz zwischen jedem MFCC Vektor zu jedem Centroid gemessen. Der Feature Vektor wird anschliessend dem Cluster mit der kleinsten gemessenen Distanz zugeteilt.

Nachdem alle Feature Vektoren den verschiedenen Clustern hinzugefügt worden sind, werden aus den Clustern die Parameter λ für das GMM bestimmt.

2.4.4 EM Algorithmus

Der Expectation-Maximization (EM) Algorithmus verfolgt das Ziel, die Parameter λ für ein Model zu optimieren. Er startet mit einem initialisierten Model λ und schätzt ein neues Model $\bar{\lambda}$, um die Wahrscheinlichkeit der Parameter zum Model zu erhöhen (A.Reynolds et al., 1995).

$$L\,(x, \lambda) > L\,(x, \bar{\lambda})$$

Die neuen Parameter dienen als Ausgangslange für die nächste Iteration. Dieser Vorgang wird solange wiederholt bis entweder ein Threshold erreicht ist oder eine bestimmte Anzahl Iteration. Da der EM Algorithmus schnell gegen ein lokales Optimum der Parameter konvergiert (T.Stadelmann, 2010), wird der Algorithmus nach 100 Iterationen beendet.

2.5 Recognition

Im Recognition-Vorgang wird die Stimme einer Person mit einem GMM identifiziert. Falls nun eine Anzahl von Stimmen überprüft werden muss, erfolgt dies über die Distanzmessung.

2.5.1 Distanzmessung

Zuerst werden für alle Stimmen die Features extrahiert. Anschliessend wird mit der Log-Likelihood Funktion die Wahrscheinlichkeit zwischen den neu extrahierten Features mit dem bestehenden GMM berechnet. Das Resultat ist eine Menge von Log-Likelihood Werten. Im nächsten Schritt werden alle Werte mit einem Threshold (Grenzwert) verglichen. Liegen die Werte unter dem Threshold, werden diese als eine andere Stimme erfasst. Liegen die Werte hingegen über dem Threshold, wurde die Aussage von derselben Person gesprochen, mit welchem das Model trainiert ist.

Für eine geeignete Bestimmung des Thresholds müssen durch Experimente die resultierenden Log-Likelihood Werte analysiert werden.

Sprecheranteil

Als Abschluss wird die Berechnung des Redeanteils näher angeschaut. Dieser ergibt sich aus der Anzahl aller Log-Likelihood Werte, welche den Threshold überschreiten, geteilt durch die Anzahl aller Log-Likelihood Werte.

$$\frac{|\,L(x) > Threshold\,|}{|\,L(x)|}$$

Es sei angenommen, dass in einem Gespräch von einer Länge von 30s alle 3s die Log-Likelihood Werte bestimmt werden. Liegen beispielsweise 3 von 10 Werten oberhalb des Thresholds, entspricht dies einem Redeanteil von 33%.

3 Konzept

3.1 Kriterium

In dieser BA soll eine Android-App entwickelt werden, welche die Stimme des Benutzers automatisch erkennt und zusätzlich dessen entsprechenden Redeanteil am Gespräch berechnet. Die höchste Priorität hat dabei die Implementierung eines Algorithmus mit einer sehr hohen Sprechererkennungsrate. Deshalb wird auf die Entwicklung einer grafischen Benutzeroberfläche (GUI) für die Android-App verzichtet, sodass mehr Zeit für die Umsetzung und das Testen des Algorithmus vorhanden ist. Um die Sprechererkennungsrate des Algorithmus zu messen, werden die TIMIT-Daten verwendet. Das Ziel des Algorithmus ist es, eine 100% Sprechererkennungsrate bezüglich den TIMIT-Daten zu erhalten.

3.1.1 TIMIT-Daten.

Bei den TIMIT-Daten handelt es sich um Aufnahmen von 630 Sprechern, welche von der Linguistic Data Consortium bereitgestellt werden.
Die Aufnahmen sind hochwertig, enthalten keine Nebengeräusche, sind nicht komprimiert und sind mit einem professionellen Mikrophone aufgenommen worden. Der Algorithmus soll in der Lage sein, mit einem trainierten Modell für jeden der 630 Sprecher dessen Stimme aus 630 Stimmen identifizieren zu können. Dabei sind für jeden Sprecher 10 Audiodateien mit einer Durchschnittlänge von 3 Sekunden vorhanden. Diese enthalten Sätze wie „Production may fall far below expectations“.

3.2 Vorgehensweise

Wie aus den Grundlagen in Kapitel 2 ersichtlich ist, besteht der Vorgang der automatischen Sprechererkennung aus den vier aufeinander aufbauenden Schritten Preprocessing, Feature Extraction, Modeling und Recognition. In allen Schritten werden diverse mathematische Berechnungen durchgeführt, welche wahrscheinlich bereits als externe Java-Bibliotheken vorhanden sind. Daher wird für die Umsetzung in Java zuerst abgeklärt, ob bereits entsprechende Bibliotheken für die oben genannten Schritte vorhanden sind und wie sie verwendet werden können. Falls diese nicht vorhanden sind, müssen sie selbst implementiert werden. Dazu besteht die Möglichkeit, die von Stadelmann in C++ geschriebene Bibliothek sclib zu verwenden und benötigte Funktionen in Java zu übersetzen. Dies sollte aber möglichst vermieden werden, da das Übersetzen von längeren mathematischen Vorgängen, wie zum Beispiel bei der Feature Extraction, sehr fehleranfällig ist.

In allem muss für die Umsetzung in Java nach Bibliotheken umgesehen werden, welche mindestens eine der drei Komponenten enthalten.

- MFCC Feature Extraction
- k-Means Algorihtmus
- GMM

Dies wird damit begründet, dass einige Vorgänge einfachheitshalber oft im selben Prozess durchgeführt werden können und hier als Komponenten bezeichnet werden. Die Feature Extraction führt beispielsweise mit Sampling und Framing ein Preprocessing durch. Ausserdem sind auch das Modeling und die Recognition mit den Log-Likelihood Berechnungen normalerweise zusammen in der GMM Komponente vorhanden. Eine Ausnahme bildet dabei der k-Means Algorithmus, welcher als unabhängige Clusteranalyse verwendet wird.

3.3 Auswahl der eingesetzten Bibliotheken

Für die Implementierung in Java werden die drei Bibliotheken CoMIRVA, Apache Commons Math und die Sphinx4 verwendet.

3.3.1 CoMIRVA

Die CoMIRVA Bibliothek bietet verschiedene Java Algorithmen bezüglich Audio und Information Retrieval. Sie wurde von Markus Schedl entwickelt und ist unter GNU GPL lizenziert. Der Vorteil dieser Bibliothek ist, dass alle drei benötigten Komponenten für die automatische Sprechererkennung vorhanden sind. Hingegen wurde sie seit 2010 nicht mehr weiterentwickelt und ist hauptsächlich von einer Person geschrieben worden. Dementsprechend ist die Wahrscheinlichkeit auf mögliche Bugs im Code gross, welche sich beim Einsatz dieser Bibliothek bemerkbar machten.

3.3.2 Apache Commons Math

Die Apache Commons Math ist eine in Java geschriebene Bibliothek mit mathematischen und statistischen Algorithmen von der Apache Software Foundation. Die Bibliotheken stehen unter der Apache License Version 2.0 und können somit auch in kommerziellen Projekten verwendet werden. Der Vorteil bei der Commons Math Bibliothek und im Allgemeinen der Apache Software Foundation ist dessen aktive Community. Die Chance auf mögliche Fehler im Code ist sehr gering, da diese von mehreren Mitgliedern geschrieben und kontrolliert werden.

3.3.3 Sphinx4

Die CMU Sphinx ist ein Projekt der Carnegie Mellon University (CMU), welches schon seit 20 Jahren im Bereich der Spracherkennung tätig ist (CMUSphinx, 2015). Das Projekt steht unter er BSD-Lizenz und kann für kommerzielle Tätigkeiten frei verwendet werden. Wie auch bei der Apache Foundation ist der grösste Vorteil hier die aktive Community, welche ständig neue Versionen veröffentlicht.

Mit der Sphinx4 Bibliothek enthält das CMU Sphinx Projekt eine Bibliothek, welche für die Spracherkennung (Das Umwandeln von gesprochenen Wörtern in Text) gedacht. Allerdings lässt sich die Konfiguration für die Feature Extraction in einer XML Datei sehr flexibel einstellen. In dieser kann definiert werden, welche Vorgänge mit welchen Werten stattfinden. Dies ist besonders von Vorteil, da CMU Sphinx ein Feature aus einem Cepstrum und dessen Delta und Double Delta (erste und zweite Ableitung) definiert (CMUSphinx, 2015). Wegen der grossen Freiheit der Bibliothek lässt sich aber bestimmen, dass die Rückgabewerte der DCT als Feature verwendet werden sollen.

3.4 Auswahl der eingesetzten Komponenten

Da nicht alle ausgewählten Bibliotheken alle benötigte Komponenten beinhalten, wurden verschiedenen Kombinationen von Komponenten versucht.

Die untenstehende Tabelle zeigt auf, welche Bibliotheken mit welchen Komponenten zum gewünschten Ergebnis einer 100% Sprechererkennungsrate bezüglich der TIMIT-Daten führten und welche nicht.

	Wurde umgesetzt und führte zum gewünschten Ergebnis
	Wurde umgesetzt, hatte aber wenig Einfluss auf das Ergebnis
	Wurde umgesetzt, führte aber nicht zum gewünschten Ergebnis
	Wurde nicht umgesetzt.

Tabelle 1: Beschreibung der Ergebnisse

Bibliothek / Komponente	CoMIRVA	Apache Commons Math	Sphinx4	sclib
MFCC Feature	X		X	X
k-Means	X	X		X
GMM	X	X	X	X

Tabelle 2: Auswahl der eingesetzten Bibliotheken und Komponenten

Die MFCC Feature Extraction der Sphinx4, der k-Means Algorithmus der Apache Commons Math und der in Java übersetzte Algorithmus für das GMM von der sclib Bibliothek führten in Kombination miteinander bei den TIMIT-Daten zu einer Sprechererkennungsrate von 100%.

Zuerst wurde jedoch mit der CoMIRVA Bibliothek gearbeitet, da sie den pragmatischen Vorteil hatte, alle drei benötigten Komponenten zu enthalten. Die Komponenten wurden dennoch eine nach der anderen verworfen, da sie nicht zum gewünschten Ergebnis führten oder gar nicht mit den benötigten Parametern funktionierten. Auch das GMM der Apache Commons Math Bibliothek brachte keinen Erfolg. Zwischendurch wurde Stadelmanns übersetzter k-Means Algorithmus verwendet, wobei dieser einen geringen Einfluss auf das gewünschte Ergebnis hatte und somit auch verworfen wurde. Nach der erfolgreichen Implementierung des Algorithmus wurde noch das GMM der Spinx4 Bibliothek ausprobiert, wobei diese sich nicht initialisieren lässt.

Hier muss erwähnt werden, dass die Sphinx4 Bibliothek am Anfang nicht zur Auswahl stand. Es wurde davon ausgegangen, dass eine Implementation in Android nicht möglich ist. Erst im Nachhinein wurde bemerkt, dass es durch verschiedene Umwege doch möglich ist. Die Komplikation mit der Sphinx4 Bibliothek wird in Kapitel 4.3.1 bei der Portierung in Android beschrieben. Die Ergebnisse mit den verschiedenen Bibliotheken können in Kapitel 5 betrachtet werden.

4 Umsetzung

4.1 Vorgehen

Die Umsetzung wird in mehrere kleine Schritte aufgeteilt, was ein strukturiertes Vorgehen ermöglichen soll.

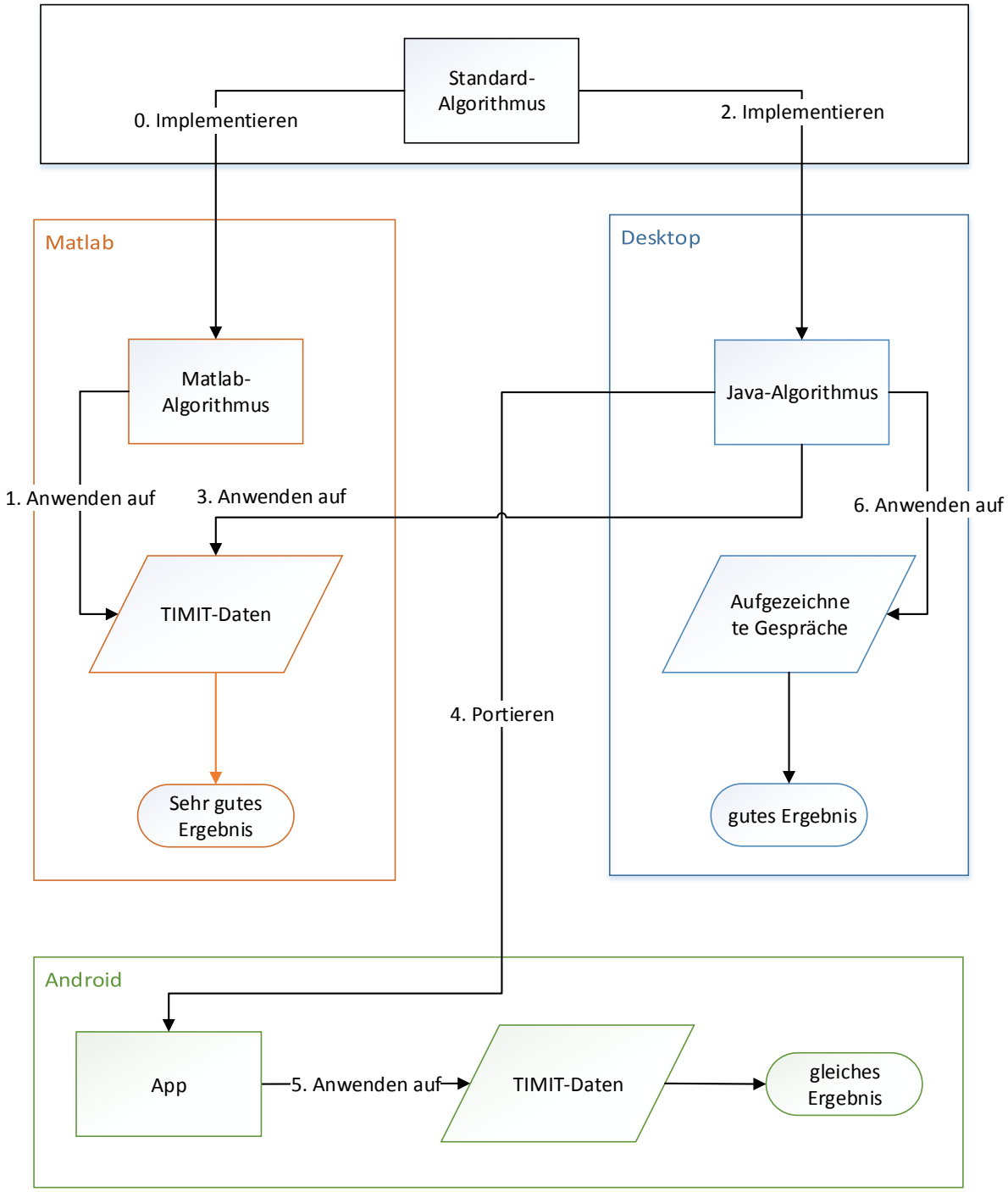

Abbildung 6: Vorgehen zur Umsetzung der Android-App

Die Blöcke in der Abbildung 6 repräsentieren verschiedenen Umgebungen. Der Standard-Algorithmus basiert auf das in Kapitel 2 vorgestellte Verfahren und kann in verschiedenen Umgebungen implementiert werden. Er wurde bereits im Vorfeld vom InIT für die Matlab-Umgebung implementiert. Ein weiterer Block ist die Desktop-Umgebung, für welche ein Java-Algorithmus geschrieben wird. Dieser wird dann in die Android-Umgebung portiert. Die Aufteilung zwischen einer Desktop- und Android-Umgebung führt zu einer effizienteren und schnelleren Implementierung, da die Desktop-Umgebung mehr Möglichkeit bietet.

In der untenstehenden Auflistung werden die Schritte kurz erklärt.

1. Zuerst wird der Matlab-Algorithmus mit den TIMIT-Daten getestet. Das Ergebnis sollte sehr gut sein, da in der Matlab-Umgebung alle benötigten Bibliotheken für die Sprechererkennung vorhanden sind. Dieses Ergebnis wird als Richtwert für den Java Algorithmus verwendet.
2. Anschliessend wird der Standard-Algorithmus in Java implementiert.
3. Im nächsten Schritt wird der Java-Algorithmus mittels der TIMIT-Daten getestet. Das Ergebnis der Sprechererkennungsrate sollte gleich wie beim Matlab-Algorithmus sein.
4. Falls der Java-Algorithmus funktioniert und das gewünschte Ergebnis erzielt, wird sie in die Android-Umgebung portiert.
5. Die Android-App wird auch mit den TIMIT-Daten getestet, wobei das gleiche Resultat wie in der Desktop-Umgebung herauskommen muss.
6. Als letztes wird in weiteren Experimente der Java-Algorithmus noch mit selbst aufgezeichneten Gesprächen getestet, um die Sprechererkennungsrate noch mit realen Bedingungen zu bestimmen. Das Ergebnis sollte ein gutes Resultat liefen.

4.2 Ablauf zum Testen der TIMIT-Daten

Für das Testen der TIMIT-Daten wird für den Matlab-Algorithmus, den Java-Algorithmus und somit auch in der Android Portierung der gleiche Ablauf verwendet.

Zur Überprüfung der Sprechererkennungsrate werden die TIMIT-Daten für jeden der 630 Sprecher in zwei Sets aufgeteilt. Im Training-Set befinden sich 8 von den 10 Audiodateien, welche mit einer ungefähren Durchschnittslänge von 3s eine Gesamtlänge von 24s zum Trainieren des GMM ausmachen. Die restlichen 2 Audiodateien machen analog 6s Aussagen für das Test-Set aus, mit welchem dann die Distanzmessung erfolgt.

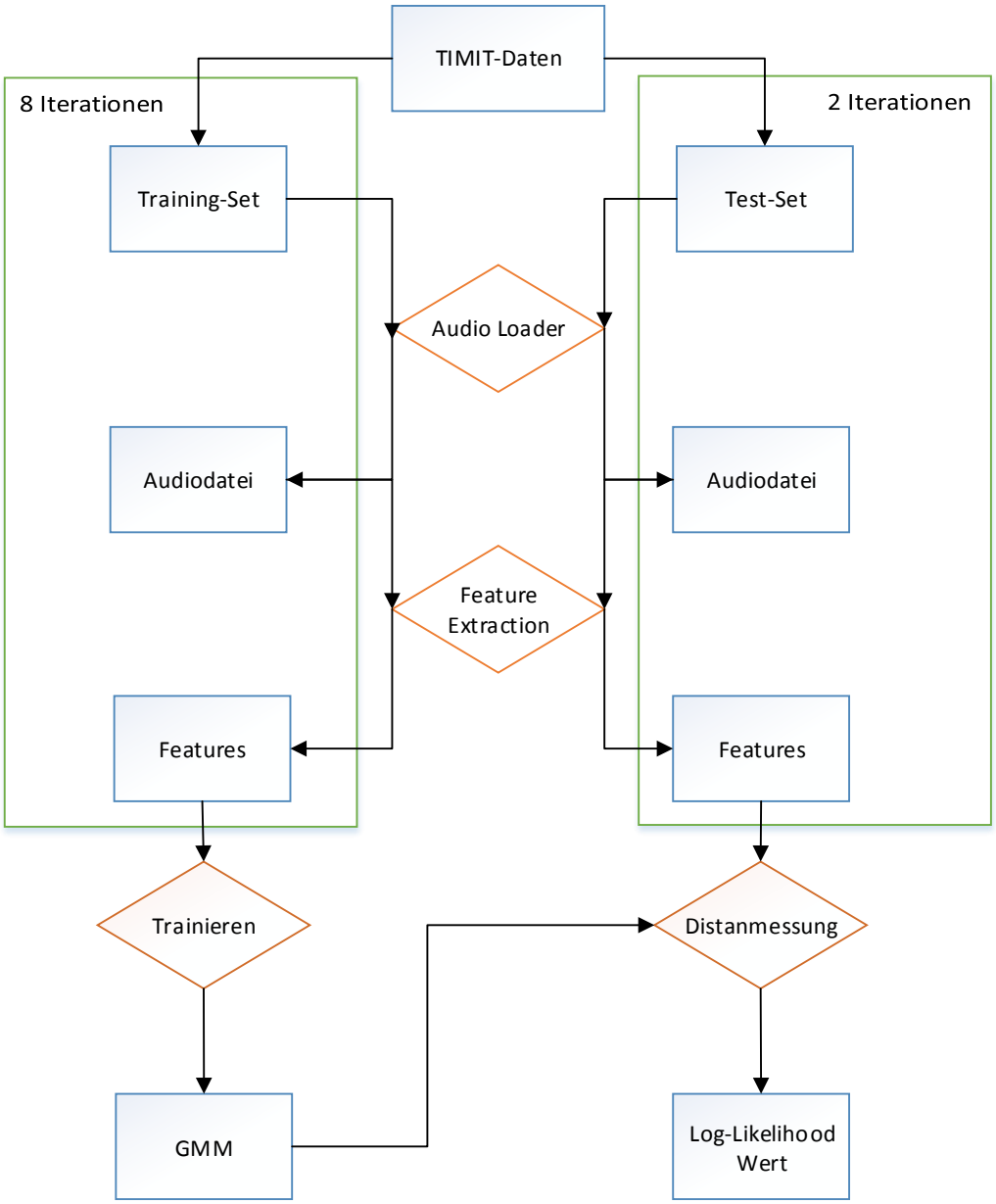

Abbildung 7: Ablauf zum Testen der TIMIT-Daten

Nachdem aus dem Training-Set für alle 630 Sprecher ein GMM erstellt worden ist und die Test-Features (Features, welche aus dem Test-Set extrahiert sind) für alle 630 Sprecher extrahiert sind, wird für die Berechnung der Sprechererkennungsrate wie in der folgenden Abbildung 8 vorgegangen.

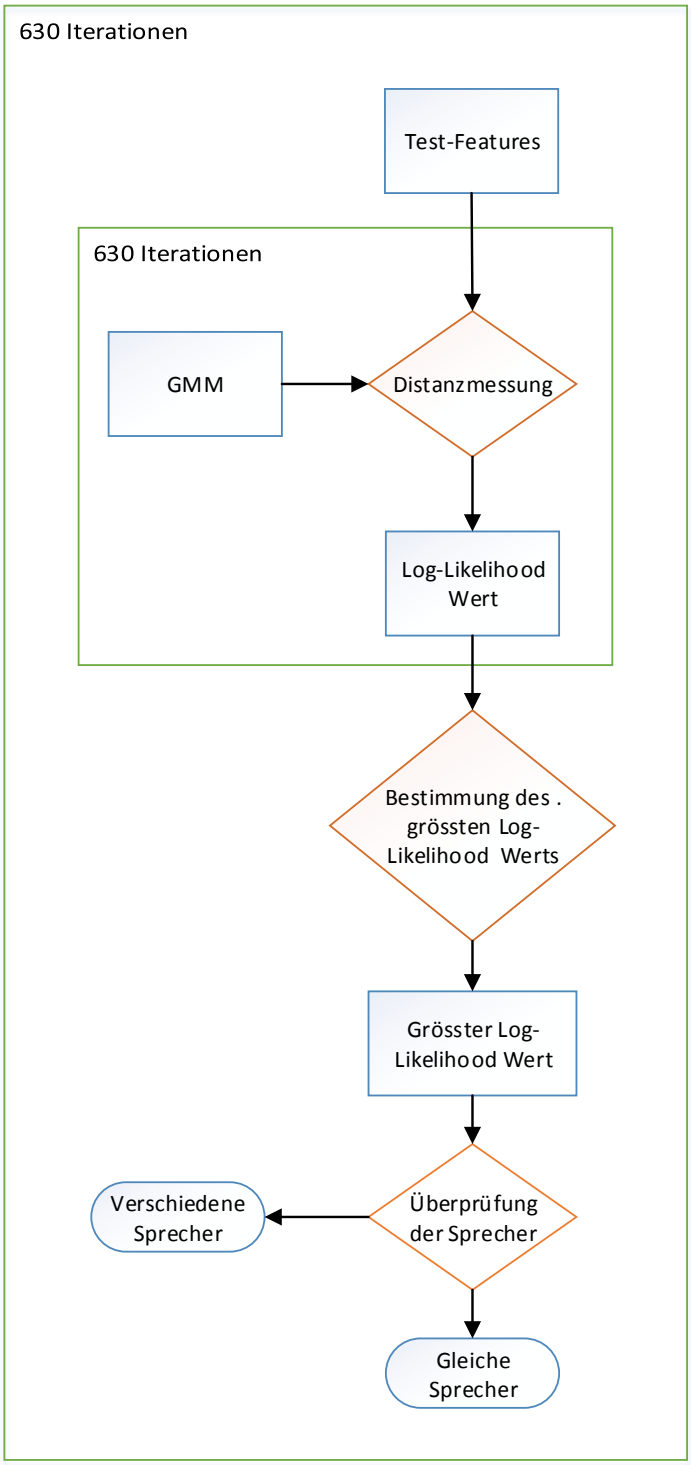

Abbildung 8: Berechnung der Sprechererkennungsrate

Zuerst werden die Test-Features eines Sprechers mit den GMMs aller Sprecher mittels der Distanzmessung verglichen. Daraus resultierten 630 Log-Likelihood Werte, aus denen anschliessend der grösste Log-Likelihood Wert bestimmt wird. Danach wird der Sprechername, aus welchem die Test-Features extrahiert sind, mit dem Sprechernamen verglichen, dessen GMM über die Distanzmessung im grössten Log-Likleihood Wert resultierte. Handelt es sich um den gleichen Sprechernamen, war die Sprechererkennung erfolgreich. Dieser Vorgang wird 630-mal für alle Test-Features wiederholt.

4.3 Implementierung des Java-Algorithmus

Beim Java-Algorithmus werden für die automatische Sprechererkennung verschieden Parameter konfiguriert, welche in der untenstehenden Tabelle ersichtlich sind. In der Kolone Kapitel wird zusätzlich angegeben, unter welchem Kapitel die Parameter und deren Werte beschrieben sind.

Preprocessing			
Prozess	Parameter	Wert	Kapitel
Sampling	Samplerate in Hz	16000	2.2.1
Highpass-Filter	Alpha	0.97	2.2.2
Framing	Framelänge in ms	20	2.2.3
	Frameüberscheidung in ms	10	

Feature Extraction			
Prozess	Parameter	Wert	Kapitel
Diskrete Fourier Transformation	Anzahl FFT Punkte	512	2.3.2
Mel Filterbank	Anzahl Filter	24	2.3.3
	Min. Frequenz in Hz	0	
	Max. Frequenz in Hz	7600	
Diskrete Cosinus Transformation	Anzahl MFCC	20	2.3.4

k-Means			
Prozess	Parameter	Wert	Kapitel
Cluster	Anzahl Cluster	32	2.4.3
	Anzahl Iterationen	100	

Modeling			
Prozess	Parameter	Wert	Kapitel
GMM	Anzahl Mixture	32	2.4
	Varianz Limit	0.01	2.4.2
EM-Algorithmus	Anzahl Iterationen	100	2.4.4

Recognition			
Prozess	Parameter	Wert	Kapitel
Distanzberechnung	Threshold	8.5	5.4

Tabelle 3: Konfiguration der Parameter zur Implementierung

4.3.1 Portierung in Android

Für die Portierung des Java-Algorithmus in Android wurde mit wenigen Ausnahmen der gleiche Code verwendet. Bei den Unterschieden handelt es sich um folgende Komponenten:

- Die Art und Weise wie Audiofiles geladen werden, jedoch nicht die Verarbeitung.
- Eine Activity-Klasse, anstatt einer Main-Methode.
- Gradle als Build-Management-Tool

Es ergibt sich aber noch eine Komplikation mit der Sphinx4 Bibliothek bezüglich der MFCC Feature Extraction Komponente, welche für die Desktopversion geschrieben ist. Als Alternative könnte eine schlankere, für die Android-Plattform gedachte Version, verwendet werden. Diese ist aber noch unter Entwicklung, wobei die Implementierung der MFCC Feature Extraciton noch inkomplett ist (N.Shmyrev, 2015). Es geht um das Problem, dass die Parameter für die Feature Extraction aus einer separaten XML-Datei geladen werden. Diese XML-Datei kann nicht direkt aus dem Android-Projekt geladen werden und befindet sich zurzeit als Zwischenlösung auf dem externen Speicher (SD-Karte) des Smartphone-Besitzers. Auf Anfrage auf der StackOverflow-Webseite wurde von einem CMU-Sphinx Mitentwickler auf die Möglichkeit hingewiesen, dass die XML-Datei in die Sphinx4 Bibliothek hinzugefügt werden kann. Diese Lösung wurde wegen mangelnder Zeit für diese Bachelorarbeit noch nicht umgesetzt.

4.4 Aufzeichnen von Gesprächen

Zum weiteren Testen des Java-Algorithmus werden zusätzlich zu den TIMIT-Daten noch eigene Gespräche aufgenommen. Diese werden mit dem Smartphone-Mikrophone aufgenommen, damit ähnliche Bedingungen wie beim realen Einsatz der Android-App herrschen. Die Gespräche werden mit einer selbst entwickelten App aufgenommen, dessen Aufzeichnungs-Algorithmus später auch in der Android-App verwendet wird.

Mit der Android SDK können jedoch aufgezeichnete Gespräche nicht im WAV-Dateiformat gespeichert werden. Die einzige Möglichkeit Audiodateien unkomprimiert abzulegen, ist als eine rohe Audiodatei ohne jegliche Header Informationen. Dazu wird wie bei den TIMIT-Daten das PCM Verfahren mit 16 Bits pro Sample verwendet. Anschliessend wird durch die externe Java-Bibliothek Simplesound die rohe Audiodatei in das WAV-Format umgewandelt.

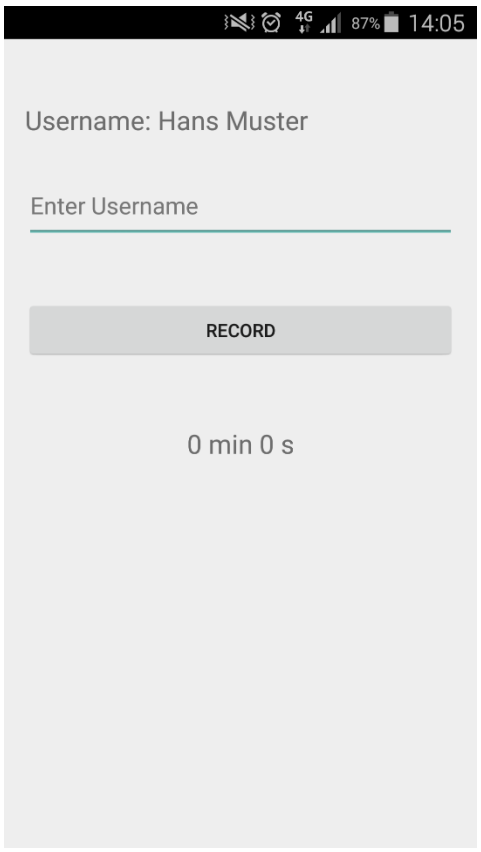

Abbildung 9: Android-App zur Aufzeichnung von Stimmen

Wie in der Abbildung 9 ersichtlich ist, kann er Benutzer seinen Namen eingeben, unter welchem dann die WAV-Datei abgespeichert wird. Als Hilfsmittel wird noch die gesprochene Zeitdauer angezeigt.

4.4.1 Editieren von Gesprächen

Für die Experimente müssen die aufgezeichneten Gespräche in beliebige Längen wie beispielsweise einer Sekunde oder noch kürzer geschnitten werden können. Dazu wird das FFmpeg Projekt eingesetzt, welche verschiedene Audio-Bearbeitungsmöglichkeiten per Kommandozeile anbietet. Wenn aber mehrere Tausend Audiodateien geschnitten werden müssen, ist FFmpeg alleine ineffizient. Aus diesem Grund wird die Apache Commons Exec Bibliothek verwendet, welche externe Prozesse mit Java ausführen kann. Dadurch können die FFmpeg Anweisungen in Java geschrieben werden, somit ist eine effiziente Bearbeitung möglich.

5 Ergebnisse

5.1 Umfeld

Zum Testen des Java-Algorithmus wird ein Samsung Notebook aus dem Jahre 2012 mit folgender Spezifikation verwendet:

Prozessor	Intel Core i5-3317U (1.7 GHz, 2 Kerne)
RAM	4GB
Betriebssystem	Windows 8.1

Tabelle 4: Spezifikationen zum Testen des Java-Algorithmus

Zum Testen der Android-App wird ein Samsung Galaxy S5 Smartphone mit folgenden Spezifikationen verwendet:

Prozessor	Qualcomm Snapdragon 801 (2.5 GHz, 4 Kerne)
RAM	2GB
Betriebssystem	Android 5.0

Tabelle 5: Spezifikationen zum Testen der Android-App

5.2 Das Testen des Matlab-Algorithmus

Die Sprechererkennungsrate des Matlab-Algorithmus beträgt für die 630 Sprecher der TIMIT-Daten 100%.

5.3 Das Testen des Java-Algorithmus

Da der Matlab-Algorithmus einen Richtwert von 100% liefert, wurde solange mit dem Java-Algorithmus getestet, bis auch dieser eine 100% Sprechererkennungsrate erzielte. Die Ergebnisse werden für eine bessere Übersicht nicht chronologisch sondern nach Komponenten aufgegliedert.

5.3.1 Ergebnisse der MFCC Feature Extraction Komponenten

Bibliothek	Ergebnis
CoMIRVA	83% Sprechererkennungsrate.
Sphinx4	100% Sprechererkennungsrate

Tabelle 6: Ergebnisse der MFCC Feature Extraction Komponenten

Evaluation
Die CoMIRVA führte nicht zum gewünschten Ergebnis, obwohl die Samplewerte, die Anzahl Feature Vektoren und andere Werte korrekt erschienen. Im nächsten Schritt wurden die Features der CoMIRVA Bibliothek anhand der externen Bibliothek JMatIO in Matlab-Datenformat abgespeichert und in Matlab geladen und getestet. Bereits schon bei 50 Sprecher beträgt die Sprechererkennungsrate 93%. Dieses Experiment zeigte, dass die Feature Extraction der CoMIRVA Bibliothek fehlerhaft implementiert sind. Sie wurde entsprechend durch die Sphinx4 Bibliothek ersetzt, welche eine 100% Sprechererkennungsrate ermöglicht.

5.3.2 Ergebnisse der k-Means Komponenten

Bibliothek	Ergebnis
CoMIRVA	Der k-Means Algorithmus ist nicht in der Lage mit einer Cluster-Anzahl von 32 zu terminieren, welches wahrscheinlich an einer schlechten Implementierung liegt.
Apache Commons Math	100% Sprechererkennungsrate.
sclib	Der k-Means von sclib wurde angewendet, jedoch nach erfolgreicher Implementierung der Apache Commons Math nicht weiter getestet.

Tabelle 7: Ergebnisse der k-Means Komponenten

Evaluation
Die k-Mean Komponente von der Apache Commons Math wird im Java Algorithmus eingesetzt.

5.3.3 Ergebnisse der GMM Komponenten

Bibliothek	Ergebnis
CoMIRVA	Für die Berechnung der Kovarianzmatrix lässt sich das Inverse einer Matrix nicht bilden. Obwohl die Möglichkeit zur Berechnung der Hauptdiagonale einer Kovarianzmatrix besteht, wird wahrscheinlich zuerst versucht die volle 19-dimensionale Kovrianzmatrix zu berechnen, um anschliessend deren Hauptdiagonale zu erhalten.
Apache Commons Math	SingularMatrixException wird bei der Berechnung der Kovarianzmatrix ausgegeben. Hier besteht die Möglichkeit nicht, nur die Hauptdiagonale berechnen zu lassen.
sclib	100% Sprechererkennungsrate
Sphinx4	Das GMM der Sphinx4 Bibliothek lässt sich nicht implementieren. Es verlangt die Kovarianzmatrix als Initialisierungsparameter, aus welcher es sich dann selbst die Varianz ausrechnet.

Evaluation

Die sclib ist die einzige Bibliothek, welche hier zum Erfolg führt. Sie braucht die Kovarianzmatrix nicht, da sie für die Varianzen direkt die Hauptdiagonale berechnet.

5.4 Das Testen von aufgezeichneten Gesprächen

Nachdem die Sprechererkennungsrate des Java-Algorithmus bezüglich den TIMIT-Daten 100% beträgt, wird der Algorithmus noch mit selbst aufgezeichneten Gesprächen getestet. Als Grundlage für dieses Experiment wird von einem Buch der gleiche Textausschnitt von mehreren Personen gelesen.

Anzahl Personen	5
Buch	Die Kunst des klaren Denkens: 52 Denkfehler, die Sie besser anderen überlassen, von Rolf Dobelli
Dauer der Aufnahme	5min pro Person
Ort	Hörsaal TE 414 in der ZHAW
Diktiergerät	Ein Smartphone-Mikrophon, welches auf einem Tisch gegenüber der Person liegt.
Qualität der Aufnahme	Hoch, keine Nebengeräusche oder andere Störfaktoren

Tabelle 8: Angaben zur Test-Umgebung

Im Vorfeld wurden für die Sprechererkennungsrate einige Thresholds verglichen, wobei Thresholds mit den Werten 8.5 und 8.0 die besten Resultate lieferten. Aus diesem Grund werden für dieses Experiment beide Thresholds verwendet und anschliessend derjenige Threshold für den Java-Algorithmus ausgewählt, welcher das bessere Resultat liefert.

Zum Testen werden die ersten 2min und 4min der Aufnahme von einer Person für das Trainieren des GMM verwendet. Für alle 5 Personen wird die letzte Minute für das Test-Set gebraucht, wobei die Aufnahme in Audiodateien mit verschiedenen Sprechdauern geschnitten werden.

Die Sprechererkennungsrate ist am Anfang bei 1.00 und sinkt bei jeder Falscherkennung.

- Falls der Log-Likelihood Wert über dem Threshold liegt, es sich aber um einen anderen Sprecher als im Training-Set handelt.
- Falls der Log-Likelihood Wert unter dem Threshold liegt, es sich aber eigentlich um den gleichen Sprecher wie im Training-Set handelt.

1min Test-Set		2min Training-Set		4min Training-Set	
Anzahl Audiodateien	Sprechdauer der Audiodatei	Threshold 8.5	Threshold 8.0	Threshold 8.5	Threshold 8.0
12 * 5P	5s	1.00	0.98	1.00	0.98
30 * 5P	2s	0.93	0.93	0.95	0.94
60 * 5P	1s	0.87	0.87	0.89	0.89
120 * 5P	0.5s	0.83	0.82	0.84	0.84
600 * 5P	0.1s	0.72	0.70	0.72	0.71
1200 * 5P	0.05s	0.66	0.63	0.66	0.64

Tabelle 9: Ergebnisse der Sprechererkennungsrate des Java-Algorithmus

Aus der obigen Tabelle 9 sind folgende Informationen ersichtlich

- Eine Sprechdauer von 5s mit einem 2min oder 4min langen trainierten GMM liefert eine 100% Sprechererkennungsrate bezüglich eines Thresholds von 8.5.
- Je kürzer die Sprecherdauer, desto schlechter die Sprechererkennungsrate.
- Ein Threshold mit einem Wert von 8.5 liefert im Schnitt die bessere Sprechererkennungsraten als ein Threshold mit einem Wert von 8.0.

Für die Android-App wird daher eine Window-Länge von 2s Sprechdauer genommen, für welche die Log-Likelihood Werte berechnet werden. Eine Sprechdauer grösser als 2s liefert eine bessere Sprechererkennungsrate, wobei aber auch die Wahrscheinlichkeit für einen Sprecherwechsel steigt und somit die Sprechererkennungsrate wieder schlechter wird. Eine Sprechdauer von weniger als 2s liefert eine relativ schlechte Sprechererkennungsrate. Die Sprechdauer von 2s wird als das optimale Verhältnis zwischen Sprecherwechsel und Sprechererkennungsrate betrachtet.

5.5 Der Sliding-Window Test

Im vorherigem Test in Kapitel 5.4 wurde der Log-Likelihood Wert beispielsweise anhand einer Sprechdauer von einer Sekunde berechnet. Da der Java-Algorithmus alle 10ms ein Frame generiert, beträgt das für 1000ms eine Anzahl von 101 Frames (+ 1 wegen der Überschneidung). Somit wurde der Log-Likelihood Wert durch 101 Frames gebildet.

Als letzter Test wird mit dem Sliding-Window nun der Log-Likelihood für jeden einzelnen Frame berechnet. Dafür wird eine Sliding-Window-Länge von 1s und 2s verwendet und eine Sliding-Window-Schrittweite von 1 Frame. Mit diesem Verfahren wird dem Window iterativ alle 10ms ein neues Frame hinzugefügt, das alte Frame aus dem Window entfernt und anschliessend die Wahrscheinlichkeit berechnet. Es könnte auch ohne das Sliding-Window die Wahrscheinlichkeit für jeden einzelnen Frame berechnet werden. Die Wahrscheinlichkeit wäre jedoch dann nicht aussagekräftig, da sich aus einem einzelnen Frame nicht genügend Features gewinnen lassen.

Für dieses Experiment wurden aus den aufgezeichneten Gesprächen von einer Person 4min Aufnahme verwendet, um ein Sprechermodell zu trainieren. Die restliche Minute der Person plus die 5min der anderen 4 Personen werden für den Sliding-Window verwendet, welches insgesamt 21min Sprechmaterial ausmacht. Aus diesen werden ungefähr 127‘000 Frames generiert.

In den Diagrammen werden noch zwei Hilfslinien eingezeichnet. Die Groundtruth gibt den Mittelwert der Log-Likelihood Werte an, welche ohne Sliding-Window berechnet wurde. Der aufgerundete Mittelwert beträgt 10 für die erste Minute und 5 für die restlichen 20min.

Für eine bessere Übersicht werden die ersten 10‘000 Log-Likelihood Werte eigene Diagramme erstellt.

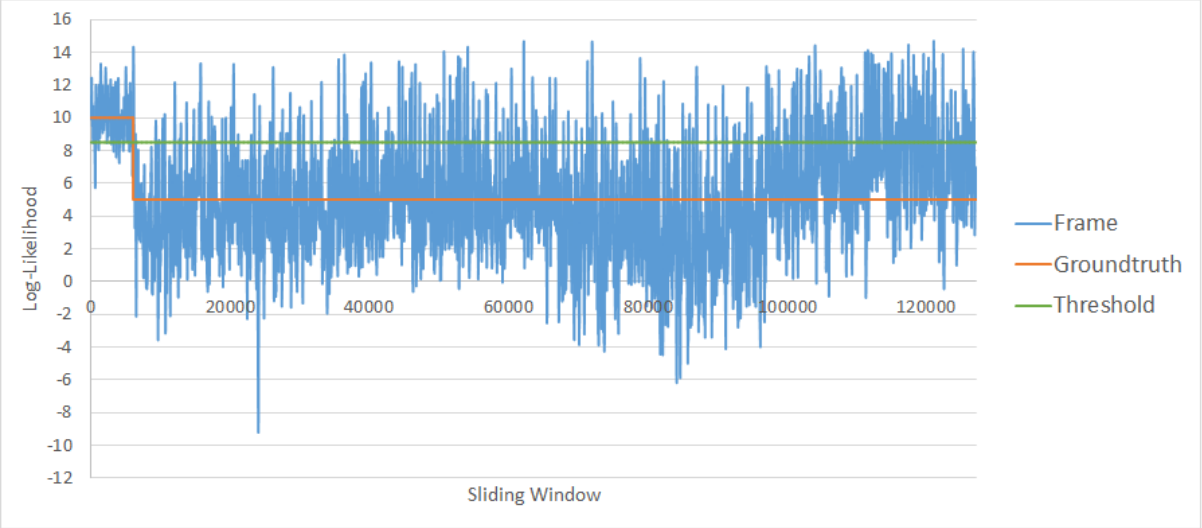

Abbildung 10: Sliding Window-Länge von 1s und 127‘000 Log-Likelihood Werten

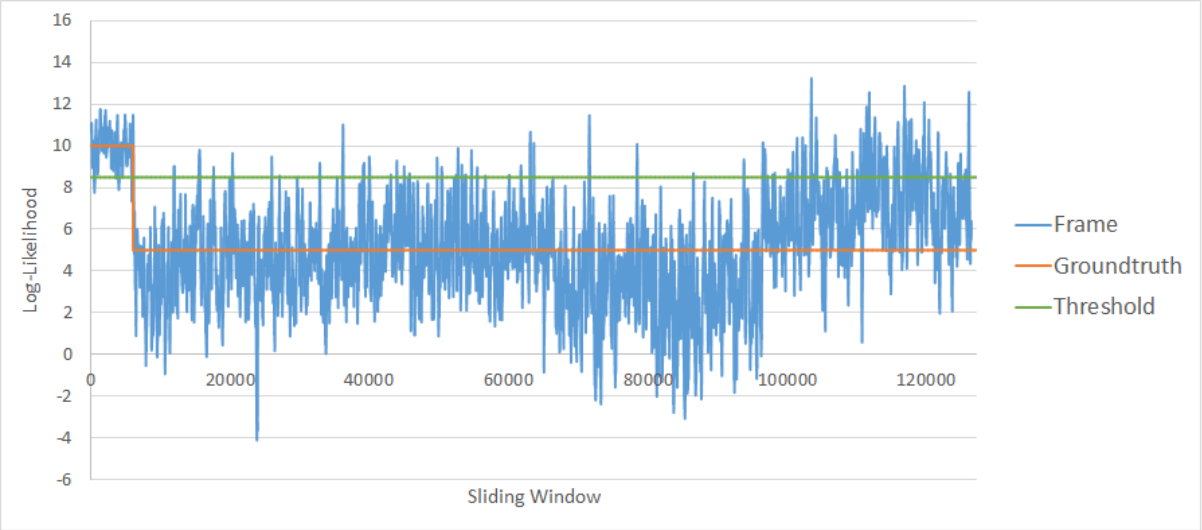

Abbildung 11: Sliding-Window-Länge von 2s und 127‘000 Log-Likelihood Werten

Wie aus den Diagrammen ersichtlich ist, liefert eine Sliding-Window-Länge von 2s bessere Resultate. Die Log-Likelihood Werte der anderen 4 Sprecher liegen vermehrt unterhalb des Thresholds. Dass eine Window-Länge von 2s bessere Ergebnisse liefert als eine Window-Länge von 1s, ist auch aus dem vorherigen Test in Kapitel 5.4 bereits bekannt. In diesen Abbildungen wird dies jedoch mit mehr Werten grafisch stärker hervorgehoben.

Abbildung 12: Sliding-Window-Länge von 1s und 10'000 Log-Likelihood Werten

Abbildung 13: Sliding-Window-Länge von 2s und 10‘000 Log-Likelihood Werten

Wie aus den Abbildungen ersichtlich ist, gib es Schwankungen bei den Log-Likelihood Werten. Die Ursache für die Schwankungen liegt nach Stadelmanns Aussage stark vereinfacht daran, dass es schwierig und bisweilen nicht eindeutig möglich ist, den Sprecher mit diesen Features und Verfahren eindeutig automatisch zu identifizieren.

Dabei nehmen die Schwankungen bei einer Sliding-Window-Länge von 2s gegenüber 1s ab, da aus den Frames mehr Features extrahiert werden kann.

6 Fazit

6.1 Zusammenfassung

In dieser Bachelorarbeit wurde ein Java-Algorithmus für die automatische Sprechererkennung entwickelt, welche eine 100% Sprechererkennungsrate bezüglich den 630 Sprechern der TIMIT-Daten liefert. Für die Implementierung wurden verschiedene Bibliotheken getestet, wobei schlussendlich die Kombination aus zwei Bibliotheken und von C++ in Java übersetzte Funktionen aus der sclib Bibliothek zum Erfolg führten. Für das Preprocessing des Sprachsignals und dessen anschliessende Feature Extraction wird die Sphinx4 Bibliothek verwendet. Die Features werden im nächsten Prozess vom k-Means Algorithmus aus der Apache Commons Math Bibliothek in 32 Cluster unterteilt. Aus diesen Clustern werden die Parameter Mittelwert, Varianz und Gewicht für das Initialisieren eines GMM berechnet. Für die Implementierung des GMM als Sprechermodell werden Funktionen aus der sclib Bibliothek verwendet. Dieser enthält für das Trainieren eines GMM einen EM-Algorithmus, mit welchem die Parameter iterativ optimiert werden. Zudem verfügt es für das Identifizieren von Stimmen eine Log-Likelihood Funktion, mit welcher die Wahrscheinlichkeit von neuen Features zum bestehenden GMM berechnet werden kann.

Der Java Algorithmus wurde in Android portiert, wobei auch die Android-App eine Sprechererkennungsrate von 100% bezüglich den 630 Sprecher der TIMIT-Daten erzielte. Abgesehen von den Android spezifischen Komponenten wie beispielsweise die Art und Weise wie eine Aufnahme geladen wird, handelt es sich um den gleichen Java-Algorithmus.

In verschiedenen Experimenten wurde im nächsten Schritt die Sprechererkennungsrate bezüglich selbst aufgezeichneten Gesprächen getestet. Dafür wurde zusätzlich eine Android-App geschrieben, welche Gespräche unkomprimiert als WAV-Dateien speichert. Es wurde von 5 Personen aus einem Buch ein 5min langer Textausschnitt gelesen. Diese wurden als Grundlage für die Experimente verwendet.

Mit den ersten 2min und 4min der Aufnahme einer Person wurde ein GMM trainiert. Für das Test-Set wurde für jede der 5 Personen aus der letzten Minute der Aufnahme Audiodateien von einer Sprechdauer von 5s, 2s, 1s, 0.5, 0,1s, 0.05s geschnitten. Danach wurde aus jeder Audiodatei die Features extrahiert und deren Log-Likelihood Wert der zum GMM berechnet. Die Werte aller Log-Likelihood wurden mit einem Threshold verglichen und überprüft, ob der richtige Sprecher erkannt wurde. Das beste Ergebnis mit einer Sprechererkennungsrate von 100% liefert eine Sprechdauer von 5s mit einem 2min oder 4min langen trainierten GMM. Die Wahrscheinlichkeit für einen Sprecherwechsel in 5s ist aber relativ hoch, sodass sich 2s besser eignen als Auswahl. Die Sprechererkennungsrate von einer Sprechdauer von 2s liegt bei 95%, falls das GMM mit 4min lang trainiert ist.
Für die Experimente wurden zwei verschiedene Thresholds verwendet, um herauszufinden, welcher Threshold die bessere Sprechererkennungsrate erzielt. Ein Threshold von 8.5 liefert im Schnitt eine bessere Sprechererkennungsrate als ein Threshold von 8.0. Aus diesen Gründen wird für die Android-App ein Threshold von 8.5 verwendet und alle 2s aus einem Gespräch die Log-Likelihood Werte berechnet.

In einem weiteren Experiment wurde mit dem Sliding-Window für jedes einzelne Frame die Log-Likelihood bestimmt. Dafür wurde mit einer 4min langen Aufnahme einer Person ein GMM trainiert. Die restliche Minute wurde mit der gesamten Aufnahme der anderen 4 Personen für das Sliding-Window verwendet, welches eine Gesamtdauer von 21min ausmacht. Für die Sliding-Window-Länge wurde 1s und 2s verwendet und eine Sliding-Window-Schrittweite von 1 Frame. Die Log-Likelihood Werte wurden in Diagrammen dargestellt, welche grafisch die Unterschiede zwischen einer Window-Länge von 1s und 2s hervorheben.

6.2 Taktischer Ausblick

Die Android-App wurde erfolgreich bezüglich der Sprechererkennungsrate getestet, für eine einsatzbereite Android-App fehlen jedoch noch einige Implementationen. In den nächsten Kapiteln werden die kurzfristigen Folgearbeiten aufgezeigt.

6.2.1 Die Implementierung des Silence-Filter

Für die richtige Berechnung des Redeanteils muss die Stille aus einem Gespräch entfernt werden. Falls in einer Zeitdauer von 6s der Sprecher des trainiertem GMM 2s lang redet und 2s andere Sprecher, sollte ein Redeanteil von 50% resultieren. Da aber zusätzlich noch 2s Stille in den 6s existieren, werden deren Log-Likelihood Werte auch als andere Sprecher erfasst. Daraus ergibt sich fälschlicherweise ein Redeanteil von 33%.

Für die Implementierung des Silence-Filters können, wie in den Grundlagen in Kapitel 2.2.2 beschrieben, die MFCC Feature Vektoren als Hilfe verwendet werden. Der erste Koeffizient im Feature Vektor enthält die Tonhöhe für den Frequenzbereich, aus welchem er extrahiert wurde. In verschiedenen Tests muss dann bestimmt werden, ab welcher Tonhöhe Stille definiert ist. Feature Vektoren, welche die bestimmte Tonhöhe nicht erreichen, können somit verworfen werden.

6.2.2 Portierung der XML-Konfigurationsdatei in Android

Wie schon in Kapitel 4.3.1 bei der Portierung erwähnt, kann die XML-Datei mit den Parameter-Konfigurationen zurzeit nicht aus der Android-App geladen werden. Diese befindet sich als Zwischenlösung auf der SD-Karte des Smartphones. Ein Lösungsansatz wurde im gleichen Kapitel angegeben.

6.2.3 Die Implementierung einer GUI

Da der Schwerpunkt der Bachelorarbeit auf der Implementierung eines Java-Algorithmus zur automatischen Sprechererkennung liegt, wurde eine GUI nicht implementiert. Die GUI kann entweder neu geschrieben werden oder es kann eine bereits bestehende Umsetzung verwendet werden, welche in einer vorherigen Bachelorarbeit am InIT geschrieben wurde.

6.3 Strategischer Ausblick

Die Android-App liefert eine sehr gute Sprechererkennungsrate für Aufnahmen in hoher Qualität. Für langfristige Ziele, wie zum Beispiel der reale Einsatz in Vorlesungen, müssen jedoch noch weitere Experimente durchgeführt werden. Es geht vor allem um die Frage, wie sich der bestehende Java-Algorithmus mit Aufnahmen, welche eine schlechtere Qualität aufweisen, verhält.

- Wie stark ist der Einfluss der Distanz zwischen des Benutzer und dem Smartphone auf die Sprechererkennungsrate? Eventuell muss somit eine maximale Distanz festgesetzt werden. Eine einfachere Lösung wäre jedoch das Tragen eines Bluetooth-Headsets.
- Welche Auswirkungen haben Nebengeräusche auf die Sprechererkennungsrate, wie zum Beispiel das gelegentliche Flüstern von Studenten untereinander während einer Vorlesung oder sonstige Störfaktoren?

Es müssen verschiedene Szenarien getestet werden, um möglichst viele Informationen zu erhalten. Aus diesen können dann entsprechende Anpassungen am Java-Algorithmus vorgenommen werden, um auch eine hohe Sprechererkennungsrate bei schlechten Aufnahmebedingungen zu erhalten.

7 Anhang

7.1 Beschreibung der elektronischen Daten

Alle Ergebnisse und entwickelten Anwendungen sind auf einer CD enthalten und werden mit der Bachelorarbeit abgegeben. Zudem sind auch die aufgenommenen Gespräche beigelegt.

7.1.1 Ergebnisse der Experimente

Folgende Excel-Tabellen sind erstellet worden:

- Apache_kMeans_Clustering.xlsx
- CoMIRVA_MFCC.xlsx
- CoMIRVA_Sample.xlsx
- Log-Likelihood Konvergenz.xlsx
- Mean_Variance.xlsx
- Sliding_Window_Test.xlsx
- Sprechererkennungsrate.xlsx

7.1.2 Entwickelte Anwendungen

Folgende Anwendungen sind entwickelt worden:

- Talkalyzer Java-Algorithmus
- Talkalyzer Android-App
- AudioRecorder zum Aufnehmen von Gesprächen
- AudioEditor zum Schneiden von Audiodateien.
- Sph2pipe Batch-Datei zum Konvertieren von Audiodateien.

7.2 Aufgabenstellung Bachelorarbeit Besadi

Bachelor-Arbeit
Institut für angewandte Informationstechnologie

Betreuung:	Dr. Thilo Stadelmann Dr. Mark Cieliebak	BA-Student:	Arash Besadi
Industriepartner:	-		
Ausgabetermin:	Montag 16. Februar 2015	Abgabetermin:	Freitag 5. Juni 2015; 11:30Uhr
Anzahl Credits:	12 bzw. 360h	Präsentation:	Form und Zeitpunkt folgen später

Thema: Entwicklung einer Android-App zur Erkennung von Redeanteilen im Unterricht

1. Einleitung und Zielsetzung

"Talkalyzer" steht für eine Android-App, die live via Mikrofon erkennt, wer wieviel redet. Damit soll es z.B. im Unterricht möglich sein zu erkennen, wie hoch der Interaktionsgrad einer Lehrveranstaltung ist. Am InIT wurde dazu in einem Forschungsprojekt und darauf folgenden Bachelorarbeiten bereits eine einfache Architektur und GUI für eine Mobil-App entwickelt; state-of-the-Art Verfahren, welche die Besonderheiten der Stimme eines Menschen in Echtzeit extrahieren, wurden in Matlab implementiert und getestet.

Im Rahmen dieser Bachelorarbeit entwickeln Sie den App-Prototyp weiter, indem Sie durch systematisches Softwareengineering die existierenden Verfahren in die App einbinden. Hierzu erweitern Sie ggf. den bestehenden Architekturvorschlag und stellen durch eine Serie geeigneter Experimente die Leistungsfähigkeit der App fest.

Das Ziel ist eine anwendbare, vorzeigbare Applikation.

2. Aufgabenstellung

a) Einarbeiten in die bestehende Applikation
   a. Kennenlernen der bestehenden Android App [1]
   b. Kennenlernen des Hintergrunds automatischer Sprechererkennung [2, Kapitel 2]
b) Übertragen der Sprechererkennungs-Verfahren aus Matlab nach Android/Java
   a. Existierender Code in Matlab wird übergeben
   b. Verwenden Sie geeignete Bibliotheken
c) Sicherstellung einer erweiterbaren und testbaren Softwarearchitektur der App
   a. Die Software ist einfach zu warten, und die einzelnen Komponenten der Software sind sauber getrennt
   b. Es gibt ein Interface für den Datenaustausch zwischen der Mobile-App und der Analyse-Komponente
   c. Die Analyse-Komponente oder Teile davon können einfach ausgetauscht werden
d) Durchführen und dokumentieren systematischer Experimente zur Bestimmung der Leistungsfähigkeit der App auf bekannten Daten (synthetische Bedingungen)
   a. Verwendung der TIMIT Daten und des experimentellen Setups aus [3]
   b. Erzeugen möglichst identischer Resultate mit bestehendem Code in Matlab und in der App

1

e) Durchführen und dokumentieren systematischer Experimente zur Bestimmung der Leistungsfähigkeit der App auf realistischen Daten (z.B. aufgezeichneten Lehrveranstaltungen)
   a. Erstellung eines Corpus an Daten inkl. „Ground Truth" (gelabelte Daten)
      i. Bestimmung einer geeigneten Corpusgrösse
      ii. Bestimmung von Faktoren, welche die Erkennungsleistung der App beeinträchtigen können (z.B. Abstand und Richtung des Mikrofons, Raumgrösse etc.) und Berücksichtigung dieser Varianz in den zu sammelnden Daten
   b. Evaluation anhand dieser neuen Daten in der App und in Matlab

3. Bericht/Präsentation/Randbedingungen

- Sie übernehmen die Leitung des Projektes. Sie organisieren, leiten und treffen alle nötigen Massnahmen, um das Projekt zu einem erfolgreichen Abschluss zu führen.
- In der Regel findet eine wöchentliche Besprechung statt. Regelmässige Agendapunkte sollen sein „diese Woche gemacht", „für nächste Woche geplant", „Probleme".
- Über die Bachelorarbeit ist ein Bericht zu schreiben. Der Bericht ist in 3-facher Ausführung (gebundene Papierversion und elektronisch) termingerecht abzugeben. Es gelten die üblichen Dokumente zum Ablauf und zur Form (Leitfaden_PABA.pdf, Zitierleitfaden, etc.).

4. Bewertungskriterien, Gewichtung:
Projektverlauf, Leistung, Arbeitsverhalten ca. ¼ ; Qualität der Ergebnisse ca. ¼ ; Form und Inhalt des Berichts und der Präsentation ca. ½

5. Literaturverzeichnis
[1] Kündig, "Mobile-App zur automatischen Sprecher-Erkennung", 2014
[2] Stadelmann, "Voice Modeling Methods for Automatic Speaker Recognition", 2010
[3] Reynolds, "Speaker Identification and Verification using Gaussian Mixture Speaker Models", 1995

Weitere Fachliteratur zum Thema wird von den Betreuern zur Verfügung gestellt bzw. vom Studierenden recherchiert.

8 Verzeichnisse

8.1 Abbildungsverzeichnis

8.2 Tabellenverzeichnis

8.3 Literaturverzeichnis

A.Reynolds et al., 1995. Robust text-independent speaker identification using Gaussian mixture speaker models. In: IEEE Transactions on Speech and Audio Processing. IEEE, p. 72–83.

A.Ziech, 2014. Ingenieur. [Online]. URL: http://www.ingenieur.de/Themen/Software/Software-Terroristen-an-Stimme-erkennen [20 05 2015].

Binksternet, 2009. Wikipedia. [Online]. URL: http://en.wikipedia.org/wiki/Sampling_(signal_processing)#/media/File:Signal_Sampling.png [08 05 2015].

CMUSphinx, 2015. Class Preemphasizer. [Online]. URL: http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/frontend/filter/Preemphasizer.html [01 06 2015].

CMUSphinx, 2015. Class RaisedCosineWindower. [Online]. URL: http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/frontend/window/RaisedCosineWindower.html [09 05 2015].

CMUSphinx, 2015. CMUSphinx. [Online]. URL: http://cmusphinx.sourceforge.net/ [25 05 2015].

CMUSphinx, 2015. Deltas Feature Extractor. [Online]. URL: http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/frontend/feature/DeltasFeatureExtractor.html [29 05 2015].

J.Lyons, 2012. practical cryptography. [Online]. URL: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ [11 05 2015].

J.Stampfli, 2014. Talkalyzer: Neue Algorithmen für automatische. Winterthur: ZHAW.

J.Wendt, 2015. Golem. [Online]. URL: http://www.golem.de/news/voiceprint-stimmenerkennung-ist-die-neue-gesichtserkennung-1504-113801.html [08 05 2105].

M.Becker et al., 2010. Spiegel. [Online]. URL:
http://www.spiegel.de/wissenschaft/mensch/hirnforschung-schon-kleinkinder-erkennen-gefuehle-in-der-stimme-a-685506.html
[23 05 2015].

N.Shmyrev, 2015. sourceforge.net. [Online]. URL:
http://sourceforge.net/p/cmusphinx/discussion/help/thread/36a5ac7b/
[25 05 2015].

T. Stadelmann zit. n. T.Mitchell, 1997. Machine Learning. In: USA: McGraw Hill.

T.Stadelmann 2010, zit. n. R. Vergin et al., 1995. Pre-Emphasis and Speech Recognition. In: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering. Montreal: IEEE, p. 1062–1065.

T.Stadelmann, 2010. Voice Modeling Methods for Automatic Speaker Recognition. Marburg: Philipps-Universität Marburg.

Wikipedia: Kovarianzmatrix, 2015. Wikipedia: Kovarianzmatrix. [Online]. URL:
http://de.wikipedia.org/wiki/Kovarianzmatrix
[24 05 2015].

Wikipedia: Mehrdimensionale Normalverteilung, 2015. Wikipedia: Mehrdimensionale Normalverteilung. [Online]. URL:
http://de.wikipedia.org/wiki/Mehrdimensionale_Normalverteilung
[24 10 2015].

Wikipedia: Phonem, 2015. Wikipedia: Phonem. [Online]. URL:
http://de.wikipedia.org/wiki/Phonem
[24 05 2015].

Wikipedia, 2015. Phon (Linguistik). [Online]. URL:
http://de.wikipedia.org/wiki/Phon_(Linguistik)
[08 05 2015].

X.Anguera et al., 2010. Speaker Diarization: A Review of Recent Research. [Online]. URL:
http://www.eurecom.fr/en/publication/3152/download/mm-publi-3152.pdf
[08 05 2015].