paluno
The Ruhr Institute for Software Technology
Institut für Informatik und Wirtschaftsinformatik
Universität Duisburg-Essen

Bachelorprojekt-Bericht

Umwandlung von Radiosendungen in Textform

Bastian Lang
3089941

Essen, 25.05.2022

Betreuung: Nils Schwenzfeier

Studiengang: Angewandte Informatik – Systems Engineering

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, am 25.05.2022

Zusammenfassung

In diesem Bericht wird das RadioSummarizer-Projekt beschrieben, mit welchem Nachrichtensendungen in Text umgewandelt werden können. Mithilfe dieser Anwendung ist es auch möglich, Audioaufnahmen auf Nachrichtensendungen zuzuschneiden. Im Bericht wird dabei auf die Anwendung des Programms und die internen Vorgänge eingegangen. Abschließend wird besprochen welche Umstände die Ausgabe der Anwendung negativ beeinträchtigen können.

Abstract

Description of the RadioSummarizer project, that allows converting news broadcasts on the radio into text.
Additionally, the application allows trimming audio files to just the part that contains the broadcast. This report describes the use of the application and the internal processes. In the end, it will also discuss problems that can negatively affect the quality of the output.

Abbildungsverzeichnis

Abbildung 1: Visualisierung des Systems _____ 14

Abbildung 2: Beispielskorrelation _____ 15

Tabellenverzeichnis

Tabelle 1 Satzzeichen Scores _____ 10

Inhaltsverzeichnis

Eidesstattliche Erklärung ..... **II**

Zusammenfassung ..... **III**

Abstract ..... **III**

Abbildungsverzeichnis ..... **IV**

Tabellenverzeichnis ..... **V**

Inhaltsverzeichnis ..... **VI**

1. Einleitung **7**

2. Definitionen **8**

2.1 Speaker diarization 8

2.2 Speech-to-Text 8

3. Datensatz und Bibliotheken **9**

3.1 Datensatz 9

3.2 Verwendete Bibliotheken 9

4. Anwendung des Programms **12**

5. System Design **14**

5.1 Übersicht 14

5.2 Detaillierter Ablauf 14

6. Bewertung **18**

6.1 Schwächen 18

6.2 Fazit 19

Literatur **21**

1. Einleitung

Das Schreiben von Zeitungsartikeln setzt voraus, dass man bereits Informationen aus anderen Quellen zusammengetragen hat. Als eine denkbare Quelle können Nachrichtensendungen aus dem Radio herangezogen werden. Es ist aber nicht praktikabel, jede Nachrichtensendung auf den verschiedenen Sendern live zu verfolgen.
Eine mögliche Lösung ist es, die Sendungen aufzuzeichnen und zu einem späteren Zeitpunkt wiederzugeben. Dieses Vorgehen beansprucht aber einen großen Zeitaufwand, da man beim Abhören an die Länge der Audiodatei gebunden ist. All dies kann dazu führen, dass man zur Einhaltung der Abgabefrist eines Artikels, nicht alle nötigen Informationen zusammenbekommen hat.
Zwar kann man die Audiodateien in schnellerer Geschwindigkeit abspielen, dennoch hat man nicht die Möglichkeit, die Audiodateien nach Themen zu filtern.
Eine große Unterstützung für Autoren wäre es daher, wenn ein Programm die Nachrichtensendungen aufzeichnet und in einen lesbaren Text transformiert. Im idealen Fall soll der ausgegebene Text automatisch in Themenbereiche untergliedert werden.
So wären die Autoren in der Lage den Inhalt der unterschiedlichen Nachrichtensendungen im eigenem Tempo zu filtern.
Ziel dieses Projektes ist es, einen Teil zur Lösung dieses Problems beizutragen. Hierbei wird der Prozess bis hin zur Erstellung eines Textes umgesetzt, bei welchem zusätzlich Sprecherwechsel gekennzeichnet werden. Sprecherwechsel werden eingefügt, da diese auch auf Themenwechsel hindeuten können.
Zur Umsetzung der Aufgabe wird auf Vosk[1] zurückgegriffen, welches eine bereits bestehende Speech-to-Text Lösung ist.
Zur Erkennung der Sprecherwechsel wird speaker diarization angewendet.
In Kapitel 2 des Berichts werden die Begriffe Speech-to-Text und speaker diarization erläutert.
Kapitel 3 geht auf den verwendeten Datensatz ein. Zusätzlich wird ein Überblick darüber geschaffen, welche Bibliotheken zur Umsetzung genutzt wurden und warum sich für diese entschieden wurde.
In Kapitel 4 wird dargestellt, wie die finale Anwendung genutzt werden kann. Dabei wird hier darauf eingegangen, welche Dateien zur Ausführung notwendig sind.
In Kapitel 5 wird das Systemdesign präsentiert und es werden die internen Vorgänge während der Ausführung beschrieben.
Das abschließende Fazit zum aktuellen Stand wird in Kapitel 6 gezogen. Es wird dabei auch auf Probleme eingegangen, welche sich auf die Qualität der Ausgabe auswirken können.

[1] https://github.com/alphacep/vosk-api

2. Definitionen

Speaker diarization und Speech-to-Text sind zwei Verfahren, die genutzt werden, um die Ausgabe der Anwendung zu generieren. Es folgt daher ein kleiner Überblick über die Grundlagen beider Verfahren.

2.1 Speaker diarization

Speaker Diarization ist laut [1] der Prozess, bei dem bestimmt wird, wann jemand in einer Audiodatei von unbestimmter Länge spricht.
Durch speaker diarization können mehrere Informationen aus Audiodateien extrahiert werden. Diese Informationen können beispielsweise mitgenutzt werden, um Gesprächsprotokolle zu erstellen. Auch in Radiosendung ist es oftmals der Fall, dass zwischen verschiedenen Sprechern gewechselt wird. Dabei kann dies im Radio durch Interviews, Reden, etc. geschehen.
Es gibt unterschiedliche Algorithmen zur Bestimmung der verschiedenen Sprecher, dabei lassen sich die weitverbreitetsten Ansätze der „Rich Transcriptions" zu ordnen.
Hierbei ist der zentrale Ansatz, die Sprecher in Cluster zu unterteilen. Dafür gibt es wiederum unterschiedliche Verfahren, welche sich in das bottom-up oder top-down Prinzip unterscheiden lassen. Bei dem bottom-up Prinzip wird mit vielen Clustern gestartet und es wird anschließend die Anzahl der Cluster reduziert, bis nur noch ein Cluster pro Sprecher existiert. Bei dem top-down Prinzip wird mit nur einem Cluster gestartet und es wird die Anzahl der Cluster erhöht, bis vermeintlich alle Sprecher erkannt wurden.

2.2 Speech-to-Text

Laut [2] handelt es sich bei Speech-to-Text um einen anderen Begriff für den Aufgabenbereich der speech recognition. Dabei behandelt Speech-to-Text die Aufgabe, gesprochene Bereiche in Audiodateien in ihre Textrepräsentation umzuwandeln. Man kann die Verfahren in zwei Kategorien unterscheiden. Zum einen gibt es die sprecherunabhängigen Modelle, welche unabhängig vom derzeitigen Sprecher die Wörter in einer Audiodatei erkennen. Die sprecherabhängigen Modelle hingegen werden vorab mit Aufnahmen von Sprechern trainiert und können so die Eigenheiten in den Stimmen der ausgewählten Sprecher lernen und ihre Vorhersage dadurch anpassen. Die gängigsten Algorithmen für Speech-to-Text Probleme sind das Hidden Markov Model, Dynamic Time Warping und neuronale Netze. Nach [2] sind neuronale Netze die vermeintlich effizientesten und effektivsten Algorithmen, da sie durch adaptives Lernen Verbindungen in hochkomplexen Daten erkennen können.

3. Datensatz und Bibliotheken

Zur Umsetzung der Anwendung wurde auf bereits bestehende Bibliotheken zurückgegriffen. Auf die relevantesten Bibliotheken wird in diesem Teil eingegangen. Um die Anwendung zu testen, wurde ein Datensatz von Radioaufnahmen genutzt, der folgend näher beschrieben wird.

3.1 Datensatz

Bei dem verwendeten Datensatz handelt es sich um eine Sammlung von Aufzeichnungen von 3 verschiedenen Radiosendern. Die Aufnahmen stammen dabei von den Sendern WDR2, rbb 88.8 und Antenne Bayern. Insgesamt liegen um die 950 Audiodateien vor.
Die aufgenommenen Audiodateien beinhalten neben den gewünschten Nachrichtensendungen auch Werbung und Musik, die vor und nach der Sendung gespielt wurde.
Abhängig von der Nachrichtensendung kann es auch vorkommen, dass Staumeldungen oder Wetterberichte Teil der Aufnahme sind. Diese lassen sich aber auch von den eigentlichen Nachrichten trennen, da bei den vorliegenden Beispielen ein einzigartiges Signal vor/nach den betroffenen Abschnitten gespielt wird. Die Audiodateien für Start- und Endsignal der jeweiligen Sendungen wurden mithilfe von einer Audiobearbeitungssoftware aus den Originaldateien herausgeschnitten und in neuen Dateien abgespeichert.
Es kommt auch vor, dass ein Sender unterschiedliche Start- und Endsignale verwendet, beispielsweise wegen wechselnder Tageszeit. Das hat zur Folge das mehrere Signale für manche Sender herausgeschnitten werden mussten.

3.2 Verwendete Bibliotheken

Der Vorgang, das Gesprochene in Text umzuwandeln, ist ein zentrales Thema dieser Arbeit. Die Ergebnisse aus diesem Schritt sind dabei auch die Grundlage für viele nachfolgende Arbeitsschritte.
Es gibt zur Lösung des Problems bereits zahlreiche Umsetzungen. Anfangs wurde mit der Speech Recognition[2] Bibliothek experimentiert, welche die Nutzung von Google Speech Recognition ermöglicht. Die von Google bereitgestellte Lösung hat jedoch das Problem, dass diese bei starken Hintergrundrauschen den Umwandlungsvorgang abbricht, ohne diesen später wieder aufzunehmen. Es werden auch keine Zeitstempel für die einzelnen Wörter generiert. Das verhindert die Möglichkeit, den Zeitpunkt des Abbrechens zu bestimmen und den nicht umgewandelten Teil der Audiodatei erneut zu behandeln. Die online Lösung von Google[3] hingegen

[2] https://github.com/Uberi/speech_recognition

[3] https://cloud.google.com/speech-to-text

generiert auch Zeitstempel, jedoch wird dieser Dienst nach einer gewissen Zeit der Nutzung kostenpflichtig. Auf Grundlage dessen wurde sich gegen die Lösungen von Google entschieden.
In der finalen Version der Anwendung wird Vosk genutzt. Vosk stellt dabei eine lokale und eine Server Variante bereit.
Vosk hat den Vorteil, dass es auch versucht verrauschte Teile der Audiodatei in Text umzuwandeln. Das hat den Vorteil, dass somit keine Lücken im Text entstehen. Vosk hat jedoch den Nachteil, dass bei sehr starken Hintergrundrauschen der Sinn des Satzes verloren gehen kann. Das liegt daran, dass die eindeutige Erkennung der Wörter nicht mehr möglich ist. Vosk erstellt zusätzlich für jedes erkannte Wort einen Zeitstempel. Die Zeitstempel werden für spätere Schritte benötigt, da Sprecherwechsel anhand eines Zeitpunktes in den Text eingefügt werden. Ein weiterer Nachteil von Vosk ist es, dass die Zeichensetzung, sowie Großschreibung nicht mit generiert werden. Dadurch wird die Lesbarkeit der Ausgabe beeinträchtigt, weswegen weitere Schritte erforderlich sind, um die Satzzeichen und die Großschreibung wiederherzustellen.

Zur Wiederherstellung der Satzzeichen wurde sich für die Bibliothek deep multilingual punctuation[4] [3] entschieden. Das Modell nutzt die Transformer Architektur und ein XLM RoBERTa Modell, um Satzzeichen vorherzusagen. Die Entscheidung fiel auf dieses Modell, da es mit einem durchschnittlichen F1-Score von 0.814 bei der deutschen Zeichensetzung akzeptable Ergebnisse liefert. Dabei werden . , ? : - wiederhergestellt, was zur Lösungen des vorliegenden Problems ausreichend ist.

	0	.	?	,	:	-	Average
EN	0.991	0.948	0.890	0.819	0.575	0.425	0.775
FR	0.992	0.945	0.871	0.831	0.620	0.431	0.782
IT	0.989	0.942	0.832	0.789	0.588	0.421	0.762
DE	0.997	0.961	0.893	0.945	0.652	0.453	0.814

Tabelle 1 Satzzeichen Scores[4]

Zur Wiederherstellung der Großschreibung wurde sich für Stanza[5] [4] entschieden. Stanza nutzt mehrere Modelle für die sprachliche Analyse von Texten. Dabei werden unteranderem Tags für Wörter generiert. Mithilfe der Tags kann dann bestimmt werden, welcher Worttyp vorliegt,

[4] https://github.com/oliverguhr/deepmultilingualpunctuation

[5] https://stanfordnlp.github.io/stanza/

wodurch beispielsweise bei Nomen eine Großschreibung vorgenommen werden kann. Die Ergebnisse dieses Modells werden durch die Satzzeichensetzung des vorhergehenden Modells beeinflusst, was zu einer Verschlechterung der Klassifizierung führen kann.
Es muss angemerkt werden, dass Stanza aufgrund der zahlreichen Modelle, eine längere Laufzeit als andere Lösungen in diesem Bereich hat. Jedoch wurde sich aufgrund der intuitiven Implementierung für die Nutzung dieser Bibliothek entschieden.

Um Sprecherwechsel zu erkennen, wurde die bereitgestellte Lösung von pyannote.audio[6] [5] gewählt. Pyannote.audio nutzt im Vergleich zu anderen Lösungen neuronale Netze zur Bestimmung von Sprecherwechsel. Probleme hat das genutzte Modell jedoch dabei, wiederkehrende Sprecher in den Audiodateien mit den Nachrichtensendungen zu erkennen. Um diesem Problem entgegenzuwirken, wurde sich dazu entschieden lediglich die Zeitpunkte eines Sprecherwechsels mithilfe des Modells festzustellen.

Das Zuschneiden der Audiodateien erfolgt mithilfe von pydub[7]. Zusätzlich wird pydub auch genutzt, um Ruhepausen innerhalb der Nachrichtensendung zu erkennen.

[6] https://github.com/pyannote/pyannote-audio

[7] https://github.com/jiaaro/pydub

4. Anwendung des Programms

Vorab muss die gewünschte Version des Vosk-Modells[8] in den Ordner „models" mit dem Namen „vosk_model" abgelegt werden.
Das Programm kann über die systemeigene Konsole gestartet werden, indem man Main.py ausführt. Bei Start des Programms können/müssen zusätzliche Argumente über Flags angegeben werden. Dabei stehen die folgenden Flags zur Verfügung:

- -b Pfad (optional)
  nimmt einen Pfad zu einem Ordner entgegen. Dieser Ordner enthält Dateien mit den Audiosignalen des Senders, welche vor dem ersten gesprochenen Segment in der Nachrichtensendung gespielt werden (.mp3). Die Dateien sollten ungefähr gleicher Länge sein. Wenn -t gesetzt ist, muss dieses Flag auch gesetzt werden.

- -c zahl (optional)
  gibt den Schellwert an, bei welchem der Korrelationswert beim Suchen des Start-/Endsignale als valide gewertet wird. Wenn das Flag nicht gesetzt wird, wird der Wert -1 als default angenommen und eine Überprüfung der Korrelation wird nicht durchgeführt.

- -d (optional)
  wenn das Flag gesetzt wird, wird der Inhalt des Intermediate-Ordners im Output-Ordner nach der Ausführung gelöscht.

- -db (optional)
  gibt an, ob Debuginformationen auf der Konsole ausgegeben werden sollen. Derzeit ist das lediglich der gefundene Höchstwert der Korrelation beim Schneiden der Audiodatei. Wenn das Flag nicht gesetzt wird, werden die Debuginformationen nicht ausgegeben.

- -e Pfad (optional)
  nimmt einen Pfad zu einem Ordner entgegen. Dieser Ordner enthält Dateien mit den Audiosignalen des Senders, welche nach dem letzten gesprochenen Segment in der Nachrichtensendung gespielt werden (.mp3). Die Dateien sollten ungefähr gleicher Länge sein. Wenn -t gesetzt ist, muss dieses Flag auch gesetzt werden.

- -h (optional)
  gibt die einzelnen Flags und deren Bedeutungen aus.

[8] https://alphacephei.com/vosk/models

- -i Pfad
  nimmt einen Pfad zu einer Datei entgegen, welche die Audiodatei mit der Radioaufzeichung enthält (.mp3 oder .wav).
  Es kann auch ein Pfad zu einem Ordner übergeben werden, welcher mehrere Audioaufzeichnungen enthält. Nur Dateien von Typ .mp3 und .wav werden berücksichtigt. Bei .wav Dateien wird angenommen, dass diese bereits mit der Anwendung erstellt wurden und werden deswegen nicht erneut vorverarbeitet.

- -l string
  nimmt Sprachenkürzel für die Sprache entgegen, welche in der Audiodatei gesprochen wird. Es müssen die Kürzel angegeben, welche auch von Stanza[9] genutzt werden. Der Default ist „de" (Deutsch)

- -o Pfad (optional)
  nimmt einen Pfad entgegen, in welcher die Ausgabe der Anwendung gespeichert werden soll. Falls das Flag nicht gesetzt wird, wird ein Ausgabeordner im Ordner des Programms erstellt.

- -t (optional)
  gibt an, ob die Source Datei vor Umwandlung noch getrimmt werden muss oder nicht. Falls es gesetzt wird, müssen auch Ordner mit Audiodateien für die Signale am Anfang/Ende der Nachrichtensendung mit den -b und -e Flag übergeben werden.

Sollte ein benötigtes Flag nicht angegeben sein, wird das Programm vorzeitig beendet.
Wenn alle Parameter übergeben werden, beginnt das Programm mit der Umwandlung der Datei.
Während der Ausführung wird ein zusätzlicher Intermediate-Ordner im Ausgabeordner erstellt. Dieser Ordner enthält Dateien, die während der Ausführung des Programms erstellt werden.
Die Konsole informiert den Nutzer über die aktuell ausgeführten Schritte.
Abschließend wird der umgewandelte Text auf der Konsole ausgegeben und in einer .txt Datei im angegebenen Ausgabeordner gespeichert.
Als Kennzeichnung des Sprecherwechsels wird <---New Speaker t---> eingefügt, wobei t die Zeit in Minuten und Sekunden ist, in der der Sprecherwechsel festgestellt wurde.

[9] https://stanfordnlp.github.io/stanza/available_models.html

5. System Design

Bevor auf die detaillierte Ausführung eingegangen wird, wird der grundlegende Programmaufbau näher beschrieben.

5.1 Übersicht

Die Ausführung des Programms lässt sich in zwei Abschnitte unterteilen. Der erste Abschnitt befasst sich damit, die Audiodatei für den zweiten Teil vorzubereiten. Darunter fällt, die Audiodatei auf die Nachrichtensendung zuzuschneiden und die Erstellung ein .wav Datei mit einem Audiokanal und einer Samplerate von 16KHz.
Der zweite Abschnitt nutzt die Ausgabe des ersten Teils, um durch speaker diarization und Speech-to-Text das Gesprochene aus der Sendung in Text umzuwandeln.
Um die Lesbarkeit der Ausgabe zu erhöhen werden zusätzlich auch die Großschreibung und Satzzeichen wiederhergestellt. In Abbildung 1 ist der Programmverlauf übersichtlich dargestellt.

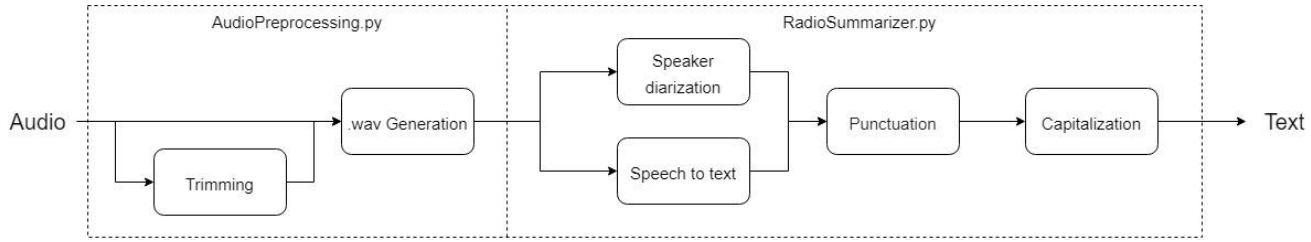

Abbildung 1: Visualisierung des Systems

5.2 Detaillierter Ablauf

Zur Ausführung des Programmes kann ein Audiofile übergeben werden, in welchem nur die Nachrichtensendung selbst oder auch die Werbung/Musik vor und nach der Sendung enthalten ist.
Es kann auch ein Ordner angegeben werden, welcher mehrere solcher Dateien enthält. Für alle Dateien vom Typ .mp3 wird erst das Zuschneiden der Audiodatei durchgeführt und anschließend werden die Dateien sequenziell in Text umgewandelt. Bei allen Dateien vom Typ .wav wird angenommen, dass diese mithilfe dieser Anwendung erstellt wurden und somit schon vorverarbeitet sind. Bei .wav Dateien wird direkt mit der Erstellung des Textes begonnen. Im weiteren Verlauf des Kapitels wird angenommen, dass eine einzige .mp3 Datei übergeben wird. Liegt eine Audiodatei mit Werbung und Musik vor, kümmert sich das Programm darum, die Audiodatei ausschließlich auf die Nachrichtensendung zuzuschneiden. Dafür muss das -t Flag gesetzt werden.
Bei einer Nachrichtensendung ist es üblich, dass ein einzigartiges Signal zu Beginn und Ende der Sendung gespielt wird. Das wird sich in der Anwendung zunutze gemacht, indem diese Signale mit übergeben werden können, um die Audiodatei zuzuschneiden.

Die Länge der Signale ist dabei abhängig vom jeweiligen Radiosender und kann stark variieren. Es ist möglich ein Audiosignal mit einer Länge von weniger als 1 Sekunde oder mehr als 2 Sekunden zu nutzen.
Manche Sender nutzen über den Tag hinweg unterschiedliche Audiosignale, dafür kann ein Ordner für jeweils die Start- und Endsignale des Senders übergeben werden.
Wichtig ist aber, dass jedes Startsignal vor dem ersten gesprochenen Segment endet und jedes Endsignal, nachdem letzten gesprochenen Segment beginnt. Die Audiodateien für die Signale müssen vom Typ .mp3 sein. Im ersten Schritt wird eine Audiodatei erstellt, welche nur auf das Segment der Nachrichtensendung geschnitten wird.
Die Umsetzung dieses Schrittes lässt sich in der trimm_audio-Methode in der AudioPreprocessing.py finden.
Um das jeweilige Audiosignal in der Datei wiederzufinden, wird die Korrelation zwischen dem Signal und der ursprünglichen Audiospur berechnet. Um die Signale in Python darstellen zu können, wird librosa [6] verwendet. Die Korrelation der Signale wird dann mit Hilfe von scipy [7] berechnet. Dadurch erhält man einen Wert, der angibt wie ähnlich verschiedene Abschnitt der Audiodatei dem Audiosignal sind.
Die entstandene Liste von Werten wird nach dem größten Wert gefiltert. Somit erhält man den Zeitpunkt, an dem das Segment beginnt, welches dem Audiosignal am ähnlichsten ist.
Ein Beispiel der ermittelten Korrelation ist in Abbildung 2 zu sehen.

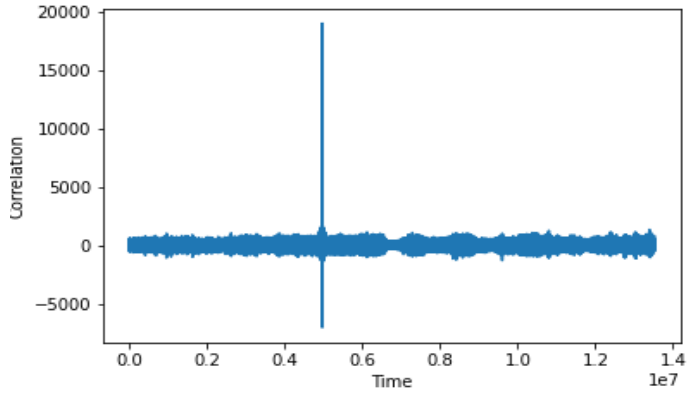

Abbildung 2: Beispielskorrelation

Wenn mit dem -c Flag ein Wert ungleich -1 übergeben wurde, wird geprüft, ob der ermittelte Wert >= dem Schwellwert ist.
Ist das der Fall, wird der Wert als valider Punkt zum Schneiden des Audios gewertet. Sollte der Wert kleiner als der Schwellwert sein, wird der Suchvorgang als misslungen gewertet. Warum der Schwellwert manuell angegeben werden muss, wird in 6.1 geschildert.
Befinden sich mehrere Dateien im jeweiligen Ordner für die Audiosignale, wird der Zeitpunkt ausgewählt, bei dem das Audiosignal mit der höchsten Korrelation ermittelt wurde. Um anschließend den Startpunkt der Nachrichtensendung zu erhalten, muss für den Beginn zusätzlich die Länge des Audiosignals hinzugerechnet werden.

Das ist erforderlich, da bei der Ermittlung der Korrelation der Startpunkt der Sequenz ermittelt wird, die dem Audiosignal ähnlich ist.
Beim Ende muss nichts hinzugerechnet werden, da sobald das Signal erkannt wird, davon ausgegangen werden kann, dass keine weiteren Nachrichten folgen werden.
Falls der bestimmte Startpunkt nach dem ermittelten Endpunkt liegt, wird die Ausführung des Programms gestoppt.
Wandelt man den erhaltenen Wert schließlich in Millisekunden um, erhält man Werte die als Start- und Endindex für ein AudioSegment aus der pydub Bibliothek genutzt werden können, umso die Audiodatei zuzuschneiden.
Die entstandene Datei wird im Intermediate-Ordner mit einer Samplerate von 16KHz und einem Audiokanal als .wav abgespeichert. Somit hat man nach Ausführung der Anwendung weiterhin Zugriff auf eine Datei, die lediglich die Nachrichtensendung enthält.
Die Datei muss mit 16KHz abgespeichert werden, da das Vosk Modell am besten mit Dateien dieses Formats funktioniert.
Wurde eine Datei übergeben, die nicht zugeschnitten werden soll, wird trotzdem eine .wav mit 16KHz im Intermediate-Ordner hinterlegt.
Mit dem nächsten Schritt beginnt die Umwandlung des Gesprochenen in Text.
In der ersten Phase wird dabei das Vosk Modell genutzt, um die Wörter in der Audiodatei zu erkennen.
Um über die Audiodatei zu iterieren, werden die Frames in der Audiodatei gesucht, bei welchen Sprechpausen erkannt werden.
Diese werden mit der split_audio-Methode in der SpeechToText.py ermittelt.
Dabei wird betrachtet, in welchen Bereichen ein gewünschter Dezibelwert für eine vorgegebenen Zeit unterschritten wird.
Dazu wird die bereitgestellte Methode silence aus der pydub Bibliothek genutzt.
Um die Anzahl der entstehenden Teile zu reduzieren, werden die kürzesten Teile mit dem Vorgänger oder Nachfolger verknüpft. Dabei wird eine Verknüpfung mit dem jeweils kürzeren Segment durchgeführt.
Die Gliederung in einzelne Abschnitte ist erforderlich, um die Qualität der Umwandlung zu erhöhen. Man könnte auch vorgeben, dass man immer die nächsten n-Frames betrachtet. Bei dieser Annahme kann es aber vorkommen, dass eine Trennung inmitten eines Wortes durchgeführt wird. Diese Aufteilung des Wortes kann sich dabei auf die Genauigkeit der Umwandlung auswirken. Um diese Fehler zu reduzieren, werden die Frames bis zur nächsten Ruhephase betrachtet.
Um die Zeitpunkte des Sprecherwechsels zu bestimmen, wird speaker diarization angewendet. Dafür wird auf ein vorhandenes Modell von pyannote.audio zurückgegriffen. Das Modell gibt für jedes Segment aus, wann dieses beginnt, endet und welcher Sprecher erkannt wurde.
Es kann dabei vorkommen, dass ein Segment mit dem gleichen Sprecher in mehrere Teilsegmente untergliedert wird.

Um dem entgegenzuwirken, werden aufeinanderfolgende Segmente mit dem gleichen Sprecher zu einem zusammengefasst.
Zusätzlich kann es während Nachrichtensendungen passieren, dass zusätzlich kleinere Einspieler abgespielt werden. Diese Einspieler lassen sich aufgrund ihrer Länge von den restlichen Sprecherwechsel unterscheiden. Um die Übersichtlichkeit der Ausgabe zu erhöhen, werden diese Segmente mit dem nachfolgenden Segment zusammengefügt.
Mit den Informationen aus dem Speech-to-Text und der speaker diarization lassen sich nun die Sprecherwechsel in den Text einfügen. Dazu wird über die einzelnen Wörter aus den Ergebnissen des Speech-to-Text Prozesses iteriert und zu einem Text zusammengefügt. Während jeder Iteration wird geprüft, ob das aktuelle Wort einen späteren Zeitstempel besitzt als der Startzeitpunkt des nächsten Sprechers. Falls das der Fall ist, wird vor dem aktuellen Wort die Kennzeichnung des Sprecherwechsels eingefügt.
Es wurde sich dafür entschieden die Einfügung des Sprecherwechsels vor der Wiederherstellung der Satzzeichen durchzuführen, da das Modell zur Wiederherstellung der Satzzeichen auch Sätze über Sprecherwechsel hinweg erkannt hat. Durch das vorhergehende Einfügen der Sprecherwechsel wird diesem Problem entgegengewirkt. Anschließend werden die Satzzeichen wiederhergestellt, indem ein multilinguales Modell zur Satzzeichenherstellung aus der deep multilingual punctuation Bibliothek genutzt wird.
Mit der Ausgabe dieses Schrittes wird abschließend die Großschreibung mithilfe von Stanza wiederhergestellt. Bei der Nutzung des Modells wird zuerst jedem Wort ein bestimmter Tag zugewiesen. Dabei kann man anhand des upos „Noun" und „Propn" erkennen, dass das jeweilige Wort großgeschrieben werden muss. Zusätzlich wird auch das vorherige „Wort" betrachtet. Handelt es sich dabei um ein Satzzeichen (außer ,) lässt sich herleiten, dass das aktuelle Wort großgeschrieben werden muss.
Zusätzlich werden Zeilenumbrüche an vorgegebenen Stellen eingefügt, um so die Übersichtlichkeit zu verbessern.
Dies ist der finale Text, der von der Anwendung generiert wurde. Dieser Text wird zum einen auf der Konsole ausgegeben und zusätzlich in eine Datei geschrieben, welche im gewünschten Ausgabeordner erstellt wird. Der Name dieser .txt-Datei entspricht dem Namen der ursprünglichen Audiodatei.

6. Bewertung

Nun folgt ein Überblick zu den erkannten Schwächen, die Auswirkung auf die Ausgabe des Programms haben können. Abschließend folgt noch ein zusammenfassendes Fazit.

6.1 Schwächen

Das Programm hat in der aktuellen Version noch Schwächen, welche Einfluss auf die Qualität der Ausgabe haben können
Bei der speaker diarization kann derzeit noch nicht zuverlässig erkannt werden, wenn der gleiche Sprecher erneut an der Reihe ist. Um trotzdem von der speaker diarization zu profitieren, wurden deswegen nur die Sprecherwechsel als solche gekennzeichnet.
Wenn man also ein genaueres Modell speziell für Radiosendungen trainiert, könnte man gegebenenfalls zuverlässig wiederkehrende Sprecher erkennen.
Derzeit können Interviews auch ein großes Problem in dem Speech-to-Text Teil darstellen. Nämlich ist es bei Interviews entscheidend, in welcher Umgebung diese geführt wurden.
Werden Interviews beispielsweise auf der Straße während starken Windes geführt, hat man in der Audiodatei laute Hintergrundgeräusche.
Diese konnten auch nicht durch Rauschentfernung angepasst werden, so dass das Modell noch weitere valide Ergebnisse lieferte. Meistens werden nach der Rauschentfernung nur einzelne Wörter oder gar keine Wörter mehr erkannt.
Die angewandten Rauschentfernungsmethoden waren dabei generalisierend, wie beispielsweise ein low-pass Filter. Eine Rauschentfernung für das einzigartige Rauschen in dem jeweiligen Interview ist nicht umsetzbar, da die Interviews gezielt auf das Gesprochene zugeschnitten werden. Somit besteht nicht die Möglichkeit, an Audioausschnitte zu gelangen, in welchen das Rauschen isoliert von der Stimme ist.
Das Rauschen kann auf mehrere Bereiche Einfluss nehmen. Es reduziert zum einen die Genauigkeit, mit welcher ein Sprecherwechsel erkannt wird, wenn beispielsweise mehrere Sprecher mit vergleichbaren Hintergrundrauschen nacheinander interviewt werden.
Zusätzlich reduziert sich auch die Genauigkeit des erkannten Textes in diesen Bereichen. Das kann teilweise dazu führen, dass der Sinn eines Satzes verloren geht.
Man muss an dieser Stelle aber anmerken, dass diese Art von Interviews nicht den Hauptteil der Nachrichtensendung ausmachen. Zusätzlich wird in der Regel der Kontext des Interviews zuvor angesprochen.
In manchen Nachrichtensendungen werden auch Reden übertragen.
Falls diese Reden auf Englisch gehalten werden, hat das Modell Probleme, diese erfolgreich in Text umzuwandeln. Der Grund dafür ist, dass das Modell nur eine Sprache isoliert betrachtet.

Ein weiteres mögliches Problem ist zudem die Feststellung des Start- und Endpunktes der Nachrichtensendungen. Es ist hier der Fall denkbar, dass die Technik des Radiosenders versagt und beispielsweise die erwarteten Signale nicht eingespielt werden. Somit lässt sich die Audiodatei nicht mit Sicherheit auf die Nachrichtensendung zuschneiden.
Wie bereits geschildert, kann der Nutzer selbst einen Schwellwert für die Korrelation angeben. Die Eingabe muss vom Nutzer kommen, da eine Filterung nach einem festen Schwellwert für alle Radiosender im Programm nicht möglich ist. Ein Grund ist, dass sich die erforderliche Länge der Audiosignal zwischen den Radiosender stark unterscheidet. Das hat wiederrum Einfluss auf die berechneten Werte der Korrelation und führt zu starken Variationen zwischen den Radiosendern.
Beispielsweise wurde ein Test mit Nachrichtensendungen von rbb88.8 und Antenne Bayern durchgeführt. Dabei wurde bei einer Nachrichtensendung von Antenne Bayern, manuell das Startsignal entfernt. Beim nachfolgenden Test wurde trotzdem eine Korrelation von 283 ermittelt, obwohl das Signal nicht mehr vorhanden war.
Bei einer unveränderten Aufnahme von rbb 88.8 wurden das Startsignal mit einer Korrelation von 180 richtig erkannt. Wenn man also anhand des Wertes aus dem Antenne Bayern Test filtert, würde man richtige Werte von rbb 88.8 ausschließen.
Die manuelle Eingabe des Nutzers ist keine ideale Lösung, da er vorab experimentieren muss, welche Schwellwerte für welche Sender angebracht sind. Zusätzlich ist es mühsam die verschiedenen Signale aus den Sendungen herauszuschneiden.
Wenn man eine Technik unabhängig dieser Signale implementiert, könnte das die Zuverlässigkeit des Systems erhöhen und den Nutzer entlasten. Dennoch ist der Regelfall, dass das erwartete Signal eingespielt wird, weswegen diese Anpassung nicht entscheidend zur Verbesserung der Ergebnisse beitragen würde.

6.2 Fazit

Abschließend lässt sich sagen, dass die Lösung des Problems nicht ohne Abzüge in der Genauigkeit der Umwandlung möglich ist, da es stark von der Qualität der Audiodatei abhängt. Wenn Nachrichtensendungen mit durchweg guter Qualität vorliegen, lassen sich auch gute Ergebnisse erzielen.
Sollten aber Teile der Audiodatei starke Hintergrundgeräusche enthalten, muss man mit einer schlechteren Qualität der Ausgabe für diese Teilbereiche rechnen.
Abseits dieser Problemfälle werden die Audios zuverlässig in Textform umgewandelt und mithilfe von Stanza und deep multilingual punctuation lassen sich zusätzlich die Satzstrukturen wiederherstellen.
Hierbei gibt es aber auch keine 100% Genauigkeit, da diese Ergebnisse auf den des Speech-to-Text Modells aufbauen.
Das Programm wurde nur mit Audiodateien der deutschen Sprache getestet. Trotzdem können auch Audios anderer Sprache an das Programm

übergeben werden. Dafür muss lediglich das jeweilige Sprachenkürzel bei Programmstart angegeben werden und das entsprechende Vosk Modell im dafür vorgesehenen Ordner abgelegt werden.
Abschließend wurde mit der Ausgabe dieser Anwendung ein neuer Datensatz erstellt. Dieser Datensatz enthält die schriftliche Repräsentation der Nachrichtensendungen und kann genutzt werden, um ein Modell zu trainieren, welches den Text in Themengebiete untergliedert.
Abseits davon liefert die Ausgabe des Programms derzeit schon einen Vorteil zur schnelleren Informationsaufnahme. Bei Audioformaten ist man an die Länge der Audiospur gebunden, wo hingegen man beim Lesen sein eigenes Lesetempo vorgibt. Somit hat man die Möglichkeit, mehr Informationen in kürzere Zeit zu erfassen, was die Erstellung von neuen Artikeln beschleunigen kann. Die Untergliederung in Themenbereiche würde so durch die Möglichkeit der Filterung von Informationen eine weitere Verbesserung im Vergleich zum reinen Audioformat mit sich bringen.

Literatur

[1] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland and O. Vinyals, "Speaker Diarization: A Review of Recent Research," in IEEE Transactions on Audio, Speech, and Language Processing, vol. 20, no. 2, pp. 356-370, Feb. 2012, doi: 10.1109/TASL.2011.2125954.

[2] Trivedi, nbspPahini A.. "Introduction to Various Algorithms of Speech Recognition: Hidden Markov Model, Dynamic Time Warping and Artificial Neural Networks." International Journal of Engineering Development and Research 2 (2014): 3590-3596.

[3] Guhr, Oliver et al. "FullStop: Multilingual Deep Models for Punctuation Prediction." SwissText (2021).

[4] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In Association for Computational Linguistics (ACL) System Demonstrations. 2020.

[5] Bredin, Herve et al. "Pyannote.Audio: Neural Building Blocks for Speaker Diarization." ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020. 7124–7128. Web.

[6] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.