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

Bachelorprojekt-Bericht

Umwandlung von Radiosendung in Textform

Bastian Lang
3089941

Ort, Datum

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.

<Ort>, am <Datum>

Zusammenfassung

Beschreibung des RadioSummarizer-Projektes, womit Nachrichtensendungen im Radio in Text umgewandelt werden sollen. Mithilfe der Anwendung können auch Audioaufnahme auf lediglich die Radiosendung zugeschnitten werden. Im Bericht wird dabei auf die Anwendung des Programms und den internen Vorgängen während der Ausführung eingegangen. Abschließend wird auch besprochen welche Umstände die Ausgabe negativ beeinträchtigen können.

Abstract

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

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1. Einleitung

Um neue Zeitungsartikel zu schreiben, benötigt man Information aus anderen Quellen. Nachrichtensendungen im Radio können eine dieser Quelle sein.
Jede Nachrichtensendung auf verschiedenen Sendern zu verfolgen, ist dabei aber nicht wirklich praktikabel. Man könnte diese zwar aufzeichnen und zu einem späteren Zeitpunkt anhören, jedoch erfordert diese Vorgehensweise viel Zeit.
Das kann dazu führen, dass man zur Einhaltung der zeitlichen Abgabefristen eines Artikels nicht alle nötigen Informationen zusammenbekommt, da man viel Zeit in das Hören der einzelnen Sendungen investieren musste.
Zwar könnte man die Audios in schnellerer Geschwindigkeit abspielen, aber man hat noch immer nicht die Möglichkeit, die Audio von Beginn an nach einem Thema zu filtern.
Eine große Unterstützung für Autoren wäre es daher, wenn ein Programm die Nachrichtensendungen aufzeichnet und so transformiert, dass am Ende ein Text erstellt wird, welcher die Informationen der Nachrichtensendungen Themenbereiche untergliedert.
So wären die Autoren in der Lage den Inhalt der unterschiedlichen Nachrichtensendungen zu filtern.
Ziel des Projektes ist es, einen Teil zur Lösung dieses Problems beizutragen. Hierbei wird der Prozess bis hin zur Erstellung eines Textes umgesetzt, bei welchem bereits Sprecherwechsel gekennzeichnet sind.
Sprecherwechsel werden eingefügt, da diese auch auf Themenwechsel hindeuten können. Somit liefern diese womöglich relevante Informationen, die zur Fortführung des Projekts hilfreich sein könnten.
Zur Umsetzung der Aufgabe wird auf Vosk zurückgegriffen, welches eine bereits bestehende Speech-to-Text Lösung ist.
Zur Erkennung der Sprecherwechsel wird speaker diarization angewendet.
In Kapitel 2 des Berichts werden kurz die Prozesse des Speech-to-Text und speaker diarization erläutert.
In Kapitel 3 wird der verwendete Datensatz besprochen und schafft einen Überblick darüber, welche größeren Bibliotheken zur Umsetzung genutzt wurden und warum sich für diese entschieden wurde.
In Kapitel 4 wird dargestellt, wie das Programm genutzt werden kann. Dabei wird hier auch darauf eingegangen, welche Dateien zur Ausführung notwendig sind.
In Kapitel 5 wird das Systemdesign präsentiert und es werden detailliert die internen Vorgänge des Programms, während der Ausführung, beschrieben.
In Kapitel 6 wird ein abschließendes Fazit zum aktuellen Stand gezogen und zeigt auch Probleme auf, welche sich auf die Qualität der Ausgabe auswirken können.

2. Defintionen

Speaker diarization und Speech-to-Text sind zwei der wichtigen Verfahren, die genutzt werden, um den Ausgabetext zu erstellen. Nachfolgend daher ein kleiner Überblick, über die Grundinformationen beider Verfahren.

2.1 Speaker diarization

Speaker diarization [1] ist der Prozess, um zu bestimmen, wann jemand in einem Audio unbestimmter Länge spricht. Dabei spielt die Anzahl der verschiedenen Sprecher keine Rolle.
Durch speaker diarization können somit mehrere Informationen aus Audiospuren extrahiert werden, welche beispielsweise mitgenutzt werden können, um Gesprächsprotokolle zu erstellen. Auch in Radiosendung ist es oftmals der Fall, dass unterschiedliche Sprecher zu Wort kommen. Dabei kann dies im Radio durch Interviews, Reden, etc. geschehen. Es gibt unterschiedliche Algorithmen zur Bestimmung der verschiedenen Sprecher. Die weitverbreitetsten Technologien lassen sich zumeist unter den „Rich Transcriptions" wieder finden.
Der zentrale Ansatz ist es, die Sprecher in Cluster zu unterteilen. Dafür gibt es wiederum unterschiedliche Ansätze, welche sich in einen bottom-up oder top-down Prinzip untergliedern lassen. Bei den bottom-up Prinzip startet man mit vielen Clustern und versucht anschließend die Anzahl der Cluster zu reduzieren, bis nur noch ein Cluster pro Sprecher existiert.
Bei top-down Ansätzen wird normalerweise mit nur einem Cluster gestartet und erhöht die Anzahl der Cluster 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 Audios in ihre Textrepräsentation umzuwandeln.Unterscheiden lassen sich die angewandten Verfahren für Speech-to-Text in zwei Kategorien. Zum einen gibt es die sprecherunabhängigen Modelle, welche unabhängig vom derzeitigen Sprecher die Wörter in der Audio erkennen sollen. Zum anderen gibt es sprecherabhängige Modelle. Diese Modelle 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. Da neuronale Netze durch adaptives Lernen Verbindungen in komplexen Daten erkennen können, sind sie die effektivsten Algorithmen für diese Art von Aufgabe.

3. Datensatz und Bibliotheken

Zur Umsetzung des Programms wurde auf bestehende Bibliotheken zurückgegriffen. Auf die relevantesten Bibliotheken wird in diesem Teil eingegangen. Um das Programm 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 unterschiedliche 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 Sendung wurden mithilfe von einer Audiobearbeitungssoftware aus der Originaldatei herausgeschnitten und in neue 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 einen Sender bestimmt werden mussten.

3.2 Verwendete Bibliotheken

Der Vorgang, das Gesprochene in Text umzuwandeln, ist ein zentrales Thema dieser Arbeit. Die Ergebnisse aus diesem Schritt haben eine tragende Rolle, da weitere Schritte darauf aufbauen.
Es gibt zur Lösung des Problems zahlreiche bestehende Lösungsmöglichkeiten. Zwischenzeitlich wurde auch mit einer lokalen Lösung von Google experimentiert.
Die von Google bereitgestellte Lösung hat jedoch das Problem, dass diese bei starken Hintergrundrauschen den Umwandlungsvorgang abbrechen kann, ohne diesen später wieder aufzunehmen. Es werden unteranderem auch keine Zeitstempel für die einzelnen Wörter generiert. Das verhindert die Möglichkeit, den Zeitpunkt des Abbrechens zu bestimmen und den nicht konvertierten Teil des Audios erneut zu behandeln.
Es gibt zwar auch eine online Lösung von Google, welche auch Zeitstempel generiert, jedoch ist diese nach einer gewissen Zeit der Nutzung des Service kostenpflichtig.

Auf Grundlage dessen wurde sich gegen die Lösungen von Google entschieden.
Die Entscheidung fiel auf Vosk[1]. Vosk stellt dabei eine lokale und eine Server Variante bereit.
Vosk hat den Vorteil, dass es versucht verrauschte Teile der Audio auch in Text umzuwandeln. Das hat den Vorteil, dass somit keine Lücken im Text entstehen. Es hat jedoch den Nachteil, dass bei sehr starken Hintergrundrauschen der Sinn des Satzes verloren gehen kann.
Das liegt daran, dass eine Erkennung der Wörter nicht mehr möglich ist.
Vosk erstellt zusätzlich aber auch Zeitstempel für jedes erkannte Wort.
Diese werden für spätere Schritte benötigt, da Sprecherwechsel anhand eines Zeitstempels in den Text eingefügt werden.
Ein weiterer Nachteil von Vosk ist, dass die Zeichensetzung, sowie Groß- und Kleinschreibung nicht mit generiert wird. Dies beeinträchtig die Lesbarkeit der Ausgabe, weswegen weitere Schritte erforderlich sind, um die Satzzeichen und die Großschreibung wiederherzustellen.

Zur Wiederherstellung der Satzzeichen wurde die Bibliothek deep multilingual punctuation[2] [3] gewählt. 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 für die deutsche Zeichensetzung akzeptable Ergebnisse liefert. Dabei werden . , ? : - wiederhergestellt, was für das vorliegende Problem auch 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

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

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

Datensatz und Bibliotheken

Zur Wiederherstellung der Großschreibung wurde sich für Stanza[3] [4] entschieden. Stanza nutzt dabei mehrere Modelle für die sprachliche Analyse von Texten. Dabei können unteranderem Tags für Wörter generiert werden. Mithilfe dieser Tags kann dann bestimmt werden, welcher Worttyp vorliegt, um so beispielsweise bei Nomen eine Großschreibung vorzunehmen. Dabei werden die Ergebnisse aus diesem Modell aber auch durch die Satzzeichensetzung des vorhergehenden Modells beeinflusst, was zu einer leichten Verschlechterung der Klassifizierung führen kann.
Angemerkt werden muss, dass es eine etwas längere Laufzeit aufgrund der verschiedenen Modelle hat als andere Lösungen in diesem Bereich. Jedoch wurde sich aufgrund der intuitiven Implementierung zur Nutzung dieser Bibliothek entschieden.

Um Sprecherwechsel zu erkennen, wurde sich für die bereitgestellte Lösung von pyannote.audio[4] [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 der Nachrichtensendung zu erkennen. Deswegen wurde sich dazu entschlossen, lediglich die Genauigkeit hinsichtlich des tatsächlichen Sprecherwechsels mit diesem Modell umzusetzen.

Das zuschneiden der Audio erfolgt mithilfe von pydub[5]. Zusätzlich wird pydub auch genutzt, um Ruhepausen innerhalb der Nachrichtensendung zu erkennen.

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

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

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

4. Anwendung des Programms

Vorab muss die gewünschte Version des Vosk-Modells 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. Diese Ordner enthält Dateien mit den Audiosignalen des Senders, welche vor dem ersten gesprochenen Segment in der Nachrichtensendung gespielt wird (.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 welcher der Korrelationswert beim Suchen des Zeitpunkts 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 auch mit auf der Konsole ausgegeben werden sollen. Derzeit ist das lediglich der gefundene Höchstwert der Korrelation beim Schneiden des Audios. Wenn das Flag nicht gesetzt wird, werden die Debuginformationen nicht ausgegeben.

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

- -h (optional)
  um die einzelnen Flags und deren Bedeutungen aufzulisten.

Anwendung des Programms

- -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 mit der Anwendung erstellt wurden und werden deswegen nicht weiter vorverarbeitet.

- -l string
  Sprachenkürzel für die Sprache, welche in dem Audio gesprochen wird. Es müssen die Kürzel angegeben, welche auch von Stanza[6] genutzt werden. Der Default ist „de" (Deutsch)

- -o Pfad (optional)
  nimmt einen Pfad entgegen, in welcher die Ausgabe des Programms 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 Audios 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, da nicht alle nötigen Parameter zur Ausführung bereitgestellt wurden.

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 während der Ausführung über den aktuellen Schritt, welcher gerade ausgeführt wird.
Abschließend wird der umgewandelte Text auf der Konsole ausgegeben und zusätzlich eine .txt Datei im gewünschten Ausgabeordner erstellt. Die erstellte .txt enthält dabei den umgewandelten Text.
Als Kennzeichnung des Sprecherwechsels wird <---New Speaker t--->

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

eingefügt, wobei t die Zeit in Minuten und Sekunden ist, in der der Sprecherwechsel erkannt wurde.

5. System Design

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

5.1 Übersicht

Grundsätzlich lässt sich die Ausführung des Programms in zwei übergeordnete Abschnitte gliedern.
Der erste Abschnitt befasst sich damit, das Audio für den zweiten Teil vorzubereiten. Darunter fällt, das Audio auf die reine Nachrichtensendung zu zuschneiden und die Erstellung ein .wav Datei mit der richtigen Samplerate.
Der zweite Abschnitt nutzt dann diese Ausgabe, um durch speaker diarization und Speech-to-Text das Gesprochene aus der Sendung als Text zu repräsentieren.
Um die Lesbarkeit dieser Ausgabe zu erhöhen werden zusätzlich auch die Großschreibung und Satzzeichen wiederhergestellt. In Abbildung 1 ist der Programmverlauf übersichtlich dargestellt.

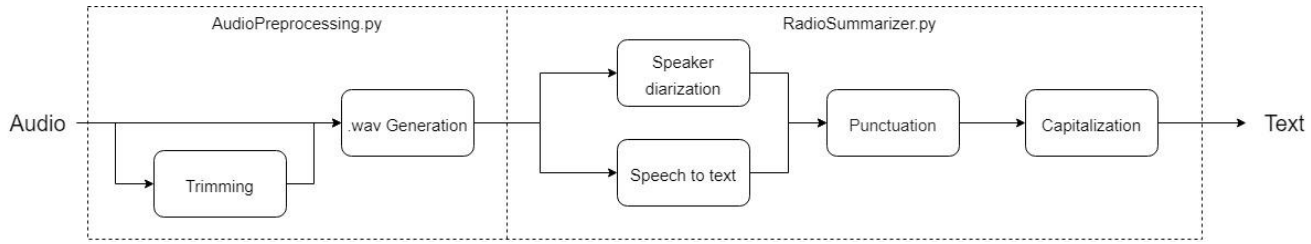

Abbildung 1: Visualisierung des Systems

5.2 Detaillierter Ablauf

Zur Ausführung des Programmes kann ein Audiofile übergeben werden, in welchem nur eine Nachrichtensendung selbst oder aber auch die Werbung/ Musik noch enthalten ist.
Es kann auch ein Ordner angegeben werden, welcher mehrere solcher Dateien enthält. Für alle Dateien vom Typ .mp3 wird erst das Zuschneiden durchgeführt und dann werden alle Dateien, die zugeschnitten wurden, sequenziell in Text umgewandelt. Bei allen Dateien vom Typ .wav wird angenommen, dass diese mithilfe des Programms erstellt wurden und somit schon vorverarbeitet sind. Bei .wav Dateien wird somit 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, das Audio ausschließlich auf die Nachrichtensendung zuzuschneiden. Dafür muss aber das -t Flag gesetzt werden.

Bei einer Nachrichtensendung ist es üblich, dass ein einzigartiges Signal zu Beginn und Ende der Nachrichten gespielt wird. Das wird sich im Programm zunutze gemacht, indem diese Signale mit übergeben werden. Die Länge der Signale ist dabei abhängig vom jeweiligen Radiosender und kann dabei stark variieren. Es ist möglich ein Audiosignal mit einer Länge von weniger als 1 Sekunde oder mehr als 2 Sekunden zu nutzen.
Da manche Sender über den Tag hinweg unterschiedliche Audiosignal nutzen, 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 müssen vom Typ .mp3 sein. Spätere Schritte nutzen .wav Dateien, weswegen nach der Vorverarbeitung eine .wav Datei im Intermediate-Ordner ausgegeben wird. Das die Eingabe eine .mp3 Datei sein soll liegt daran, dass die Dateien im genutzten Datensatz vom Typ .mp3 sind.
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 Method in der AudioPreprocessing.py finden.
Um das jeweilige Audiosignal in der Datei wiederzufinden, wird die

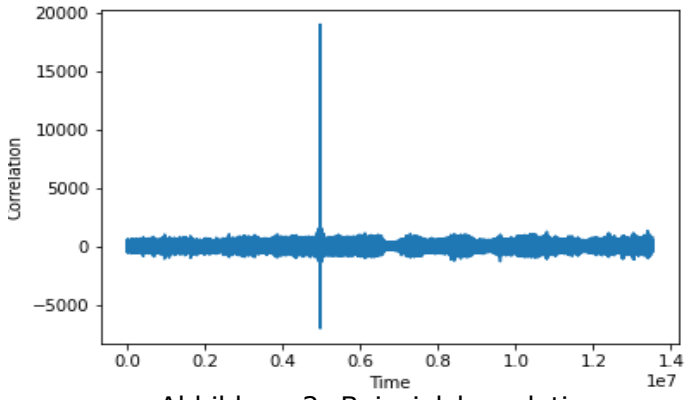

Abbildung 2: Beispielskorrelation

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, welcher angibt, wie ähnlich verschiedene Abschnitt des Audios dem Audiosignal sind.
Die entstandene Liste von Werten wird nach dem größten Wert gefiltert. Dadurch erhält man den Zeitpunkt, bei welchem das Segment beginnt, welches dem Audiosignal am ähnlichsten ist. Ein Beispiel der ermittelten Korrelation ist in Abbildung 2 zu sehen.
Wenn mit dem -c Flag ein Wert ungleich -1 übergeben wurde, wird geprüft, ob der ermittelte Wert >= dem Schwellwert ist.

Ist das der Faill, 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 Audiosignal, wird abschließend der Zeitpunkt genommen, bei welchem das Audiosignal mit der höchsten Korrelation gefunden wurde.
Um anschließend den Startpunkt der Nachrichtensendung zu erhalten, muss für den Beginn nun die Länge des Audiosignals hinzugerechnet werden. Das ist erforderlich, da wir bei der Ermittlung der Korrelation den Startpunkt der Sequenz erhalten, 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, welche man als Start- und Endindex für ein AudioSegment, aus der pydub Bibliothek, nutzen kann, umso das Audio zuschneiden zu können.
Die hierbei entstandene Datei wird im Intermediate-Ordner mit einer Samplerate von 16000kHz als .wav abgespeichert. Somit hat man nach Ausführung des Programms auch schnellen Zugriff auf eine Datei, mit der reinen Nachrichtensendung.
Die Datei wird mit 16000kHz abgespeichert, da das Vosk Modell am besten auf Dateien dieser Form funktioniert.
Wurde eine Datei übergeben, die nicht zugeschnitten werden soll, wird trotzdem eine .wav mit 16000kHz 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 dem Audio zu erkennen.
Um über das Audio zu iterieren, werden die Frames in dem Audio gesucht, bei welchen Sprechpausen vorliegen.
Dies wird 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 bereitgestellt 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 vollzogen.
Der Gesamtprozess die Frames von Ruhephasen im Gesprochenen zu finden ist erforderlich, um die Qualität der Umwandlung zu erhöhen. Man könnte auch vorgeben, dass man immer die nächsten x-Frames

betrachtet. Dabei kann es aber vorkommen, dass ein Wort in zwei Teil aufgeteilt wird. Diese Trennung des Wortes kann sich dabei auf die Genauigkeit der Umwandlung auswirken. Und um diese Fehler zu reduzieren, betrachten wir die Frames bis zur nächsten Ruhephase. Dadurch wird die Chance erhöht, dass ein Wort nicht in zwei Segmente getrennt wird.

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 und 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 bei Nachrichtensendungen dazu kommen, dass es einen kleinen Einspieler gibt. Diese Einspieler lassen sich aufgrund ihrer Länge von den restlichen Sprecherwechsel unterscheiden. Um die Übersichtlichkeit der Ausgabe zu erhöhen, werden diese Segmente dem nachfolgenden Segment angehängt.

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 dem Speech-to-Text Ergebnis iteriert und 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 entschieden die Einfügung des Sprecherwechsels vor der Wiederherstellung der Satzzeichen durchzuführen, da das Modell zur Wiederherstellung der Satzzeichen auch Sätze über Sprecherwechsel hinweg erkennt. 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 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.

Die Lesbarkeit dieser Ausgabe wird durch einen weiteren Schritt verbessert. Dabei werden Zeilenumbrüche an vorgegebenen Stellen eingefügt, um so die Übersichtlichkeit zu stärken.
Somit erhält man den finalen Text, der als Ausgabe des Programms genutzt wird. Dieser wird zum einen auf der Konsole ausgegeben und zum anderen wird dieser aber auch in eine Datei geschrieben, welche in dem 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 derzeit Auswirkung auf die Ausgabe des Programms haben können. Abschließend folgt noch ein zusammenfassendes Fazit.

6.1 Schwächen

Das Programm hat in dieser Form jedoch noch Schwächen, welche Raum für Veränderungen lassen, so dass die Qualität der Ausgabe verbessert werden kann.
Bei der speaker diarization kann derzeit noch nicht zuverlässig erkannt werden, wenn der gleiche Sprecher wieder das Wort übernimmt. Um trotzdem von der speaker diarization zu profitieren, wurden deswegen nur die Sprecherwechsel an sich gekennzeichnet.
Wenn man also ein genaueres Modell als Grundlage nimmt, könnte man gegebenenfalls wiederkehrende Sprecher erkennen. Das könnte sich als Folge möglicherweise auch auf die Gliederung in Themen auswirken.
Derzeit können Interviews auch ein großes Problem in dem Speech-to-Text Teil darstellen. Dabei ist es entscheidend, in welcher Umgebung die Interviews geführt wurden.
Werden Interviews beispielsweise auf der Straße während starken Windes geführt, hat man in dem Audio starke 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 ein low-pass Filter. Eine Rauschentfernung für das einzigartige Rauschen in dem jeweiligen Interview ist nicht umsetzbar, da die Interviews gezielt auf das Gesprochene geschnitten werden und nicht die Möglichkeit besteht, an Audioausschnitte zu gelangen, in welchen das Rauschen isoliert von der Stimme ist.
Die Probleme wirken sich dabei auf mehrere Ergebnisse aus. Es reduziert sich dadurch die Genauigkeit, mit welcher ein Sprecherwechsel erkannt wird, wenn 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 des Satzes verloren geht. Man muss an dieser Stelle aber anmerken, dass diese Art von Interviews nicht den Hauptteil der Nachrichtensendung ausmachen. Da der Kontext der Interviews zudem vorher üblicher Weise erklärt wird, lassen sich auch viele Informationen durch die Sätze vor und nach dem Interview herleiten.

Bewertung

In manchen Nachrichtensendungen werden auch Reden ausgestrahlt. Falls diese Reden auf Englisch gehalten wurden, hat das Modell Probleme, diese erfolgreich in Text umzuwandeln. Der Grund dafür ist, dass das Modell nur die deutsche Sprache isoliert betrachtet.
Hier könnte man eine Verbesserung erzielen, indem man ein Modell trainiert, was auf der deutschen und englischen Sprache gleichzeitig angewendet werden kann.

Ein weiteres mögliches Problem ist auch das Erkennen des Start- und Endpunktes der Nachrichtensendung. Es ist auch hier der Fall denkbar, dass die Technik des Radiosenders versagt und das erwartete Signal nicht eingespielt wird. Somit lässt sich das Audio nicht mit Sicherheit auf die reine Nachrichtensendung schneiden.
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 die erforderliche Länge der Audios für den Start-/Endsignal sich zwischen den Radiosender stark unterscheiden. 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 Aufnahme einer Nachrichtensendung von Antenne Bayern, manuell das Startsignal entfernt. Beim nachfolgenden Test wurde dennoch ein Zeitpunkt ermittelt, bei dem eine Korrelation von 283 vorliegt.
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 Testes filtern, 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 auch mühsam die verschiedenen Signale aus den Sendungen herauszuschneiden. Da Signale mit Wechsel der Tageszeiten wechseln können, erfordert dies einen extra Aufwand vom Nutzer selbst. Wenn man jedoch eine Technik unabhängig dieser Signale implementiert, könnte das die Zuverlässigkeit des Systems steigern und den Nutzer entlasten. Dennoch ist der Regelfall, dass das erwartete Signal eingespielt wird, weswegen diese Verbesserung nicht entscheidend zur Verbesserung der Ergebnisse beitragen würde. Dennoch ist es wichtig zu erwähnen, da es im Bereich des Möglichen ist, dass so ein Fall eintritt.

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 ohne weiteres gute Ergebnisse erzielen.
Sollten aber Teile des Audios starke Hintergrundgeräusche enthalten, muss man mit einer schlechteren Qualität der Ausgabe für diese Teilbereiche rechnen. Das liegt unteranderem daran, dass für Speech-to-Text Probleme, die Qualität der Audio essentiell ist.
Abseits dieser Problemfälle werden die Audios gut in Textform umgesetzt und mithilfe von Stanza und deep multilingual punctuation lassen sich zuverlässig Satzstruktur wiederherstellen.
Hierbei gibt es aber auch keine 100% Genauigkeit, da diese Ergebnisse auf den des Speech-to-Text Modells aufbauen.
Getestet wurde das Programm nur mit Audios der deutschen Sprache, dennoch können auch Audios anderer Sprache an das Programm übergeben werden. Dafür muss einfach das jeweilige Sprachenkürzel bei Programmstart angegeben werden.
Im nächsten Schritt kann mit der Ausgabe dieses Programms und den bestehenden Datensatz der Audiodateien ein neuer Datensatz erstellt werden.
Dieser Datensatz würde dann aus den entsprechenden Texten zu den Audios bestehen und könnte 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.
Der nächste Schritt würde dann durch das Filtern der 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.