HSB
Hochschule Bremen
City University of Applied Sciences

Hochschule Bremen
Fakultät 4 – Elektrotechnik und Informatik
Internationaler Studiengang Medieninformatik (B.Sc.)

Bachelorarbeit

- Entwicklung eines Sprachassistenten für Programmierarbeiten in Python –

Rene Wentzel Matr. Nr.: 5090763

Abstract

In dieser Bachelorarbeit werden die Potenziale und Herausforderungen der Integration einer Spracheingabe in den Workflow des Programmierens untersucht. Ziel ist es herauszufinden, ob und inwiefern die Nutzung einer Spracheingabe den Programmierprozess effizienter und angenehmer gestalten kann und ob sich so die Nutzungslast der Tastatur verringern lässt. Dazu wird zunächst eine Analyse des Aufbaus von Code Editoren und Sprachassistenten durchgeführt. Basierend auf den gewonnen Erkenntnissen wird ein Prototyp eines Code Editors mit integriertem Sprachassistent konzipiert und anschließend implementiert. Die Effizienz und Benutzerfreundlichkeit des Prototyps wird getestet, indem Programme mit dem Code Editor erstellt werden. Die Ergebnisse dieser Arbeit sollen Aufschluss darüber geben, ob eine Spracheingabe als potenzielle Ergänzung oder Alternative zur Traditionellen Eingabemethoden bei der Programmierung sinnvoll ist.

This bachelor's thesis examines the potentials and challenges of integrating voice input into the programming workflow. The goal is to determine whether and to what extent the use of voice input can make the programming process more efficient and pleasant, and whether it can reduce the usage load of the keyboard. To this end, an analysis of the architecture of code editors and voice assistants is initially conducted. Based on the insights gained, a prototype of a code editor with an integrated voice assistant is conceptualized and implemented. The efficiency and user-friendliness of the prototype are tested by creating programs with the code editor. The results of this study aim to provide insight into whether voice input is a viable supplement or alternative to traditional input methods in programming.

Inhalt

1. Einleitung

Die Tastatur ist seit jeher ein Eckpfeiler der Computertechnologie und ein wichtiges Werkzeug für die Eingabe von Informationen in elektronische Geräte. Ihre Rolle als primäres Eingabegerät hat die Art und Weise, wie wir mit Computern interagieren, grundlegend geprägt. Trotz ihrer weit verbreiteten Anwendung und ihres unbestreitbaren Nutzens sind einige potenzielle Nachteile und Herausforderungen mit der Verwendung von Tastaturen verbunden.
Obgleich der weitreichenden Verwendung von Tastaturen, gehören Tippfehler zu ihrer Verwendung dazu. [1] Solche Tippfehler können bei der Programmierung von Software zu unerwünschten Effekten führen. Ein Buchstabendreher bei einer Variablen oder ein falscher Wert wird von der Entwicklungsumgebung nicht zwangsläufig als Fehler erkannt. Die Fehlersuche kann unnötig Zeit beanspruchen. Weiterhin gibt es bei der Erstellung von Code immer wiederkehrende feste Programmstrukturen, wie If-Abfragen oder Schleifen, die mit jeder Verwendung erneut eingegeben werden oder händisch aus bereits erstelltem Code kopiert werden müssen. Je öfter diese Programmstrukturen manuell geschrieben werden, desto wahrscheinlicher sind Tippfehler. Die Arbeitsweise, mit jeder neuen Programmstruktur denselben Code erneut einzugeben, birgt ein Verbesserungspotenzial.
Aus ergonomischer Sicht kann die längere Benutzung einer Tastatur zu Beschwerden in Hand, Schulter und Rücken führen. Das Arbeiten mit einer Tastatur führt oft dazu, dass der Oberkörper in einer Position verharrt und es so zu Verspannungen kommt. [2] Neben gesundheitlichen Beschwerden, kann dies auch zu einer verringerten Effizienz, einem langsameren Arbeitsablauf führen.

Sprachassistenten(SA) wie Amazons Alexa, Apples Siri oder Microsofts Cortana haben in den letzten Jahren zunehmend an Beliebtheit gewonnen. Diese Beliebtheit geht unter anderem auf technologische Fortschritte zurück, die in den Bereichen Spracherkennung und -ausgabe, durch die Weiterentwicklung von Natural Language Processing, erzielt wurden. So ist es aktuellen Systemen möglich auf Nutzeranfragen natürlicher zu antworten und im Vergleich zu früheren sprachbasierten Benutzerschnittstellen einen erweiterten Funktionsumfang anzubieten. [3]
Die Erkennung gesprochener Sprache gehört heutzutage zu den gängigen Technologien von Smartphone und Computer. Eine Spracherkennungssoftware muss nicht mehr langwierig trainiert und auf eine bestimmte Person angepasst werden, und auch muss der Nutzer nicht mehr abgehackt und überbetont sprechen, um von der Software erkannt zu werden. [4]
Ein Vorteil einer Spracheingabe ist die Entkopplung von der Tastatur. Bei der

Eingabe per Sprache kann zum Beispiel die Sitzhaltung geändert werden, ohne dass der Eingabefluss unterbrochen wird. Zudem ist auch eine parallele Eingabe möglich. Ähnlich wie einem realen Assistenten, könnte man dem Computer sagen, dass er eine Schleife in Zeile 30 erstellen soll, während man selbst in Zeile 20 arbeitet. Diese Auslagerung von bekannten Strukturen kann potenziell zu einer effizienteren, schnelleren und weniger fehlerhaften Arbeitsweise führen.
Bei der Nutzung von Diktiersoftware zur Texteingabe zeigt sich, dass diese Methode deutlich schneller ist, als die Eingabe über eine Tastatur [5]. Die Fehlerquote ist bei der Spracheingabe zwar höher als bei der Tastatureingabe, die absolute Fehlerquote bleibt jedoch auf einem niedrigen Niveau und die meisten Fehler beruhen auf falsch verstandenen Abkürzungen oder einzelnen Buchstaben. [5]

In dieser Arbeit soll ein Hilfswerkzeug entwickelt werden, das es ermöglicht bestimmte Programmieraufgaben über Sprachanweisungen ausführen zu lassen. Der Schwerpunkt der Aufgaben liegt bei der Erstellung sich wiederholender Programmstrukturen. Es soll ein Code Editor entwickelt werden, der sowohl per Tastatur und Maus bedient werden kann, als auch mit Hilfe eines SA. Ziel ist es eine Entlastung der Tastatur oder – je nach Bedarf – einen effizienteren Arbeitsfluss durch eine kombinierte Nutzung von Tastatur und Spracheingabe zu ermöglichen.
Um ein entsprechendes Hilfswerkzeug zu erarbeiten, wird zunächst die Funktionsweise eines SA analysiert. Weiterhin wird untersucht welche Anforderungen ein SA für diese Arbeit mit sich bringt. Für die wichtigsten Aspekte werden Methoden zur Umsetzung betrachtet und verglichen. Auf Grundlage dieser Gegenüberstellung werden jene Methoden ausgewählt, die am besten den Anforderungen dieser Arbeit entsprechen. Ebenfalls wird untersucht welche Anforderungen die Entwicklung eines einfachen Code Editors mit sich bringt und auch dazu werden geeignete Werkzeuge zur Umsetzung recherchiert. Als Resultat wird ein dem ausgewählten Methoden entsprechendes Programm konzipiert, umgesetzt und beispielhaft angewendet. Bei dem Code Editor handelt es sich um einen simplen Editor für das Schreiben, Speichern, Laden und Ausführen von Python Code. Er stellt keine voll umfängliche Entwicklungsumgebung dar.

2. Analyse von Sprachassistenten und Code Editoren

Um einen Code Editor und einen SA für die Unterstützung von Programmieraufgaben konzipieren zu können, werden zunächst der Umfang und Aufgabenbereich gängiger SA, sowie die Architektur analysiert und auch der Aufbau und Umfang von Code Editoren untersucht.

2.1 Was ist ein Sprachassistent?

Sprachassistenten, synonym auch „Voice Assistant“, „Voicebot“ oder „Digitaler Assistent“ genannt, sind Programme, die natürliche Sprache verstehen und dem Nutzer auf verschiedene Weise helfen. Apples Siri, Googles Google Assistant und Microsofts Cortana sind bekannte Beispiele für einen SA, die auf dem Smartphone oder dem Computer verwendet werden. Amazons Alexa - ein sogenannter Smart Speaker - ist ein SA in Form eines eigenständigen Gerätes.

Hauptsächlich werden mit SA verschiedene Aufgaben ausgeführt; etwa die Abfrage nach dem aktuellen Wetter, das Starten eines Timers, das Suchen nach Begriffen im Internet oder wie im Falle von Alexa auch das Hinzufügen von Produkten in den Amazon Warenkorb.
Dem Anwendungsgebiet eines SA sind dabei keine Grenzen gesetzt. So bietet Amazons Alexa Anwendungen – sogenannte Alexa Skills – in diversen Bereichen an wie Finanzen, Gesundheit und Fitness, Heimdienste, Humor, Shopping und Bildung.

Ausschlaggebend für einen SA ist, dass keine traditionelle Eingabe durch Tastatur oder Touchscreen erfolgt. Ein SA wird gänzlich über die Sprache gesteuert und versteht dabei die natürliche Sprache des Nutzers. Der Assistent hört jederzeit mit und reagiert, sobald er angesprochen wird. Wie ein persönlicher Assistent.
Antworten des SA werden in den meisten Fällen ebenfalls in Form von Sprache wiedergegeben, sodass dem Nutzer das Gefühl vermittelt wird, er unterhält sich mit seinem Gerät.

2.2 Aufbau eines Sprachassistenten

Die Funktionsweise eines SA lässt sich in fünf Bereiche unterteilen: Lauschen, Aufwachen, Verstehen, Ausführen und Antworten (Siehe Abbildung 1).

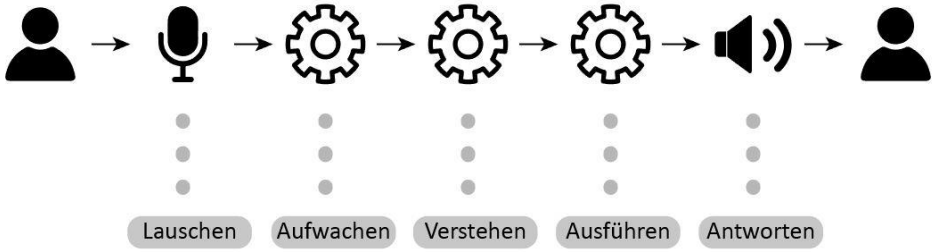


Abbildung 1: Grafische Darstellung des Aufbaus eines Sprachassistenten

2.2.1 Lauschen

Der SA hört fortlaufend über das Mikrofon eingehende Audiosignale ab. Die Audiosignale werden aufbereitet und Störsignale herausgefiltert.
In diesem Schritt wird das Audiosignal für die weitere Verarbeitung in Text übersetzt und vorverarbeitet. Die Vorverarbeitung beinhaltet üblicherweise das Kleinsetzen aller Buchstaben, das Entfernen von unwichtigen Wörtern und Sonderzeichen und das korrigieren und gleichsetzen gleicher Wörter. [6]

2.2.2 Aufwachen

Ein SA hat üblicherweise ein Aktivierungswort. Bekannt sind „Okay Google“ von Google Assistant, „Alexa“ von Alexa oder „Hey Siri“ von Siri. Das Aktivierungswort signalisiert dem SA, dass ein Sprachbefehl folgt. Ohne Nennung des Aktivierungswortes lauscht der Assistent zwar mit, führt aber keine Anweisungen aus. [6]

2.2.3 Verstehen

Nach dem Aktivieren per Aktivierungswort, verarbeitet der SA den nachfolgenden Befehl. Die Eingabe wird analysiert und einer bekannten Funktion zugeordnet. Der SA erkennt also welches Intent – welche Absicht - hinter der Eingabe des Nutzers steckt. Zudem wird die Eingabe nach zusätzlichen Informationen – Entitäten - durchsucht, die für das ermittelte Intent nützlich oder notwendig sind. [6] Gibt der Nutzer beispielsweise die Anweisung „Starte einen Timer von 10 Minuten“, dann entnimmt der SA das Intent „Timer starten“ mit der Entität „Dauer: 10 Minuten“.

2.2.4 Ausführen

In diesem Schritt wird führt der SA die zugeordnete Funktion aus. Bei der Anweisung „Starte einen Timer von 10 Minuten“ wird also eine Timer-App gestartet und auf 10 Minuten gestellt. Bei einer Frage nach dem Wetter, wird eine entsprechende Anfrage an einen Online-Wetterdienst gestellt. [6]

2.2.5 Antworten

Nachdem die Anweisung des Nutzers ausgeführt wurde, gibt der SA ein Feedback. Entweder mit den angeforderten Informationen oder mit einer Bestätigung, dass die Aufgabe ausgeführt wurde. Die Antwort wird in Form natürlicher Sprache wiedergegeben, wobei eine Übersetzung von Text zu Sprache zum Einsatz kommt. [6]

Die Übersetzung von Sprache zu Text und die Text Klassifizierung stellen sich hierbei als wichtigste Bestandteile eines SA heraus. Sie sind dafür verantwortlich, dass der SA einerseits überhaupt die Sprache des Menschen verarbeiten kann und andererseits verstehen kann, welche Anweisung der Benutzer ausführen möchte.

2.3 Funktion und Umfang eines Code Editors

Ein Code Editor ist eine Texteditor, der speziell auf das Schreiben und Verwalten von Programmcode ausgelegt ist. Dafür enthält ein Code Editor Funktionen, die das Schreiben von Code angenehmer und effizienter gestalten, wie zum Beispiel das hervorheben der Syntax oder das automatische vervollständigen von Code. Oft ist es auch möglich den Code über den Editor auszuführen. [7]

2.3.1 Syntax Hervorhebung

Bei der Syntax Hervorhebung werden verschiedene Begriffe im Code farblich hervorgehoben. Auf diese Weise verbessert sich die Lesbarkeit des Codes, wodurch dessen Kontext schneller verstanden wird. [7]
Wie in Listing 1.1 zu sehen, werden unterschiedliche Teile der Syntax unterschiedlich farblich hervorgehoben. Schlüsselwörter für Programmstrukturen wie eine Funktion oder eine Exception, so wie feste Werte wie True, False oder None werden orange dargestellt. Der Name einer Funktion wird blau hervorgehoben und String Inhalte werden grün markiert.

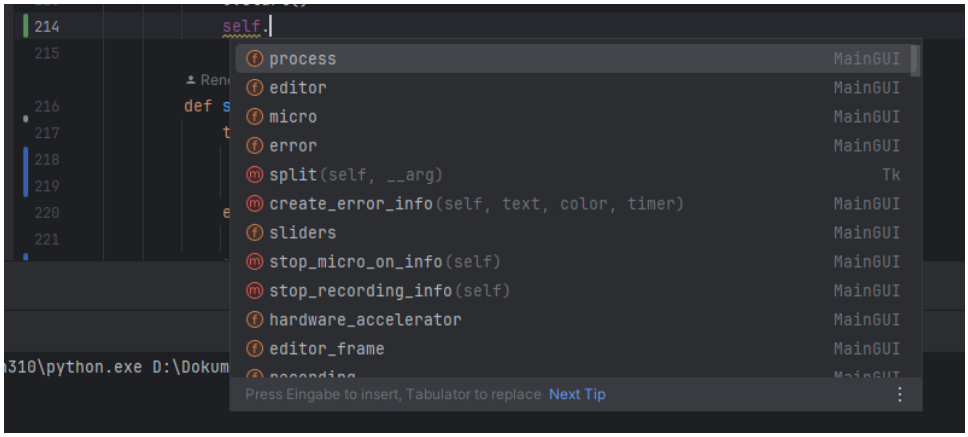
```
def stop():
    try:
        subprocess.Popen("taskkill /F /T /PID %i" % self.process.pid,
                         shell=True)
    except Exception:
        pass
```

Listing 1.1: Beispiel für die Syntax Hervorhebung von PyCharm.

2.3.2 Auto-Vervollständigung

Die Auto-Vervollständigung versucht vorherzusehen welches Wort der Benutzer eingeben möchte und schlägt dieses zur Vervollständigung vor. Mit einer Bestätigung – häufig über die Enter oder Tab Taste – kann der Vorschlag akzeptiert und der Rest des Wortes automatisch erstellt werden. [7]

In Code Editoren werden auf diese Weise auch Vorschläge für den Zugriff auf Attribute und Funktionen einer Klasse unterbreitet (Siehe Listing 1.2)



Listing 1.2: Auto-Vervollständigung Vorschläge für eine Klasse

2.3.3 Ausführen des Codes

In einigen Code Editoren ist auch das Ausführen des jeweiligen Codes möglich. Dafür verfügt der Editor entweder über einen eigenen Compiler, bzw. Interpreter oder er nutzt vorhandene Werkzeuge des Computers. [7] Ein Code Editor mit dieser Funktion hat üblicherweise ein zusätzliches Ausgabe- und Eingabefeld für den ausgeführten Code.

2.3.4 Dateiverwaltung

Wie normale Text Editoren ist es auch bei einem Code Editor möglich Code Dateien zu verwalten. Das Speichern, Öffnen und Neuerstellen von Dateien gehört zum Standard. [7] Viele Code Editoren bieten auch umfangreichere Verwaltungsmöglichkeiten für Projekte an.

3 Methoden zur Text Klassifizierung

Die automatische Zuordnung von einem in Text übersetzten Sprachbefehl zu einer auszuführenden Funktion ist eines der Hauptkomponenten eines SA. Da die auszuführenden Funktionen bei jedem SA anders sind, muss eine individuelle Lösung erarbeitet werden. Es gibt zwar vortrainierte Modelle wie BERT oder GPT, aber diese müssen dennoch durch weiteres, individuelles Training mit angepassten Datensätzen auf die gewünschte Klassifizierung abgestimmt werden.

3.1 Vorauswahl der Methoden

Für die Klassifizierung von Texten gibt es eine Vielzahl an Methoden, von denen sich einige nochmal in verschiedene Varianten aufteilen. Auf Grundlage einer Untersuchung verschiedener Methoden zur automatischen Text Klassifizierung [8] wird eine Vorauswahl für geeignete Methoden getroffen. Die Ergebnisse der Untersuchung zeigen, dass die Methoden Naive Bayes (NB), Random Forest (RF) und Künstliche Neurales Netze (KNN) eine Genauigkeit von über 90% für kurze Texte aufweisen [8]. Damit sind sie am vielversprechendsten für die Klassifizierung von kurzen Sprachbefehlen und werden weiter untersucht.

3.2 Naive Bayes

Naive Bayes – auch Simple Bayes oder Idiot's Bayes - ist ein Klassifizierungsmodell, das den Satz von Bayes zusammen mit der Annahme, dass Features bei gegebener Klasse unabhängig sind, nutzt. Auch wenn diese Annahme der Unabhängigkeiten in der Praxis oft nicht zutrifft, liefert NB dennoch häufig eine leistungsfähige Klassifizierungsgenauigkeit. Wegen dieser Genauigkeit, der geringen Recheneffizienz und anderen Merkmalen, ist NB in der Praxis weit verbreitet. [9]

Das NB Modell gilt als einfach zu implementieren. Das Training und auch die Klassifizierung sind schnell und benötigen wenig Rechenleistung. Jedoch wird das Modell ungenauer, wenn es um zusammenhängende Features geht. [10, 11]
Der Name „*Naive* Bayes" führt auf jene Annahme zurück, dass Features nur eine Abhängigkeit zu der jeweiligen Klasse, aber nicht untereinander besitzen. Da in der natürlichen Sprache die einzelnen Wörter durch grammatische Regeln und dem Kontext sehr wohl in Verbindung zueinander stehen, werden wichtige Informationen ignoriert. Die Texte „Hallo Freund" und „Freund Hallo", werden beispielsweise identisch behandelt. Daher wird diese Methode als Naiv bezeichnet.

Die Unabhängigkeit der Features spiegelt sich auch in der Anwendung des Satzes von Bayes wider.
Für die Abschätzung einer Wahrscheinlichkeit kann der Satz von Bayes wie folgt angewendet werden:

$$P(C_k \mid x_1, x_2, \ldots . x_n) = \frac{P(C_k)P(x_1, x_2, \ldots . x_n \mid C_k)}{P(x_1, x_2, \ldots . x_n)}$$

Wobei C die k-te Klasse und x die Features sind.

Für NB ist nur der Zähler der Gleichung interessant, da der Nenner nicht von Klasse C_k anhängt und die Features, die dem Nenner übergeben werden, gegeben und somit effektiv konstant sind. Dementsprechend wird die Formel angepasst: [12]

$$P(C_k \mid x_1, x_2, \ldots . x_n) = P(C_k)P(x_1, x_2, \ldots . x_n \mid C_k)$$

Beziehungsweise:

$$P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$$

Für die Text Klassifizierung werden vier Varianten von NB verwendet, Multinomial, Poisson, Bernoulli und Negative Binomial. Ein Vergleich der Varianten in einem Paper von Susana Eyheramendy, David D. Lewis und David Madigan [13] zeigt, dass die Multinomial Variante am besten abschneidet. Daher wird diese Variante für das weitere Vorgehen gewählt.
Die Multinomial Variante verwendet die Frequenz der Wörter in einem Text als Feature für die Klassifizierung.
Um das NB Modell zu trainieren, wird jeder Klasse eine Menge an passenden Texten zugeordnet. Danach wird ein Wortschatz aus allen vorhandenen Wörtern aller Klassen erstellt. Abbildung 2 zeigt ein Beispiel für ein NB Modell, welches Aussagen den Klassen „Freundlich“ und „Unfreundlich“ zuordnet. Der Wortschatz besteht aus sechs Wörtern; Helfen, Danke, Geduld, Unangemessen, Irrelevant und Fehler.

Im nächsten Schritt wird die Wahrscheinlichkeit berechnet mit der jedes Wort abhängig von der Klasse vorkommt:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Wobei $P(A \cap B)$die Anzahl des Wortes in der Klasse B ist und P(B) die Anzahl aller Wörter in der Klasse B sind.

Die Klasse „Freundlich“ im Beispiel aus Abbildung 2 beinhaltet 17 Wörter, das Wort „Helfen“ ist sechs Mal enthalten. Entsprechend lautet die Formel für die Wahrscheinlichkeit des Wortes „Helfen“ in der Klasse „Freundlich“ F:

$$P(Helfen \mid F) = \frac{6}{17} = 0{,}3529$$

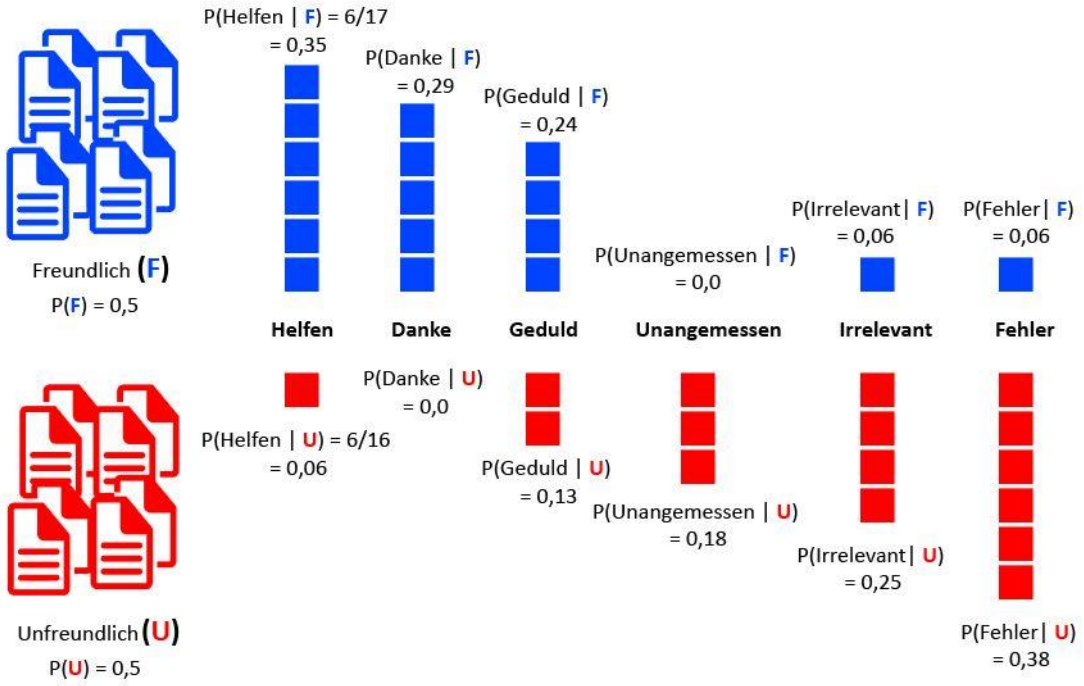


Abbildung 2: Beispiel für einen trainierten Naive Bayes Klassifikator. Rechenergebnisse gerundet auf zwei Nachkommastellen.

Zuletzt wird jeder Klasse eine A-Priori-Wahrscheinlichkeit zugeteilt. Im Beispiel aus Abbildung 2 gibt es keinen Grund zu der Annahme eine Klasse sei grundlegend häufiger vertreten, daher wird jeder Klasse dieselbe Wahrscheinlichkeit zugeteilt. P(F)=0,5 und P(U)=0,5.

Für die Anwendung des trainierten Modells, wird die zuvor genannte vereinfachte Version des Satzes von Bayes angewandt. Angenommen der zu klassifizierende Text T lautet „Helfen Irrelevant“, dann wird die Wahrscheinlichkeit für die Klassen „Freundlich“ F und „Unfreundlich“ U gemäß dem Beispiel aus Abbildung 2 wie folgt berechnet:

Freundlich:

$$P(F|T) = P(F)P(Helfen \mid F)\, P(Irrelevant \mid F)$$

$$P(F|T) = 0{,}5 * 0{,}35 * 0{,}06 = 0{,}105$$

Unfreundlich:

$$P(U|T) = P(U)P(Helfen \mid U)\, P(Irrelevant \mid U)$$

$$P(F|T) = 0{,}5 * 0{,}06 * 0{,}25 = 0{,}075$$

Der Text wird der Klasse mit dem größten Ergebnis zugeordnet. In diesem Beispiel der Klasse „Freundlich“.

Es ist möglich, dass ein Wort niemals in den Trainingsdaten einer Klasse vorkommt. Das führt unweigerlich dazu, dass die Wahrscheinlichkeit für diese Klasse auf null fällt, sobald jenes Wort im zu klassifizierenden Text enthalten ist. Das ist unerwünscht und sorgt für eine hohe Varianz.
Um das zu verhindern können die Wahrscheinlichkeiten durch einen Pseudozähler α ≥ 0 geglättet werden. Durch das Glätten wird die Varianz reduziert, gleichzeitig jedoch auch die höchstmöglichen Wahrscheinlichkeiten. Dadurch erhöht sich der Bias. In diesem Fall sind die Wahrscheinlichkeiten zwischen den Klassen ausgeglichener, je größer der Bias ist. [10]
Im NB Modell wird die Anzahl jedes Wortes in den Trainingsdaten für jede Klasse um den Wert α erhöht. Eine Möglichkeit einen passenden Wert für α zu bestimmen ist die Rastersuche.

3.3 Random Forest

Das RF Modell wurde 2001 von Leo Breiman [14] entwickelt und erstellt zufällig eine große Anzahl an Entscheidungsbäumen für die Klassifizierung. Die Ausgaben der Bäume werden über eine Mehrheitsentscheidung zu einem einzigen Ergebnis zusammengefasst. [15]
Die Entscheidungsbäume werden nach der CART-Methode erstellt [16]. Decision Trees – Entscheidungsbäume – erhalten ihren Namen von ihrem baumartigen Aufbau. Wie ein Baum haben sie eine Wurzel (Root Node), Äste (Branches) und Blätter (Leaf Nodes). Wie in Abbildung 3 zu sehen bildet die Root Node den Anfang

des Baumes. Die Root Node und die Internal Nodes (auch Decision Node) sind Entscheidungsknoten, an ihnen werden bestimmte Entscheidungsregeln aufgestellt. Diese Regeln können entweder konkrete Aussagen sein oder Schwellenwerte. Je nach Ergebnis wird dem entsprechenden Branch zur nächsten Internal Node gefolgt. Die Verarbeitung der Daten beginnt an der Root Node und läuft den Ergebnissen der Entscheidungsregeln entsprechend dem Baum entlang, bis eine Leaf Node erreicht ist. Leaf Nodes stellen die Enden des Baumes dar und damit das Ergebnis der Verarbeitung; in der Text Klassifizierung also die Klasse, die dem Text zugeordnet wird. [17, 18]

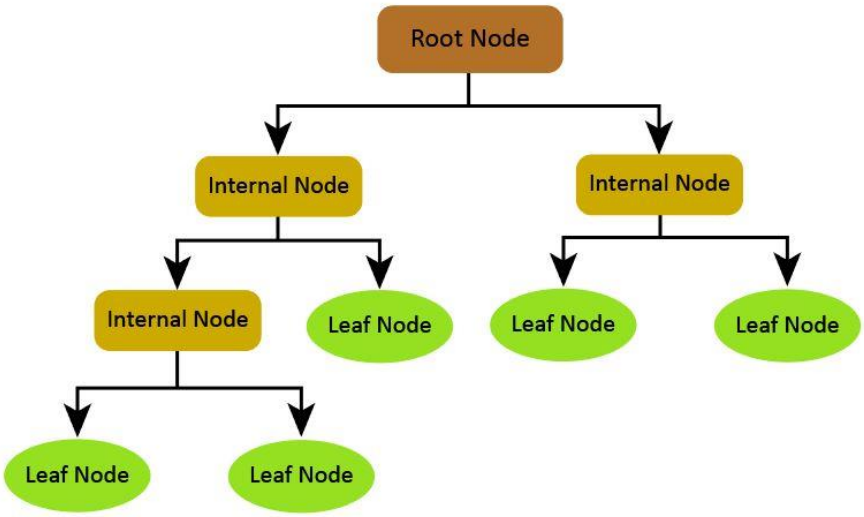


Abbildung 3: Grafische Darstellung des Konzeptes eines Entscheidungsbaums

Es gibt verschiedene Algorithmen, um einen Entscheidungsbaum zu erstellen. Für RF Modelle wird eine Form des CART Algorithmus verwendet [16].
Um einen Entscheidungsbaum zu erstellen, der mehr Informationen verwendet als Keywords („ist Wort XY enthalten“), müssen zuerst Features aus dem Text entnommen werden. Das kann mit Hilfe der Bag-of-Words Methode erreicht werden, wie bei dem NB Modell, oder über die TF-IDF Methode. Ein Vergleich beider Methoden zeigt, dass TF-IDF eine höhere Genauigkeit erzielt [19].
Die Randomisierung des RF wird auf zwei Weisen implementiert, während das Modell trainiert wird. Zuerst werden aus den Trainingsdaten Bootstrap Datensätze erstellt, indem zufällig Proben[1] ausgewählt werden. Enthält ein Datensatz wie im Beispiel in Tabelle 1.1 vier Proben, werden zufällig vier Proben für den Bootstrap Datensatz gezogen. Die Auswahl findet dabei wie bei einem Würfelwurf statt. Proben können mehrfach ausgewählt werden und es gibt keine Garantie, dass am Ende alle Proben der Trainingsdaten in den Bootstrap Datensätzen enthalten sind. [15]

[1] Proben sind in diesem Zusammenhang die Daten einer einzelnen Testperson (Siehe Tabelle 1.1)

Der Bootstrap Datensatz in Tabelle 1.2 beispielsweise enthält je einmal die erste und zweite Probe des Datensatzes aus Tabelle 1.1, zweimal die dritte Probe, jedoch nicht die vierte.

Bei großen Datensätzen sind etwa 1/3 der Proben aus den Trainingsdaten nicht in den Bootstrap Datensätzen enthalten. Diese Out-of-Bag (OOB) Proben bilden einen nützlichen Datensatz zum Testen und Anpassen der Entscheidungsbäume. [15]

	Brustschmerzen	Guter Blutkreislauf	Verstopfte Arterien	Gewicht Kg	Herzerkrankung
1	Nein	Nein	Nein	75	Nein
2	Ja	Ja	Ja	90	Ja
3	Ja	Ja	Nein	110	Nein
4	Ja	Nein	Ja	80	Ja

Tabelle 1.1: Trainingsdatensatz Beispiel mit vier Proben von Patienten, die auf Herzerkrankungen untersucht wurden.

	Brustschmerzen	Guter Blutkreislauf	Verstopfte Arterien	Gewicht Kg	Herzerkrankung
2	Ja	Ja	Ja	90	Ja
1	Nein	Nein	Nein	75	Nein
3	Ja	Ja	Nein	110	Nein
3	Ja	Ja	Nein	110	Nein

Tabelle 1.2: Aus dem Datensatz der Tabelle 1.1 erstellter Bootstrap Datensatz.

Die zweite Randomisierung erfolgt im nächsten Schritt, an den Entscheidungsknoten. An jedem Knoten wird zufällig eine Menge n an Entscheidungsregeln ausgewählt. Üblicherweise entspricht n der gerundeten Quadratwurzel der Anzahl der Entscheidungsregeln, kann zwecks Optimierung jedoch angepasst werden. Nach dem CART Algorithmus wird entschieden welche der gewählten Entscheidungsregeln letztendlich für den Baum verwendet wird. Dies wird solange wiederholt, bis die gewünschte Genauigkeit erreicht wird. [15] Die Prozedur, vom Bootstrapping bis zur Erstellung des Entscheidungsbaumes, wird wiederholt, bis ein "Wald" aus Entscheidungsbäumen entsteht. Üblich sind 100 – 1000 Bäume.

Mit den OOB Proben, kann jeder Entscheidungsbaum getestet und seine Genauigkeit ermittelt werden. Die Genauigkeit des RF entspricht generell dem Durchschnitt der Genauigkeiten aller Entscheidungsbäume.
Für die Klassifizierung durchläuft die Probe alle Entscheidungsbäume des RF. Die Klasse die von den meisten Entscheidungsbäumen ausgegeben wurde, entspricht dem Endergebnis. [15]
Das RF Modell erzielt bei der Klassifizierung eine hohe Genauigkeit. Durch die Nutzung einer Vielzahl an Entscheidungsbäumen wirkt es auch dem Overfitting eines einzigen Entscheidungsbaumes entgegen. Für eine möglichst hohe Genauigkeit ist jedoch oft eine große Menge an Entscheidungsbäumen notwendig. Dies macht das Modell rechenintensiver und erhöht die Rechenzeit. [8, 20, 21, 22]

3.4 Künstliches Neuronales Netzwerk

Ein KNN – auch Artificial Neural Network - ist ein Modell, das versucht die Funktionsweise und Struktur eines biologischen Neuronalen Netzwerkes zu imitieren.

Grundbaustein sind künstliche Neuronen – auch Nodes genannt –, die aus eingehenden Werten, durch eine festgelegte Methode einen Ausgangswert berechnen.
Die Berechnung besteht aus der Gewichtung, Summenbildung und Aktivierung der eingehenden Werte (Abbildung 4.1). Die eingehenden Werte (x_1, x_2, …, x_n) werden mit zuvor festgelegten Gewichtungen (w_1, w_2, …, w_n) multipliziert. Die Gewichtungen bestimmen wie viel Einfluss der eingehende Wert auf das Ergebnis hat. Im nächsten Schritt wird die Summe aus den Eingangswerten und einem Bias gebildet. Der Bias ist hierbei eine zusätzliche Gewichtung des gesamten Neurons, unabhängig der Eingabewerte. Im Verlauf des Trainings und der Optimierung eines KNN – der sogenannten Backpropagation - werden die Gewichtungen und der Bias der Neuronen angepasst. Das Ergebnis wird vor der Ausgabe über eine Aktivierungsfunktion aktiviert. [23]
Je nach Art und Ziel des KNN können unterschiedliche Aktivierungsfunktionen zum Einsatz kommen. Eine Schwellenwertfunktion führt beispielsweise dazu, dass der Ausgabewert stets entweder 0 oder 1 beträgt. Die Sigmoidfunktion hingegen ermöglicht einen Ausgangswert zwischen 0 und 1. [24]

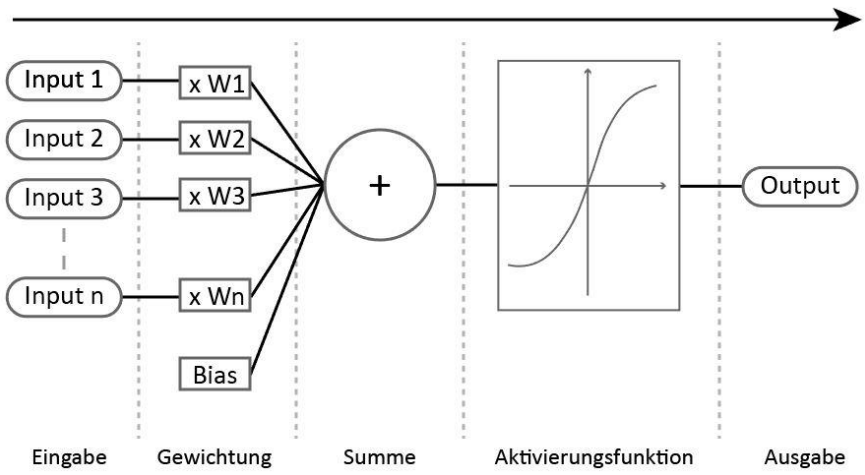


Abbildung 4.1: Grafische Darstellung der Funktionsweise eines Künstlichen Neurons

Aus einer Vielzahl Künstlicher Neuronen wird ein KNN gebildet, das in drei Schichten (Layer) unterteilt ist; dem Input Layer, Hidden Layer und Output Layer (Abbildung 4.2). Daten werden in den Input Layer eingespeist und durchlaufen das Netzwerk an Künstlichen Neuronen, bis das Ergebnis im Output Layer ausgegeben wird. Während der Input und Output Layer immer aus einer Schicht besteht, kann der Hidden Layer eine Vielzahl an Schichten enthalten.

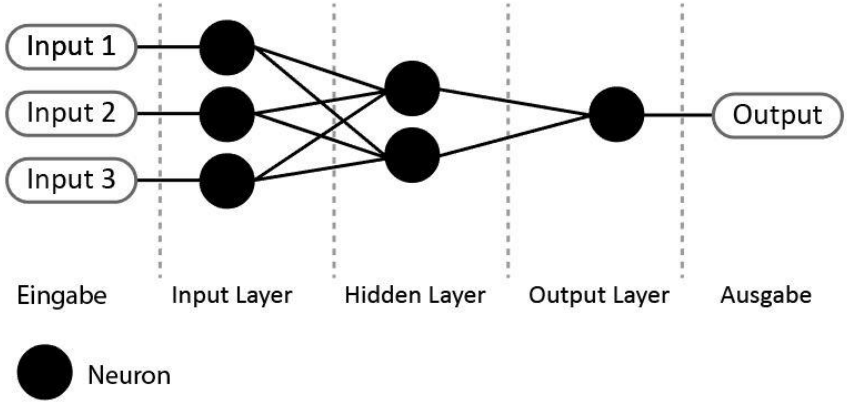


Abbildung 4.2: Grafische Darstellung eines einfachen Künstlichen Neuronalen Netzwerkes

Der Aufbau eines KNN – also die Anzahl an Hidden Layer, die Anzahl der Künstlichen Neuronen und deren Verbindung untereinander - kann sich je nach Variante unterscheiden. Grundlegend kann ein KNN der Feed-Forward und Recurrent Topologie zugeordnet werden.
In einem Feed-Forward Neural Network fließen die Daten in einer Richtung vom Input zum Output (Abbildung 4.3 Links). In einem Recurrent Neural Network (RNN) hingegen gibt es zusätzlich Verweise auf vorherige Künstliche Neuronen (Abbildung 4.3 Rechts). Auf diese Weise bekommen RNN eine Art „Gedächtnis“. [23]

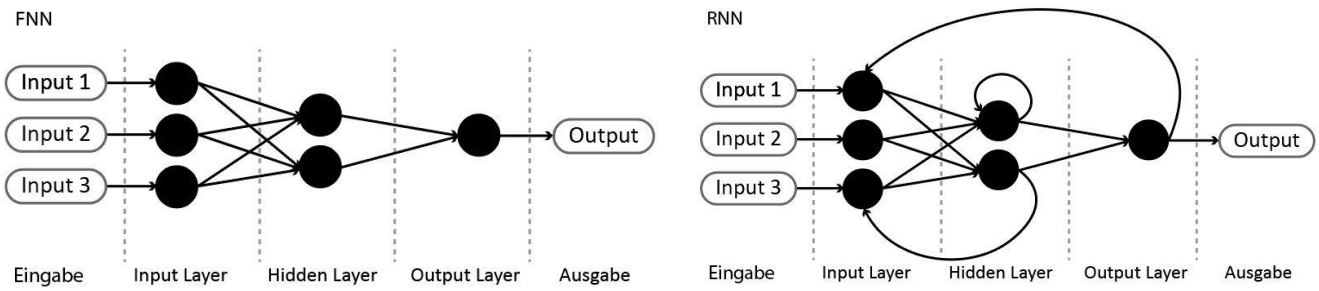


Abbildung 4.3: Links - Grafische Darstellung der Topologie eines Feed-Forward Neural Network. Rechts – Grafische Darstellung der Topologie eines Recurrent Neural Network.

Für die Text Klassifizierung werden üblicherweise Varianten des RNN oder das Convolutional Neural Network (CNN) eingesetzt. Ein Vergleich zwischen den RNN Varianten Long Short-Term Memory (LSTM) und Gated Recurrent Unit und CNN zeigt, dass alle drei Methoden bei kurzen Texten eine ähnlich gute Genauigkeit von um die 90% vorweisen [25].
Ein weiterer Vergleich zwischen LSTM und ein CNN bei der Klassifizierung von Filmkritiken zeigt, dass das CNN ein wenig genauer abschneidet und gleichzeitig weniger rechenintensiv ist [26]. Aufgrund der ähnlich hohen Genauigkeit aller Varianten und die geringere Rechenintensität, wird ein CNN als Kandidat für den Methodenvergleich gewählt.

CNN wurden ursprünglich für den Bereich Computer Vision entwickelt und sind von der Architektur auf das Verarbeiten von Bildern ausgelegt. Selbst einfache Modelle mit nur einem Hidden Layer erzielen jedoch auch bei der Text Klassifizierung sehr gute Ergebnisse [27].
Möchte man ein Bild mit einem KNN verarbeiten, so stellt die Anzahl der Eingabewerte schnell ein Problem dar. Ein kleines Bild von nur 32x32 Pixeln und 3 Farbkanälen, wie es bei Icons üblich ist, hätte bereits über 3000 Eingabewerte, wenn die einfachste Art eines CNN - mit nur einem Neuron im Hidden Layer – verwendet wird. Bei größeren Bildern und/oder einem Netzwerk mit mehreren Neuronen steigt die Anzahl der Eingabewerte enorm. Um die Zahl der Eingabewerte zu verringern werden Abschnitte des Bildes unter anderem durch Faltung zusammengefasst. Diese Vorverarbeitung erweitert das KNN um weitere Schichten, die sich nach dem Input Layer ansiedeln; der Convolutional Layer und der Pooling Layer. [28, 29, 10]

Im Convolutional Layer wird das Bild mit einem Kernel – üblicherweise eine 3x3 Matrix – gefaltet. Der Kernel selbst ist zunächst zufällig und wird im Laufe des Trainings und der Optimierung angepasst. Die Größe der resultierenden Feature Map – auch Activation Map - kann durch die Schrittweite – Stride -, mit der der Kernel das Bild durchläuft, verkleinert werden.
Abbildung 4.4 zeigt wie der Kernel ein 6x6 Pixel Bild mit einer Schrittweite von einem Pixel durchläuft. Die daraus resultierende Feature Map entspricht 4x4 Pixel. Dem entgegen kann Zero Padding eingesetzt werden, um die Ränder des Bildes besser

verarbeiten zu können. Dabei wird die Bild-Matrix mit einem Rand aus Null wert Pixeln umgeben, sodass die eigentlichen Ränder des Bildes bei der Faltung präsenter sind. Da sich das Bild auf diese Weise um zwei Pixel vergrößert, vergrößert sich auch die Feature Map. Bei einer Schrittweite von 1 ist die Feature Map mit Zero Padding gleich groß dem Ursprungsbild ohne Zero Padding. [28, 29, 10]

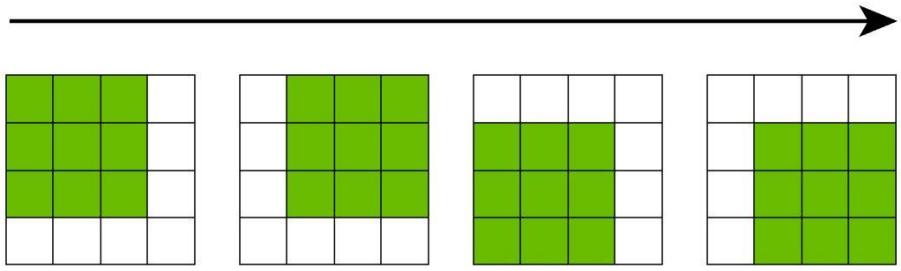


Abbildung 4.4: Grafische Darstellung eines Kernels (Grün), der bei der Faltung das Bild mit einer Schrittweite von einem Pixel durchläuft.

Im nächsten Schritt wird die Feature Map mit der ReLu Funktion aktiviert. Dadurch werden alle negativen Ergebnisse auf 0 gesetzt (Siehe Abbildung 4.5).

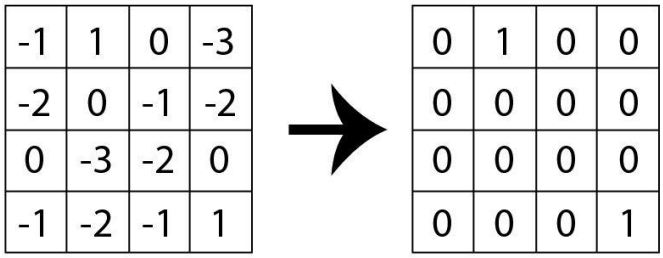


Abbildung 4.5: Matrix von und nach der Aktivierung mit der ReLu Funktion.

Im Pooling Layer wird die Matrix weiter herunterskaliert. Dafür wird eine weitere Faltung vorgenommen, üblicherweise mit einem 2x2 Maximum- oder Durchschnittsfilter Kernel und einer Schrittweite von zwei. Die Matrix wird also je vier Werte auf den jeweiligen Höchstwert (oder Durchschnitt) – und damit um ein Viertel der ursprünglichen Größe - reduziert. [28, 29, 10]
Abbildung 4.6 zeigt beispielhaft eine 4x4 Feature Map, die durch das Pooling auf eine 2x2 Matrix reduziert wird.

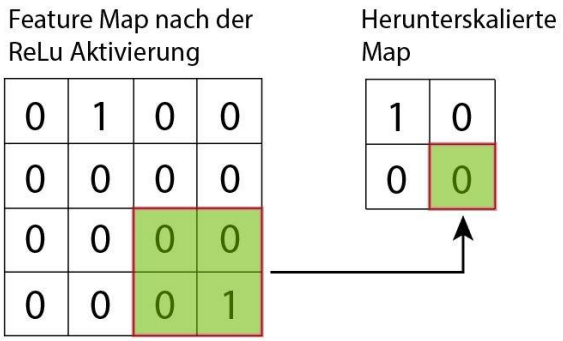


Abbildung 4.6: Darstellung der Herunterskalierung einer Feature Map mittels Max Pooling

Die reduzierten Daten werden dann einem Standard FFN KNN über einen Fully Connected Layer zugeführt. Als Fully Connected Layer wird eine Schicht bezeichnet, wenn jedes Neuron mit allen Neuronen der vorherigen Schicht verbunden ist. Der Output Layer eines CNN besitzt für jedes Ergebnis, das es ermitteln kann, ein Neuron. Anders gesagt, es gibt für jede mögliche Klasse einen separaten Ausgang. Ein CNN kann aus mehreren Convolution, Pooling und Fully Connected Layer bestehen. Erstere beide sind dabei stets zusammenhängend. [28, 29, 10]

Um einen Text zu falten, muss dieser zuerst in Form einer Matrix dargestellt werden. Dafür wird jedes Wort mittels Word Embedding als Vektor dargestellt. Die Vektoren aller Wörter bilden dann die Matrix (Siehe Abbildung 4.7). Der Kernel wird dabei auf eine 3x1 Matrix reduziert, bzw. auf eine 3x3 Matrix in der nur eine Reihe mit anderen Werten als 0 gefüllt ist, damit sich die Faltung immer nur auf die Vektoren eines Wortes beziehen. [27, 30, 10]

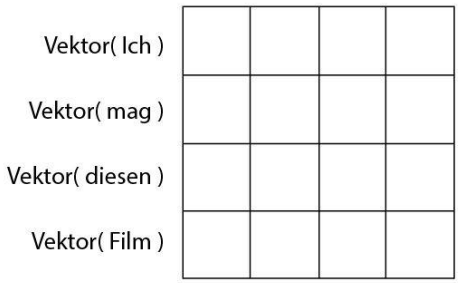


Abbildung 4.7: Grafische Darstellung eines Satzes in Form einer Matrix

CNN erreichen eine hohe Genauigkeit und benötigen durch die Faltung weniger Berechnungen als normale KNN. Dementsprechend erreicht ein CNN eine bessere Leistung. Die hohe Genauigkeit ist jedoch nur mit Hilfe einer sehr großen Menge an Trainings-Datensätzen möglich. Weiterhin ist es äußerst Rechenintensiv. [25, 26, 31]

3.5 Schlüsselwörter

Die Klassifizierung eines Textes anhand von Schlüsselwörtern alleine ist in den meisten Fällen eine eher ungenaue Methode. Durch die Natur der Sprache kann dieselbe Aussage durch viele verschiedene Formulierungen getätigt werden. Eine Formulierung kann dabei ganz andere Wörter enthalten als eine andere.
Die Umstände und Ziele dieser Arbeit machen Schlüsselwörter jedoch zu einem geeigneten Kandidaten für die Klassifizierung. Ziel der Arbeit ist eine effiziente Alternative für sich wiederholende Handlungen, es ist also davon auszugehen, dass sich der Benutzer bei der Formulierung seines Sprachbefehls kurz fassen wird, anstatt ihn unnötig lang zu umschreiben. Auf Grundlage dieser Annahme enthält jedes Intent des SA gewisse Wörter, die einerseits zwangsweise in einem Sprachbefehl vorkommen müssen und andererseits nicht in Sprachbefehlen für andere Intents vorkommen können.
Wenn der Benutzer beispielsweise einen Thread erstellen möchte, wird er zwangsweise das Wort „Thread“ in seiner Anweisung verwenden. Eine Umschreibung wäre zwar möglich, jedoch äußerst unpraktisch und daher unlogisch. Weiter wird das Wort „Thread“ nicht in einer Anweisung für einen anderen Intent, etwa einer IF-Anweisung oder dem Kopieren eines Textes vorkommen. Auch dies wäre irrational und nicht zielführend.

Über Schlüsselwörter kann die Klassifizierung des Textes über eine Reihe von IF-Anweisungen umgesetzt werden und nicht über komplexe Systeme die vergleichsweise viele Berechnungen erfordern. Daher wird weniger Rechenleistung und damit auch weniger Zeit benötigt, um den Sprachbefehl zuzuordnen.
Die Einschränkungen dieser Methode bestehen lediglich daraus, dass der Benutzer seine Anweisung nicht zu umschreibend formulieren und Anweisungen nicht mischen darf.

4 Methoden zur Spracherkennung

Bei der Text Klassifizierung muss das verwendete Modell auf die jeweiligen Klassen trainiert werden. Bei der Spracherkennung, bzw. Speech-to-Text hingegen ist das nicht der Fall. Alle Modelle haben dieselbe Grundfunktion; das Übersetzen von gesprochener Sprache in Text. Daher können vortrainierte State-of-the-Art Modelle und APIs ohne weitere Arbeit verwendet werden.

Ein Paper von 2023 vergleicht die aktuellen Open Source Modelle OpenAIs Whisper[2], Metas wav2vec[3] 2.0, Nvidias NeMo[4] und Googles closed source ASR Service[5]. Für den Vergleich wurden 100 zufällige Aufnahmen von sozialwissenschaftlichen Umfragen verwendet und die Wortfehlerrate bei der Übersetzung gemessen. Die Ergebnisse zeigen, dass Whisper mit 4,7%, bzw. 5,6% die geringste Fehlerquote aufweist. [32]

4.1 Faster Whisper

Ein eigener Test wurde mit der Whisper Neuimplementierung Faster Whisper[5](FW) durchgeführt. Laut eigenen Angaben arbeitet FW bis zu viermal schneller als das Original, bei gleicher Genauigkeit. FW ermöglicht eine schnellere Verarbeitung mit Hilfe der Grafikarte und der Nvidia Technologie CUDA. Hierfür ist eine Nvidia Grafikkarte nötig. Der Test wurde mit einer Nvidia GTX 1070 Grafikkarte durchgeführt.
Der Test, mit einigen Beispiel-Befehlen, die so im fertigen Projekt vorkommen könnten, lieferte folgende Ergebnisse:

[2] https://openai.com/index/whisper
[3] https://ai.meta.com/research/impact/wav2vec/
[4] https://www.nvidia.com/de-de/ai-data-science/products/nemo/
[5] https://cloud.google.com/speech-to-text?hl=de
[6] https://github.com/SYSTRAN/faster-whisper

	Tiny	Small	Medium	Large-v3
Endlosschleife	0,34s	0,42s	0,89s	1,45s
If-Abfrage A größer B	Nicht erkannt	Nicht erkannt	1,02s	1,53s
Erstelle mir eine Endlosschleife	0,35s	0,52s	1,05s	1,72s
Erstelle eine If-Abfrage	Nicht erkannt	0,55s	0,96s	1,55s
Erstelle eine For-Schleife von 1 bis 100	0,35s	0,58s	1,12s	1,69s
Erstelle mir eine If-Abfrage mit X kleiner 3	Nicht erkannt	0,65s	1,10s	1,78s

Tabelle 2: Verarbeitungszeit in Sekunden von Faster Whisper nach vortrainierten Modell und Beispielsatz.
Die Zeiten entsprechen dem Mittel aus fünf Durchläufen, gerundet auf zwei Nachkommastellen.

Wie in Tabelle 2 zu sehen, ist die Verarbeitungszeit geringer, je kleiner das Sprachmodell ist. Gleichzeitig sinkt jedoch die Genauigkeit der Übersetzung. Es muss also zwischen Genauigkeit und Geschwindigkeit abgewogen werden. Der Test zeigte weiterhin, dass die Modelle „Small“ und „Tiny“ Projektspezifische Wörter wie „For Schleife“ oder „If Abfrage“ nicht verlässlich erkennen können. Bei den Modellen „Medium“ und „Large-v3“ gab es wenige Probleme.

4.2 Insanely Fast Whisper

Die Whisper Neuimplementierung Insanely Fast Whisper[7] verspricht laut eigenen Angaben eine rund 20 fache Geschwindigkeit gegenüber dem original Whisper Modell und eine etwa vierfache Geschwindigkeit gegenüber FW. Eine Wiederholung des mit FW durchgeführten Tests mit diesem Modell zeigt jedoch keinen merkbaren Geschwindigkeitszuwachs gegenüber der FW Lösung. Das liegt sehr wahrscheinlich daran, dass die Benchmarks der Insanely Fast Whisper Entwickler mit einer Nvidia A100 – 80GB[8] Grafikkarte durchgeführt wurden. Eine Grafikkarte, die speziell für den Bereich KI und Datenanalyse ausgelegt ist. Weiterhin zeigte sich, dass die vorherige Methode – FW – einfacher umzusetzen ist.

[7] https://github.com/Vaibhavs10/insanely-fast-whisper
[8] https://www.nvidia.com/de-de/data-center/a100/

5 Konzeption eines Code Editors mit Sprachassistenten

In Kapitel 2 wurden die Funktionsweisen von Code Editoren und SA untersucht. Weiter wurden in Kapitel 3 und 4 geeignete Methoden für die Umsetzung der Spracherkennung und der Text Klassifizierung ausgearbeitet. Nun gilt es mit Hilfe dieser Erkenntnisse ein Code Editor mit SA zu konzipieren, der das Erstellen von Code erleichtert, indem gewisse Programmieraufgaben über Sprachbefehle automatisch erstellt werden können.

5.1 Wahl der Methode für die Spracherkennung

Die in Kapitel 4 durchgeführten Tests haben gezeigt, dass es mit einer Grafikkarte, die nicht speziell auf den Bereich KI und Datenanalyse ausgelegt ist, keine merklichen Geschwindigkeitsunterschiede bei den Whisper Neuimplementierungen FW und Insanely Fast Whisper gab. Weiterhin war das FW Modell einfacher zu implementieren und bietet mehr Einstellungsmöglichkeiten für die Hardwarebeschleunigung.
Aus diesen Gründen wird FW als Methode für die Spracherkennung gewählt.

Aufgrund der Ungenauigkeit der Sprachmodelle „Tiny“ und „Small“, und dem großen Unterschied in der Verarbeitung bei dem Modell „Large-v3“, wird das Sprachmodell „Medium“ für die Umsetzung gewählt.

5.2 Wahl der Methode zur Text Klassifizierung

CNN, RF und NB zeichnen sich bei umfangreichen Trainingsdaten und guter Optimierung durch eine hohe Genauigkeit aus. Das CNN benötigt dafür jedoch sehr große Mengen an Trainingsdaten und ist, ebenso wie RF, rechenintensiver als die anderen Methoden. Weiterhin sind beide Methoden durchaus komplex und für die Anwendung in dieser Arbeit zu übertrieben.
NB erreicht eine ähnlich hohe Genauigkeit, gilt jedoch als einfach zu implementieren und benötigt weniger Rechenleistung. Jedoch benötigt NB ebenfalls eine gewisse Menge an Trainingsdaten, um verlässliche Ergebnisse zu liefern.

In einem Test wurde für jedes Intent eine Reihe von Sprachbefehlen aufgenommen, ursprünglich um herauszufinden welche Vorverarbeitungen für die Klassifizierung nötig sind. Dabei hat sich auch die Annahme aus Abschnitt 3.5 bestätigt; die Befehle unterscheiden sich nur durch einige wenige Wörter. Das bedeutet auch die NB Methode würde hauptsächlich anhand dieses wenigen Wörter die Klassifizierung vornehmen. Alle anderen Wörter wären dabei sogar potenziell hinderlich.

Die Klassifizierung durch eben jene wenigen Schlüsselwörter direkt ist daher die bevorzugte Methode für diese Arbeit. Die Umsetzung ist vergleichsweise simpel, die Rechenintensität ist gering und das „Training“ dieser Methode – das Ausarbeiten der Schlüsselwörter – bedarf nicht viel Arbeit.

5.3 Konzeption des Sprachassistenten

Der SA orientiert sich an den Aufbau aus Abschnitt 2.2. Es werden jedoch Anpassungen vorgenommen, um ein effizienteres Arbeiten zu ermöglichen. Abbildung 5 zeigt den Aufbau des Assistenten in diesem Projekt.

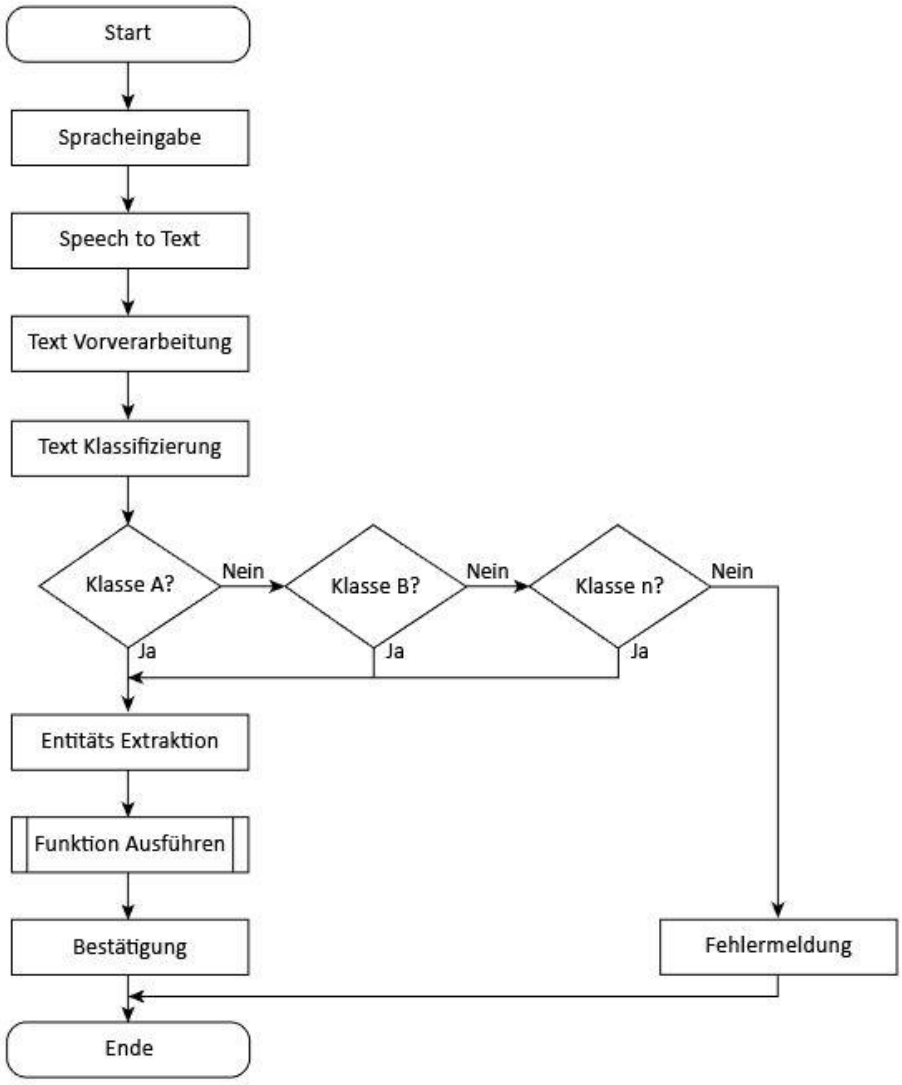


Abbildung 5: Programmablaufplan des zu entwickelnden Sprachassistenten

5.3.1 Spracheingabe

Ein konventioneller SA lauscht dauerhaft mit und wartet auf ein Aktivierungswort. Das kann einerseits zu Datenschutzfragen führen, da – wenn auch nur kurz - effektiv alles in Reichweite des Mikrofons automatisch aufgenommen wird. Andererseits bedeutet das auch, dass der Assistent dauerhaft arbeitet und nach dem Aktivierungswort

scannt. Des Weiteren ist das Prüfen des Aktivierungswortes ein zusätzlicher Arbeitsschritt.
Um die Spracheingabe möglichst effizient zu gestalten, wird das Aktivierungswort durch eine Lautstärkenaktivierung ersetzt. Weiterhin hat der Nutzer die Möglichkeit den SA per Button ein- und auszuschalten. Der SA lauscht so nicht mehr dauerhaft mit, sondern nur, wenn der Nutzer ihn bewusst einschaltet. Anstatt die Aufnahme dauerhaft nach einem Aktivierungswort zu scannen, wird lediglich der Lautstärkepegel überwacht und bei Überschreitung einer gewissen Lautstärke die Aufnahme gestartet.

Bei aktiviertem SA sind drei Faktoren für die Aufnahme ausschlaggebend; ein Schwellwert für den Start der Aufnahme, ein Schwellwert für den Stopp der Aufnahme und ein Pufferzeitraum für den Stopp der Aufnahme.
Ersterer Schwellwert legt eine minimale Lautstärke fest, ab der die Aufnahme gestartet wird. Dies dient dazu, dass die Aufnahme nur dann startet, wenn der Nutzer auch tatsächlich eine Spracheingabe tätigt.
Letzterer Schwellwert sorgt dafür, dass die Aufnahme beendet wird, sobald eine gewisse Lautstärke unterschritten wird. So wird die Aufnahme gestoppt, wenn der Nutzer zu Ende gesprochen hat.
Der Pufferzeitraum verzögert den Stopp der Aufnahme nach Unterschreiten des Schwellwertes. Dies ist nötig, um zu verhindern, dass die Aufnahme bei kleineren Wortpausen abbricht.
Alle drei Faktoren können innerhalb des Code Editors vom Nutzer angepasst werden. Da das FW Model mit Audiodateien arbeitet, wird die Aufnahme in einer temporären Datei gespeichert. Nach der Verarbeitung wird die Datei wieder gelöscht.

5.3.2 Speech to Text

Wie in Abschnitt 5.1 begründet, wird das Fast Whisper Modell mit dem Sprachmodell „Medium“ für die Transkription verwendet. Das Modell wird in das Programm integriert, nutzt also keinen Cloud-Service, wodurch das Programm auch ohne Internetverbindung funktioniert. Die Transkription auf den eigenen Rechner durchzuführen kann jedoch rechenintensiv sein. Um ein möglichst schnelles Ergebnis zu erhalten, gibt es daher die Option diesen Arbeitsschritt von der Grafikkarte durchführen zu lassen, sofern eine Nvidia Grafikkarte vorhanden ist. Hierfür wird die Programmierschnittstelle CUDA verwendet.

5.3.3 Text Vorverarbeitung

Der erzeugte Text wird „bereinigt“, um die Effektivität der Text Klassifizierung zu erhöhen. Die Vorverarbeitung enthält Lowercasing, Sonderzeichenentfernung, Wortkorrektur, Stemming und Tokenisierung.

Lowercasing
Beim Lowercasing wird der Text in Kleinbuchstaben umgewandelt. Auf diese Weise kann es später nicht zu ungewollten Unterscheidungen kommen, etwa wenn ein Wort am Satzanfang steht.

Sonderzeichenentfernung
In diesem Schritt wird die Interpunktion und alle Sonderzeichen aus dem Satz entfernt. Hierdurch wird verhindert, dass ein Wort mit einem Komma oder Punkt als eigenständiges Wort angesehen wird.

Tokenisierung
Bei der Tokenisierung wird der Text anhand der Leerzeichen in einzelne Wörter zerlegt. Der Text, ein String Objekt, wird so in einen Vektor aus String Objekten umgewandelt.

Stemming
Durch das Stemming werden Wörter auf ihren Stamm gekürzt. Das Verfahren ist sehr simpel, da es den Stamm wörtlich „heraustrennt“. Es wird nicht der tatsächliche Wortstamm gesucht, sondern das Ende des Wortes so weit gekürzt, dass es mit anderen gekürzten Variationen übereinstimmt. Das Wort „Katzen“ wird so zu „Katze“ oder „Katz“ gekürzt. „Gutes“ und „Besseres“ wird jedoch zu „Gut“ und „Besser“, obwohl beide Wörter den gleichen Wortstamm haben. Dadurch ist Stemming kein sehr genaues Verfahren, jedoch weniger rechenintensiv wie etwa der Lemmatization, bei der das Wort auf seinen tatsächlichen Wortstamm gebracht wird.

5.3.4 Text Klassifizierung
Wie in 5.2 begründet, wird für die Text Klassifizierung eine Abfrage nach Schlüsselwörtern verwendet. Es werden Schlüsselwörter ausgearbeitet, mit denen jedes Intent eindeutig identifiziert werden kann. In einem weiteren Schritt wird eine Reihe von Beispielanweisungen aufgenommen, in denen die Schlüsselwörter vorkommen. Anstelle einer Wortkorrektur, werden häufiger falsch übersetzte Schlüsselwörter (Zum Beispiel „Ifa Frage“ anstatt „If Abfrage“) ebenfalls den Schlüsselwörtern hinzugefügt. Die Schlüsselwörter durchlaufen anschließend die Text Vorverarbeitung und gegebenenfalls entstandene Duplikate werden entfernt.

5.3.5 Intents
Für den SA sind verschiedene Intents geplant, um das Programmieren zu erleichtern.

- Das Navigieren zu einer bestimmten Zeile
- Das Kopieren und Löschen von markiertem Text oder einer bestimmten Zeile
- Das Einfügen von Text
- Das Erstellen einer If-Abfrage
- Das Erstellen einer For-Schleife

- Das Erstellen einer While-Schleife
- Das Erstellen eines Match Case
- Das Erstellen einer Exception
- Das Erstellen einer Endlosschleife
- Das Erstellen eines Timers
- Das Erstellen eines Threads
- Das Erstellen eines Print Befehls
- Das Erstellen eines Inputs

Zudem ist die Option geplant jedes Intent an der aktuellen Cursor Position oder in einer angegebenen Zeile ausführen zu lassen.

5.3.6 Entität Extraktion

Bei der Entität Extraktion wird der Text nach zusätzlichen Informationen durchsucht. Für jedes Intent wird nach Entitäten für eine Zeilenangabe gesucht. Dies sind die Schlüsselwörter „Zeile“ oder „Reihe“ und eine darauf folgende Zahl. Enthält ein Text zum Beispiel einen Abschnitt „in Zeile 15“, dann wird diese Zeilenangabe extrahiert und für die Ausführung des Intents genutzt.
Das Navigieren zu einer bestimmten Zeile benötigt eine solche Zeilenangabe, um erfolgreich zu sein.
Für die Erstellung einer If-Abfrage und While Schleife werden exemplarisch nach weiteren Entitäten gesucht; nach einem Operator und zwei Vergleichswerten - beispielsweise „a größer 17“ -, um dem Benutzer beim Erstellen dieser Strukturen mehr Möglichkeiten zu geben.

5.3.7 Funktion Ausführen

Für jedes Intent wird eine Funktion erstellt, die die jeweilige Aktion ausführt. Dabei werden die aus der Entität Extraktion gewonnenen Informationen geprüft und darauf basierend eine Aktion ausgeführt. Wird bei einer Anweisung eine Zeilenangabe gemacht, wird die Aktion an der entsprechenden Zeile ausgeführt. Ohne Zeilenangabe, wird die Aktion an der aktuellen Position des Cursors, bzw. in dessen Zeile ausgeführt. Die Aktion zum Navigieren in eine bestimmte Zeile wird ohne Zeilenangabe nicht ausgeführt.
Wenn bei der Erstellung der If-Abfrage keine Vergleichswerte und kein Operator angegeben werden, springt der Cursor an die entsprechende Stelle, um eine manuelle Eingabe zu beschleunigen.

5.3.8 Bestätigung und Fehlermeldung

Wurde der Befehl ausgeführt, gibt es eine Bestätigung. Anstelle einer Sprachausgabe, erscheint eine kurze Pop-Up Nachricht, um die erfolgreiche Ausführung mitzuteilen. Auf die gleiche Weise wird mitgeteilt, wenn eine Anweisung nicht zugeordnet werden kann oder es andere Probleme gab. Je nach

Benachrichtigung sind Pop-Up Nachrichten in unterschiedlichen Farben geplant, um den Benutzer schnell den jeweiligen Status zu vermitteln, auch ohne dass der Text der Nachricht gelesen wird.

5.4 Konzeption des Code Editors

Der Code Editor wird über eine Grafische Oberfläche gestaltet (Siehe Abbildung 6) und über einige Funktionen verfügen. Es ist wichtig, dass der Texteditor, in dem der Benutzer später seinen Code schreibt, programmatisch bedienbar ist, damit der SA sinnvoll seine Anweisungen ausführen kann. Der Texteditor wird das programmatische Einfügen und Auslesen von Inhalt und das Auswählen von Zeilen, sowie die Position innerhalb einer Zeile unterstützen. Der Texteditor wird über eine Zeilenanzeige verfügen, eine automatische Einrückung unterstützen und eine automatische Vervollständigung für Funktionszeichen wie Klammern und Anführungsstriche enthalten, um einen effizienten Arbeitsfluss zu unterstützen.
Es wird die Möglichkeit geben Python Code auszuführen, um das Programmieren effizienter zu gestalten. Dafür wird die grafische Oberfläche einen weiteren Texteditor enthalten, der die Ausgaben des ausgeführten Programmes anzeigt, und ein Eingabefenster über welches Eingaben an das Programm gesendet werden können.
Weiter ist für den Code Editor eine Menü Leiste geplant, über die eine Dateiverwaltung und Optionen für den SA erreicht werden und der SA De-/Aktiviert werden kann. Die Dateiverwaltung enthält das Erstellen eines neuen Projektes, das Öffnen von Python Dateien und das Speichern des Projektes. Die Optionen für den SA enthalten die Auswahl des zu verwendenden Aufnahmegerätes, eine Auswahl des Hardwarebeschleunigers für die Sprache zu Text Übersetzung und eine Anpassung der Schwellwerte für die Sprachaktivierung des SA.

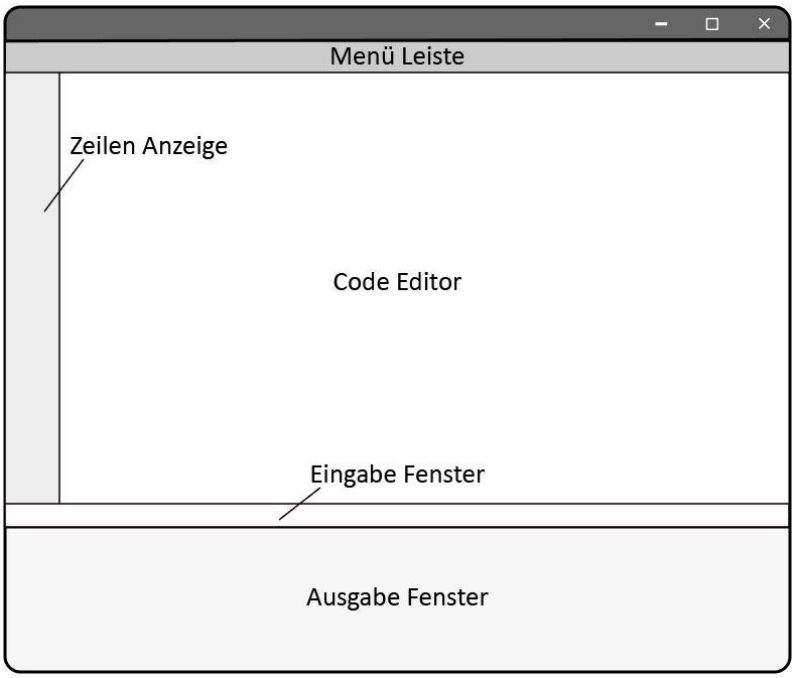


Abbildung 6: Konzept der grafischen Oberfläche des Code Editors

5.5 Anforderungen und Werkzeuge

Python bietet viele Bibliotheken[9] mit denen die Umsetzung einiger Schritte erleichtert werden kann. Folgende Bibliotheken werden für diese Arbeit verwendet:

Die grafische Oberfläche wird mit Hilfe von Tkinter[10] erstellt. Tkinter gehört zum Lieferumfang von Python und bietet eine einfache Möglichkeit grafische Oberflächen zu erstellen. Das enthaltene Text Widget bietet verschiedene Möglichkeiten der programmatischen Bedienung. Unter anderem kann Inhalt mit genauen Positionsangaben eingefügt, gelöscht und ausgelesen werden und auch die Position des Cursors kann programmatisch ausgelesen und gesetzt werden. Damit eignet sich das Text Widget gut für die Umsetzung dieser Arbeit.
Für die Darstellung der Zeilennummern das Modul Tklinenums[11] verwendet. Dieses Modul liefert eine spezielle Text Box, mit der die Zeilennummern eines Tkinter Text Widgets ausgelesen und angezeigt werden können. Das Ausführen des Projektes im Code Editor wird über das Subprocess[12] Modul umgesetzt. Dieses Modul ermöglicht das Erstellen eines neuen Prozesses und das Erstellen einer Verbindung für die Eingabe, Ausgabe und/oder Fehlerausgabe.

[9] Bibliothek, Modul und Package werden im Kontext von Fremdcode synonym verwendet.
[10] https://docs.python.org/3/library/tkinter.html
[11] https://pypi.org/project/tklinenums/
[12] https://docs.python.org/3/library/subprocess.html

Pyaudio[13] ist eine Bibliothek für Audio Ein- und Ausgaben. Sie ermöglicht unter Anderem das Anzeigen von Audio-Eingabegeräten und das Aufnehmen von Audiodaten eines gewählten Eingabegeräts. Daher wird Pyaudio für eben diese Funktionen verwendet.
Um die Aufnahme für die weitere Verarbeitung zwischen zu speichern, wird das Wave[14] Modul verwendet, ein benutzerfreundliches Interface, um .wav Dateien zu öffnen und erstellen. Um das Starten und Beenden der Aufnahme über die Lautstärke zu ermöglichen, muss die Lautstärke gemessen werden. Hierfür kommt Audioop[15] zum Einsatz. Ein Modul das nützliche Operationen an Audio Fragmente ermöglicht, darunter das Messen der Lautstärke.
Das Time[16] Modul enthält verschiedene zeitbezogene Funktionen; dadurch lässt sich eine einfache Stoppuhr erstellen, die für den Timeout der Sprachaufnahme zuständig ist. Um die temporäre Audioaufnahme nach der Verarbeitung wieder zu löschen, wird das Os[17] Modul eingesetzt. Dieses ermöglicht das Ausführen verschiedener betriebssystemspezifischer Funktionen.
Für das Übersetzen der Aufnahme in Text, kommt – wie in Abschnitt 5.1 begründet - das FW Modul zum Einsatz.

Für die Text Vorverarbeitung kommen zwei Module zum Einsatz. Das Re[18] Modul ermöglicht Operationen mit Hilfe von regulären Ausdrücken. Zum Beispiel das Suchen nach bestimmten Mustern im Text. Das Stem.Snowball Modul aus der NLTK[19] Bibliothek ermöglicht das Stemmen von Wörtern in Verschiedenen Sprachen.

[13] https://pypi.org/project/PyAudio/
[14] https://docs.python.org/3/library/wave.html
[15] https://docs.python.org/3/library/audioop.html
[16] https://docs.python.org/3/library/time.html
[17] https://docs.python.org/3/library/os.html
[18] https://docs.python.org/3/library/re.html
[19] https://www.nltk.org/

6. Implementierung des Prototypen

Nachdem die Konzipierung des Code Editors und des SA abgeschlossen ist, gilt es nun diese zu implementieren. Hierfür wird PyCharm[20] für die Codeerstellung und Github[21] als Versionskontrolle verwendet. Das Projekt wird in Python 3.10 umgesetzt.

6.1 Implementierung des Code Editors

Im ersten Schritt wird der Code Editor entwickelt. Es wird die grafische Oberfläche erstellt, die benötigten Funktionalitäten hinzugefügt und zuletzt drei Fenster für die Optionen des SA implementiert.

6.1.1 Hauptfenster

Zuallererst wird das Hauptfenster der Entwicklungsumgebung erstellt, da alles weiter hierauf aufbaut (Siehe Abbildung 7).

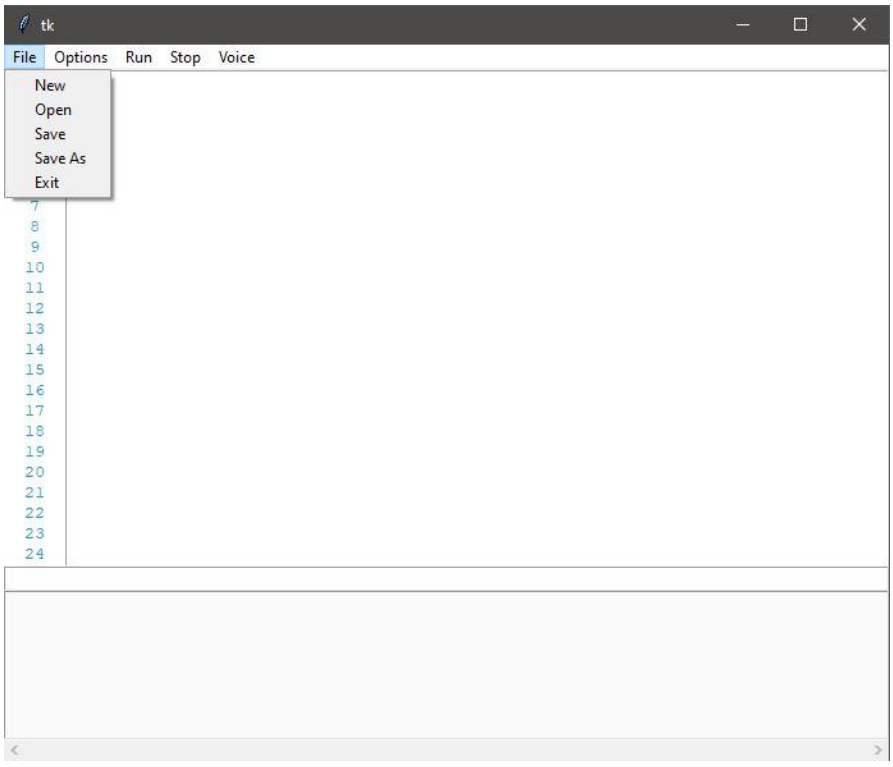


Abbildung 7: Das Hauptfenster der Entwicklungsumgebung mit geöffnetem „File“ Menü.

Für das Hauptfenster wird die Klasse MainGUI erstellt und als Tkinter root Fenster definiert, dem alle weiteren Elemente hinzugefügt werden (Siehe Listing 2.1)

[20] https://www.jetbrains.com/de-de/pycharm/

[21] https://github.com/

```
class MainGUI(tkinter.Tk):
    def __init__(self):
        tkinter.Tk.__init__(self)
```
Listing 2.1: Erzeugen einer Klasse als Tkinter root Fenster.

Dem Hauptfenster werden zwei Text Widgets, ein Entry Widget und eine Menüleiste hinzugefügt.
Ein Text Widget dient als Code Editor in dem programmiert wird, das zweite Text Widget dient später als Ausgabekonsole. Über das Entry Widget werden Eingaben in das ausgeführte Programm ermöglicht.
Die Menüleiste erhält ein Drop Down Menü „File“, mit den Unterpunkten „New“, „Open“, „Save“, „Save As“ und „Exit“; ein zweites Drop Down Menü „Options“ mit den Unterpunkten „Microphone“, „Recording Settings“ und „Hardware Accelerator“; und die Menüpunkte „Run“, „Stop“ und „Voice“. Listing 2.2 zeigt beispielhaft das Hinzufügen einer Menüleiste mit dem „File“ Menü und dessen Unterpunkten. Über dem Parameter „label“ wird dem Menüpunkten ein Name zugewiesen. Der Parameter „command“ verweist auf eine Funktion, die bei einem Klick auf den Menüpunkt aufgerufen wird.

```
menu_bar = Menu(self)
file_bar = Menu(menu_bar, tearoff=0)
file_bar.add_command(label="New", command=new_file)
file_bar.add_command(label="Open", command=open_file)
file_bar.add_command(label="Save", command=save)
file_bar.add_command(label="Save As", command=save_as)
file_bar.add_command(label="Exit", command=exit)
menu_bar.add_cascade(label="File", menu=file_bar)
```
Listing 2.2: Erstellung einer Menüleiste und Hinzufügen eines Menüpunktes mit Unterpunkten.

Weiterhin wird neben dem Code Editor ein TkLineNumbers Widget aus der gleichnamigen Bibliothek hinzugefügt, um die Zeilenzahl des Code Editors anzuzeigen. Neben den üblichen Vorteilen solch einer Anzeige, hat sie eine zusätzliche Relevanz für den SA, da die Befehle auch Zeilenangaben enthalten können. Wie in Listing 2.3 zu sehen, wird das Widget ähnlich einem Tkinter Text Widget erstellt. Der zusätzliche Parameter „textwidget“ gibt an von welchen Text Widget die Zeilen gezählt werden sollen. An dem Code Editor wird ein Event Listener gebunden, durch den die Zeilenanzahl mit jeder Änderung im Code Editor aktualisiert wird.

```
linenums = TkLineNumbers(master=self.editor_frame,
                         textwidget=self.editor, justify="center",
                         colors=("#2197db", "#ffffff"), bd=0)
linenums.pack(fill="y", side="left")
self.editor.bind("<<Modified>>",
                 lambda event: self.after_idle(linenums.redraw), add=True)
```

Listing 2.3: Erstellung eines TkLineNumbers Widget aus der tklinenums Bibliothek

6.1.2 Funktionen der Entwicklungsumgebung

Im nächsten Schritt werden die Dateiverwaltung und das Ausführen des Codes implementiert.

Das Tkinter.filedialog Modul stellt die Methoden „asksaveasfilename“ und „askopenfilename“ zur Verfügung, die einen Dialog zum Öffnen, bzw. Speichern von Dateien erzeugen. Genauer gesagt werden über die Dialoge die entsprechenden Dateipfade ausgewählt. Das Öffnen, bzw. Speichern findet in einem weiteren Schritt statt. Listing 3.1 zeigt die Funktion, die für das Öffnen einer Datei zuständig ist. Über die „askopenfilename“ Funktion wird ein entsprechender Dialog für die Dateiauswahl geöffnet. Über den Parameter „filetypes“ kann bestimmt werden, welche Dateitypen geöffnet werden dürfen. In diesem Fall sind nur Python Dateien erlaubt.

Da bei Abbruch des Dialogs kein gültiger Dateipfad entsteht, wird dieser nochmals geprüft.

Über Pythons open() Funktion wird die Datei tatsächlich geöffnet und der Inhalt in den zuvor geleerten Text Widget des Code Editors kopiert. Das Text Widget der Ausgabekonsole wird ebenfalls geleert und der Dateipfad wird für zur weiteren Verwendung – zum Beispiel dem Speichern - in eine Klassenvariable gespeichert.

```
def open_file():
    path = askopenfilename(filetypes=[('Python Files', '*.py')])
    if path.endswith('py'):
        with (open(path, 'r') as file):
            code = file.read()
            self.editor.delete("1.0", END)
            self.editor.insert("1.0", code)
            console.delete("1.0", END)
            self.file_path = path

    else:
        print("Fehlermeldung: Es handelt sich nicht um eine Python Datei.")
```

Listing 3.1: Funktion zum Öffnen einer Datei über einen Filedialog

Für das Ausführen des Codes wird ein neuer Prozess gestartet. Listing 3.2 zeigt die Funktion, die bei der Betätigung des „Run“ Buttons ausgeführt wird. Der aktuelle Code wird zuerst gespeichert, danach wird die Popen() Funktion der Subprocess Bibliothek genutzt, um den gespeicherten Code in einem neuen Prozess

auszuführen. Der erste Parameter gibt die Datei an, mit der der neue Prozess gestartet werden soll. „self.file_path“ gibt den Dateipfad der aktuellen Datei an. Die Parameter „stdin“, „stdout“ und „stderr“ erlauben ein Auslesen der Ausgaben und Fehler des Prozesses und das Senden von Eingaben. Der Parameter „shell“ wird auf True gesetzt, wodurch der Prozess über die Shell des Betriebssystems ausgeführt wird. Um den Code Editor nicht zu blockieren, solange der Code ausgeführt wird, wird die „run_code“ Funktion in einem eigenen Thread aufgerufen.

```
def run_code():
    save()
    self.process = subprocess.Popen(self.file_path, bufsize=0,
                                    stdin=subprocess.PIPE,
                                    stdout=subprocess.PIPE,
                                    stderr=subprocess.PIPE, shell=True)
    o = threading.Thread(target=read_output)
    o.daemon = True
    o.start()
```

Listing 3.2: Funktion für die Ausführung einer Datei in einem neuen Prozess

In einem zweiten Thread wird die Funktion „read_output“ aufgerufen, die die Ausgaben des Prozesses kontinuierlich auslesen und in das Ausgabe Text Widget der GUI schreibt, solange der Prozess existiert (Siehe Listing 3.3).
Die „poll()“ Funktion des Prozesses gibt an, ob der Prozess noch existiert. Solange er existiert, liest die Funktion die Ausgaben und Fehler des Prozesses.
Die Attribute „stdout“ und „stderr“ enthalten die Ausgaben und die Fehlermeldungen des Prozesses. Diese werden nacheinander – sofern vorhanden – in die Ausgabe Konsole des Code Editors geschrieben. Da der Prozess Byte Objekte als Ausgaben liefert, werden diese bei dem Schreiben in die Ausgabe Konsole in Text übersetzt.

```
def read_output():
    while self.process.poll() is None:
        for out in self.process.stdout:
            if out:
                console.insert(END, out.decode().strip() + "\n")
                console.see(END)
                print(out)
        for err in self.process.stderr:
            if err:
                console.insert(END, err.decode().strip() + "\n", 'error')
                console.see(END)
                print(err)
```

Listing 3.3: Eine Funktion für das Auslesen eines Prozesses.

Die in der Dokumentation der Subprocess empfohlenen „Popen.communicate()“ Funktion wird nicht für die Kommunikation verwendet, da diese dazu führt, dass die Kommunikation nach einer Eingabe geschlossen wird. Das Auslesen der Ausgabe soll jedoch dauerhaft stattfinden und auch sollen mehrere Eingaben möglich sein.

Für die Eingabe wird ein Event Listener für die Enter-Taste an das Entry Widget gebunden. Sobald in dem Widget die Enter-Taste gedrückt wird, wird eine Funktion ausgelöst, die den enthaltenen Text an den Prozess gesendet (Siehe Listing 3.4). Über die „stdin.write()“ Funktion wird die Eingabe an den Prozess gesendet. Da der Prozess auch für die Eingabe mit Byte Objekten arbeitet, wird der Text des Entry Widgets zuvor in ein solches übersetzt. Wird vom Prozess, keine Eingabe erwartet, wird diese verworfen.

```
def input_def(content):
    try:
        self.process.stdin.write(bytes(content + '\n', 'utf-8'))
        self.process.stdin.flush()
        console.insert(END, content + "\n")
    except OSError:
        pass
```

Listing 3.4: Senden einer Eingabe an den Prozess „process“.

Um den ausgeführten Code zu beenden, wird ebenfalls die „Popen()“ Funktion verwendet (Siehe Listing 3.5). Über die Anweisung „taskkill“ kann der Prozess anhand seiner ID beendet werden. Die Prozess ID wird dabei direkt aus dem Prozess ausgelesen.

```
def stop():
    try:
        subprocess.Popen("taskkill /F /T /PID %i" % self.process.pid,
                         shell=True)
    except Exception:
        pass
```

Listing 3.5: Eine Funktion zum Beenden eines Prozesses über dessen ID.

6.1.3 Optionsfenster

Für die Mikrofon Auswahl, die Auswahl der Hardware Beschleunigung und den Aufnahme Optionen wird jeweils eine Klasse mit einem Unterfenster erzeugt. Listing 4.1 zeigt beispielhaft die Erstellung der Klasse der Hardware Beschleunigung Option. Über „Toplevel.__init__()“ wird die Klasse als Toplevel Fenster initialisiert. Anschließend wird der Fokus auf das Fenster gelegt und die Größe des Fensters auf 300x300 Pixel gesetzt. Neben den spezifischen Inhalten, verfügt jedes Fenster über einen Bestätigen und Abbrechen Button.

```
class HardwareWindow(Toplevel):

    def __init__(self, master):
        Toplevel.__init__(self, master)
        self.focus_set()
        self.geometry("300x300")
```

Listing 4.1: Erstellung der Klasse "HardwareWindow" und Initialisierung als Tkinter Toplevel Fenster

Die Auswahloptionen der Hardware Beschleunigung sind in Form eines Dictionarys mit zwei String Werten gespeichert. Der erste Wert sieht wie folgt aus: „CPU – int8“: [„cpu“, „int8“].
In dem Fenster wird ein Listbox Widget - eine Auswahl Liste - erzeugt, die mit den Auswahloptionen gefüllt wird (Siehe Listing 4.2). Mit Hilfe einer For-Schleife wird durch die in der Variable „hardware_dict“ gespeicherten Auswahloptionen iteriert und jede Option der Listbox hinzugefügt. Anschließend wird die Liste in dem Fenster positioniert.

```
hardware_list = Listbox(self)
for x, y in enumerate(hardware_dict):
    hardware_list.insert(x+1, y)
hardware_list.pack(padx=10, pady=10, fill=tkinter.BOTH, expand=True)
```

Listing 4.2: Erstellung einer Listbox mit Inhalt aus der „hardware_dict“ Variable

Beim Schließen des Fensters durch den Bestätigung Button werden die gewählten Werte zur weiteren Verarbeitung übergeben. Zusätzlich wird im Hauptfenster eine Funktion ausgeführt, die nach der Implementierung des SA dafür verantwortlich sein wird, dass die gewählte Option übernommen wird.

Das Optionsfenster für die Mikrofon Auswahl wird auf dieselbe Weise erstellt, jedoch wird der die Listbox mit den vorhandenen Aufnahmegeräten des Computers gefüllt. Eine Liste mit diesen Geräten kann mit Hilfe der Pyaudio Bibliothek erstellt werden (Siehe Listing 4.3).

```
def check_devices():
    self.connected_devices = {}
    if self.py_audio is None:
        self.py_audio = pyaudio.PyAudio()
    else:
        self.py_audio.terminate()
        self.py_audio = pyaudio.PyAudio()
    device_count = self.py_audio.get_device_count()
    for i in range(device_count):
        # Get the device info
        device_info = self.py_audio.get_device_info_by_index(i)
        # Check if this device is a microphone (an input device)
        if device_info.get('maxInputChannels') > 0:
            self.connected_devices[device_info.get('name')] =
```

Listing 4.3: Auslesen der aktuellen Aufnahmegeräte über Pyaudio

Zuerst wird ein Pyaudio Objekt erstellt. Ist ein Objekt bereits vorhanden, wird es neu erstellt. So wird sichergestellt, dass die Liste der Aufnahmegeräte aktuell ist. Über die „get_device_count()“ Funktion wird die Anzahl der gefundenen Audiogeräte ermittelt. Mit Hilfe dieser wird dann in einer Schleife über die „get_device_info_by_index()“ Funktion die Information jedes gefundenen Gerätes ausgelesen. Es werden die maximalen Input Channels abgefragt, um zu ermitteln, ob ein Gerät ein Eingabegerät ist. Zuletzt werden der Name und die ID jedes Eingabegerätes in einem Dictionary gespeichert.

Für die Aufnahme Optionen werden anstelle einer Listbox drei Slider erstellt. Je einer für die Schwellwerte der Aktivierung und Deaktivierung der Audioaufnahme und einer für die Verzögerung bis zum tatsächlichen Aufnahmestopp. Listing 4.4 zeigt beispielhaft die Erstellung des Sliders für den Schwellwert zum Stopp der Aufnahme.

```
volume_stop_frame = Frame(self)
volume_stop_frame.pack(side="top")
stop_volume = Scale(master=volume_stop_frame, from_=0, to=2000,
                    resolution=50.0, length=150, orient=HORIZONTAL)
stop_volume.set(self.master.sliders.get("stop"))
stop_volume.pack(side="right", padx=10)
volume_stop_label = Label(master=volume_stop_frame,
                          text="Deactivation Volume: ")
volume_stop_label.pack(side="right")
```

Listing 4.4: Erstellung eines Sliders mit Beschriftung mit Hilfe von Tkinter

Zuerst wird ein Frame erstellt, an dem später der Slider und dessen Beschriftung ausgerichtet werden. Über die „pack()“ Funktion wird der Frame selber an das obere Ende des Fensters ausgerichtet. Die „Scale()“ Funktion erstellt einen Slider. Mit dem Parameter „Master“ wird der zuvor erstellte Frame als Ausrichtungspunkt festgelegt. Die Parameter „from“ und „to“ geben an welchen Zahlenbereich der Slider agiert und

über den „resolution“ Parameter kann die Schrittweite festgelegt werden. Mit dem Parameter „orient“ wird der Slider horizontal ausgerichtet.
Als nächstes wird eine Variable des Hauptfensters genutzt, um den Wert des Sliders festzulegen. Dieser entspricht der aktuellen Einstellung des SA. Mit Bestätigen des Fensters wird der neue Wert des Sliders in diese Variable geschrieben. Der Slider wird über „pack()“ mit einem Padding von 10 Pixeln an die rechte Seite des Frames ausgerichtet. Zuletzt wird ein Label für die Beschriftung des Sliders erstellt und ebenfalls auf die rechte Seite des Frames – und damit links neben den Slider – ausgerichtet.

6.2 Implementierung des Sprachassistenten

Der SA wird in vier Teile aufgeteilt, die Sprachaufnahme und Transkription, die Text Klassifizierung, die Entität Extraktion und die Umsetzung der Intents.

6.2.1 Sprachaufnahme und Transkription

Für die Umsetzung der Sprachaufnahme und die Transkription wird eine neue Klasse erstellt. Die Implementierung wird über zwei Hauptschleifen realisiert. Die innere Schleife ist für die Sprachaufnahme zuständig, die äußere Schleife behandelt die Transkription und steuert die innere Schleife.

Wird der SA das erste Mal oder nach einer Änderung des Hardware Beschleunigers gestartet, wird das FW Modell initialisiert (Siehe Listing 5.1).
Zu Beginn der Initialisierung wird eine Pop-Up Nachricht erzeugt, die in Abschnitt 6.2.5 genauer erläutert werden und in Listing 5.1 als Kommentar dargestellt sind.
Über die Funktion „WhisperModel“ wird ein FW Modell erzeugt. Im ersten Parameter wird das Sprachmodell gewählt. Die Parameter „device“ und „compute_type“ bestimmen mit welchem Gerät – also CPU oder GPU – und mit welcher Rechenart das Modell arbeiten soll. Die Variablen dieser Parameter besitzen einen Standardwert, der über das Optionsmenü des Hauptfensters (Siehe Abschnitt 6.3.1) angepasst werden kann. Bei dem allerersten Start des Modells wird das ausgewählte Sprachmodell heruntergeladen. Bei dem „medium“ Modell sind das ein paar hundert Megabyte, was je nach Internetanbindung Zeit beansprucht.
Tritt bei der Initialisierung ein Fehler auf – etwa wenn eine Hardware Beschleuniger Option ausgewählt wurde, die nicht mit dem Gerät kompatibel ist – wird der SA gestoppt und es erscheint eine Pop-Up Fehlermeldung im Hauptfenster; in dm Listing ebenfalls als Kommentar markiert. Gestoppt wird der SA, indem ein Boolean Wert gesetzt wird, den die zuvor genannten Schleifen als Abbruchbedingung haben.

```
def configure_model(self):
    #self.owner.create_processing_info("Initialize Voice Assistant...")
    try:
        self.model = WhisperModel("medium", device=self.tts_device,
                                                    compute_type=self.tts_compute_type)
    except Exception:
        self.owner.stop_micro_on_info()
        self.stop()
        self.owner.va_on = False
        #self.owner.create_error_info("Error. Hardware Accelerator is not
                                                                supported by hardware", timer=3000)
```

Listing 5.1: Initialisierung des Faster Whisper Sprachmodell für die Transkription

Zum Starten des SA wird eine Funktion erstellt, die – ebenso wie das Stoppen – über den „Voice“ Button des Hauptfensters aufgerufen werden kann. Wie Listing 5.2 zeigt, wird beim Start das FW Modell initialisiert, sofern nicht bereits vorhanden. Danach wird die Mikrofon Indexnummer von dem Optionsmenü des Hauptfensters übernommen und ein PyAudio Objekt erstellt. Wenn es bei der Initialisierung des FW Modells keine Fehler gab, wird die „main()“ Funktion der Klasse aufgerufen, welche unter anderem die beiden Hauptschleifen enthält.

```
def start(self, microphone_id):
    self.stop_all = False
    if not self.model:
        self.configure_model()
    if microphone_id:
        self.input_device_index = microphone_id
    self.py_audio = pyaudio.PyAudio()
    if not self.stop_all:
        self.main()
```

Listing 5.2: Start Funktion des Sprachassistenten

Die „main()“ Funktion öffnet mit Hilfe des Pyaudio Objektes einen Stream zum ausgewählten Eingabegerät (Siehe Listing 5.3). Die Namen der meisten Parameter sind selbsterklärend. „format“ legt das Sample Format des Streams fest, hierfür stellt Pyaudio Formate zur Verfügung. Über „input“ wird festgelegt, ob der Stream ein Eingangs- oder Ausgangsstream ist und „input_device_index“ wählt das Eingabegerät anhand des Indexes. Dieser ist standardmäßig das erste gefundene Gerät und kann über das Optionsmenü im Hauptfenster angepasst werden.

```
stream = self.py_audio.open(format=pyaudio.paInt16, channels=1, rate=16000,
                        input_device_index=int(self.input_device_index)
                        , input=True, frames_per_buffer=1024)
```

Listing 5.3: Erstellung eines Audio Streams mit Pyaudio

Listing 5.4 zeigt den Aufbau der zuvor genannten äußeren Hauptschleife. Es wird überprüft, ob der geöffnete Stream eine bestimmte Lautstärke übersteigt. Der Schwellwert wird aus dem Hauptfenster ausgelesen und wird über dessen Optionsmenü angepasst. Wird der Schwellwert überschritten, wird die Funktion „recording()“ ausgeführt (Siehe Listing 5.4), die die zweite Hauptschleife enthält und für die Aufnahme zuständig ist.
Sobald die Aufnahme beendet ist, wird die Aufnahme mit Hilfe der „transcribe()“ Funktion des FW Modells in Text übersetzt. Der erste Parameter gibt den Pfad zur temporären Audio Datei an, die in der Aufnahme-Schleife erstellt wurde. „beam_size“ bestimmt wie viele Pfade bei der Strahlensuche in der Transkription verwendet werden. Indem der Parameter „condition_on_previous_text“ auf False gesetzt wird, wird vorheriger Text bei der Übersetzung nicht beachtet. Dies reduziert das Halluzinieren des Modells.

```
while not self.stop_all:
    if check_volume(stream.read(1024)) > self.owner.sliders.get("start"):
        self.recording(stream)
        print("processing...")
        #self.owner.create_processing_info("Processing...")
        segments = self.model.transcribe(self.temp_audio_file_path,
                                         beam_size=5, language="de",
                                                              condition_on_previous_text=False)
        text = ""
        for segment in segments:
            text += segment.text + " "
        print("processing done")
        #self.owner.stop_processing_info()
        if self.owner:
            self.owner.process_va_data(text)
        os.remove(self.temp_audio_file_path)
```

Listing 5.4: Schleife zum Starten der Aufnahme und Transkribieren der Beendeten Aufnahme

Da das Fast Whisper Modell die Transkription in Segmenten vornimmt, müssen diese zusammengesetzt werden.
Der Text wird an eine Funktion im Hauptfenster übergeben, durch die der Text weiter verarbeitet wird. Zuletzt wird die temporäre Audio Datei wieder gelöscht.
Der Benutzer wird per Pop-Up Nachricht über den Status der Verarbeitung informiert. Diese sind in dem Listing wieder als Kommentare markiert.

Wie im vorherigen Absatz erwähnt, wird die „recording()“ Funktion (Siehe Listing 5.5) ausgeführt, wenn die Lautstärke des Streams den Gesetzten Schwellwert überschreitet. Zu Beginn wird ein Timer gestartet, dieser wird später für die Beendigung der Aufnahme verwendet. Als nächstes wird die innere Hauptschleife für die Aufnahme gestartet. Die Daten des Streams werden mit jedem Durchlauf der

Schleife in eine Variable gespeichert.
Danach wird gemessen, ob die Lautstärke unter dem Schwellwert gefallen ist, der im Optionsmenü des Hauptfensters festgelegt werden kann. Liegt die Lautstärke darüber, wird der zuvor gestartete Timer zurückgesetzt. Liegt die Lautstärke darunter, wird geprüft, wie lang das bereits der Fall ist. Liegt die Lautstärke länger unter dem Schwellwert als die festgelegte Höchstdauer, wird die Aufnahme beendet. Die Höchstdauer kann ebenfalls über das Optionsmenü des Hauptfensters eingestellt werden. Nachdem die Aufnahme beendet wurde, wird sie temporär in eine .wav Datei gespeichert.

```
def recording(self, stream):
    frames = []
    self.voice_time_start = time.time()
    print("recording:...\n")
    #self.owner.create_recording_info(text="Recording...")
    while not self.stop_all:
        data = stream.read(1024)
        frames.append(data)
        if check_volume(data) < self.owner.sliders.get("stop"):
            self.voice_time_end = time.time()
            deltatime = self.voice_time_end - self.voice_time_start
            print(deltatime)
            if deltatime > self.owner.sliders.get("timer"):
                self.start_time = time.time()
                break
        else:
            self.voice_time_start = time.time()
    self.bitstream_to_wave(frames, self.temp_audio_file_path)
    print("recording ended")
    #self.owner.stop_recording_info()
```

Listing 5.5: Aufnahme Funktion des Sprachassistenten.

Die an dem „Voice“ Button gebundene Funktion im Hauptfenster erstellt zum Starten – sofern nicht bereits vorhanden - eine Instanz der SA Klasse und startet dessen „start()“ Funktion in einem neuen Thread.

6.2.2 Text Klassifizierung

Für die Text Klassifizierung wird eine neue Datei mit zwei Funktionen erstellt, eine zum Vorverarbeiten des transkribierten Textes und eine für die anschließende Klassifizierung. Nach der Sprachaufnahme wird der übersetzte Text an diese beiden Funktionen übergeben.

Zuerst wird geprüft, ob überhaupt Text übersetzt wurde; ist das nicht der Fall, gibt die Funktion ein Listenobjekt mit dem Wert 0 zurück, welches zu einer Wertung als „Anweisung nicht erkannt“ führt. Hat der Text einen Inhalt, wird dieser vorverarbeitet.

Über Pythons „lower()“ Funktion wird der Text in Kleinbuchstaben umgewandelt. Über Pythons „replace()“ Funktion werden ausgewählte Sonder- und Satzzeichen aus dem Text entfernt.
Als nächstes wird der Text an Leerstellen in ein Array aus Wort Token aufgeteilt. Da die Transkription den Text in einem Format ausgibt, durch das letzte Sonder-/Satzzeichen des Textes nicht durch den regulären Ausdruck entfernt werden kann, wird dieser manuell über Pythons „replace()“ Funktion entfernt. Abschließend wird der Text mit Hilfe der „stem()“ Funktion des Snowball Stemmers der NLTK Bibliothek gestemmt (Siehe Listing 6.1).
Zum Schluss wird aus den Wörtern des vorverarbeiteten Textes und des Originaltextes eine Liste aus Wortpaaren erstellt und im Hauptfenster gespeichert. Diese Liste wird später für die Entität Extraktion verwendet.

```
def preprocess_text(owner, text):
    if len(text) == 0:
        return ["0"]
    #lowercasing
    text = text.lower()
    original_text_list = text.split()
    original_text_list[-1] = original_text_list[-1].strip(
        string.punctuation)
    print(f'Original text: {text}')
    #remove special chars stopwords and tokenize
    forbidden_chars = [".", ",", ";", ":", "!", "?", "_",
                       "-", "(", ")", "\"", "\'"]
    for word in forbidden_chars:
        text = text.replace(word, "")
    # print(f'Text ohne Sonderzeichen: {text}')
    # text = text.strip(string.punctuation)
    text_list = text.split()
    #stemming
    text_list = [snowball.stem(word) for word in text_list]
    print(f'Verarbeiteter text: {text_list}')

    original_text_tuple_list = list(zip(original_text_list, text_list))
    print(f'Tuple List: {original_text_tuple_list}')
    owner.original_text_tuple_list = original_text_tuple_list

    return text_list
```

Listing 6.1: Vorverarbeitung des transkribierten Sprachbefehls

Anschließend werden für jedes Intent eine Reihe von Sprachbefehlen aufgenommen, transkribiert und vorverarbeitet. Aus den dann vorverarbeiteten Texten werden Schlüsselwörter für jedes Intent entnommen und jeweils einem Listen Objekt gespeichert. Auf die gleiche Weise werden Schlüsselwörter für die Zeilenangabe

entnommen und in ein Listen Objekt gespeichert.
Die Liste an Schlüsselwörtern für den Befehl zum Erstellen eines Threads sieht zum Beispiel wie folgt aus: thread_keywords = ['thread', 'wett', 'thwett', 'sweat']
Die Wörter wett, thwett und sweat sind mit in der Liste enthalten, da „Thread“ – bei undeutlicherer Aussprache – vermehrt in diese Wörter übersetzt wurde. Wie in Abschnitt 5.3.4 begründet, wird für diese Wörter keine Korrektur vorgenommen; stattdessen werden sie als Schlüsselwörter übernommen.

In der zweiten Funktion, die für die Text Klassifizierung zuständig ist, wird eine If-Else Abfrage erstellt, in der das Array des Textes nach jedem Schlüsselwort des jeweiligen Intents durchsucht wird. Listing 6.2 zeigt beispielhaft den ersten Teil der Abfrage. Die If-Abfrage durchläuft jedes Wort des Text Arrays „text_list“ und vergleicht es mit jedem Wort aus der Schlüsselwort Liste „jump_to_keywords“. Wurde ein Schlüsselwort gefunden, werden eine Funktion zur Ermittlung der benötigten Zeilen und die Funktion zur Umsetzung des Intents ausgeführt. Beide Funktionen werden in den nächsten Abschnitten – 6.2.3 und 6.2.4 – erstellt. Wird kein passendes Schlüsselwort gefunden, wird eine Meldung ausgegeben, dass keine Anweisung erkannt wurde.

```
if any(word in text_list for word in jump_to_keywords):
    row = f.find_one_row(text_list)
    e.jump_to_row(editor, owner, row=row)
```

Listing 6.2: Suche nach einer Reihe von Schlüsselwörtern in einem Text Array

6.2.3 Entitäts Extraktion

Bei der Entitäts Extraktion werden in jeweils einer Funktion die erste und alle Zeilenangaben einer Sprachanweisung ermittelt. Listing 7.1 zeigt die Funktion zur Ermittlung der ersten Zeilenangabe.
Als erstes wird das Text Array zu einem einzigen String zusammengefügt. Danach wird jedes gefundene Schlüsselwort für die Zeile im Text durch „ zeile „ ersetzt. Zu beachten ist, dass das Wort von zwei Leerzeichen umschlossen ist. Wenn der Text dieses eine, neue Schlüsselwort enthält, wird der Bereich vor dem Schlüsselwort abgetrennt. Der Teil nach dem Schlüsselwort wird nach einer Nummer durchsucht. Die erste Nummer die gefunden wird, wird als gefundene Zeile zurückgegeben. Wird kein Schlüsselwort oder keine Nummer zurückgegeben, wird None zurückgegeben.

```
def find_one_row(text_list):
    text = " ".join(text_list)
    for word in zeile_keywords:
        text = text.replace(word, " zeile ")
    if " zeile " in text:
        text = text.split(" zeile ")[1]
        text = text.split(" ")
        for word in text:
            if word.isdigit():
                print(f'Zeile: {word}')
                return word
        else:
            print("No Number")
            return None
    else:
        print("No Zeile")
        return None
```

Listing 7.1: Funktion für die Durchsuchung eines Textes auf Zeilenangaben

Auf diese Weise wird die erste Nummer nach dem Schlüsselwort „zeile“ als Zeilenangabe erkannt. In der Variante für mehrere Zeilenangaben, werden alle Nummern nach dem Schlüsselwort zurückgegeben.

Für die Anweisung der If-Abfrage und While Schleife werden zusätzlich Informationen über die Vergleichswerte und -operatoren gesucht. Dies erfolgt in zwei Schritten, der Bestimmung, ob eine Negation der Vergleichsanweisung vorliegt und der Extraktion des Vergleichsoperators und die dazugehörigen Werte, bzw. Variablen. Für ersteres wird der Text nach den Schlüsselwörtern „nicht“ und „not“ durchsucht.
Letzteres wird in Listing 7.3 dargestellt. Hierfür wird die Liste aus Wortpaaren aus Originaltext und vorverarbeiteten Text verwendet. Da die Vergleichsvariablen Wörter sein können, ist es wichtig sie in ihrer Originalform zu ermitteln. Mit der Wortpaar Liste kann stets auf beide Versionen eines Wortes zugegriffen werden. Auf diese Weise können weiterhin die Vorteile der Vorverarbeitung, wie etwa das Stemmen, genutzt werden.
Die Wortliste wird nach Schlüsselwörtern durchsucht und dann wird das vorherige und nachfolgende Wort je als Vergleichswert/-variable angesehen. Dafür werden zuerst bestimmte Stoppwörter entfernt. Eine Eingabe nach dem Muster „…a nicht größer als 5“ würde nicht „a“ und „5“ als Vergleichswerte erkennen, sondern „nicht“ und „als“. Durch die Entfernung dieser Wörter wird sichergestellt, dass das nicht passiert.
Als nächstes wird jedes vorverarbeitete Wort der Liste mit jedem Schlüsselwort für Vergleichsoperatoren verglichen. Die Schlüsselwörter umfassen „kleiner“, „größer“ und „gleich“, ebenfalls vorverarbeitet, also „klein“, „gross“ und „gleich“. Wurde ein

Schlüsselwort gefunden und handelt es sich dabei um „gross“ oder „klein“ wird das darauffolgende Wort ebenfalls untersucht. Auf diese Weise wird überprüft, ob es sich z.B. um ein „größer“ oder „größer gleich“ handelt. Trifft dies zu, wird der Operator angepasst und das Wort übersprungen.
Danach wird das gefundene Wort (bzw. Wörter) als Operator gespeichert, das vorangehende Wort als erster Vergleichswert und das nachfolgende als zweiter. Dieser Schritt wird ebenfalls durchgeführt, wenn es sich nicht um „größer/kleiner gleich“ oder ein anderes Schlüsselwort handelt.

```
def extract_if_operators(text_tuple_list):
    variable_a = None
    variable_b = None
    operator = None
    for i in range(0, len(text_tuple_list) - 1):
        for keyword in if_operator_keywords:
            if keyword == text_tuple_list[i][1]:
                if keyword == "gross" or keyword == "klein":
                    if text_tuple_list[i + 1][1] == "gleich":
                        variable_a = text_tuple_list[i - 1][0]
                        operator = text_tuple_list[i][1] + " gleich"
                        variable_b = text_tuple_list[i + 2][0]
                        return [variable_a, variable_b, operator]

                variable_a = text_tuple_list[i - 1][0]
                operator = text_tuple_list[i][1]
                variable_b = text_tuple_list[i + 1][0]
                return [variable_a, variable_b, operator]
    return [variable_a, variable_b, operator]
```

Listing 7.2: Funktion zur Extraktions von Vergleichsoperatoren und –Variablen.

6.2.4 Intents

Für jedes Intent wird eine Funktion erstellt, die die entsprechende Anweisung im Code Editor des Hauptfensters ausführt. Weiterhin werden Funktionen erstellt, um die aktuelle Position des Cursors im Code Editor zu ermitteln, den Cursor an eine bestimmte Stelle des erstellten Codes zu setzen, sicherstellt, dass benötigte Module importiert werden und um eine korrekte Einrückung von eingefügtem Code sicherstellt. Die korrekte Einrückung richtet sich nach den Einrückungen der betreffenden Zeile, den Einrückungen der Vorherigen Zeile, ob die vorherige Zeile eine Programmstruktur enthält, die eine weitere Einrückung erfordert und ob es sich bei dem einzufügenden Code um eine solche Programmstruktur handelt.

Listing 8.1 zeigt beispielhaft die Funktion für die Erstellung einer While Schleife. Die Parameter „editor“ und „owner“ geben das Text Widget des Code Editors und das Hauptfenster an. Über „row“ kann optional die Zeile angegeben werden, in der der Code platziert werden soll. Zuerst wird in der Funktion „adjust_row“ die Eingabe der

Zeile als Kommastellenzahl und String formatiert, sodass sie für die „insert()“ Funktion des Text Widgets geeignet ist. Die Angabe „4“ wäre zum Beispiel nicht gültig und wird daher in „4.0“ geändert.
Danach wird die Syntax erstellt und in die korrekte Einrückung hinzugefügt (Siehe Listing 8.2). Zuletzt wird die Syntax in das Code Editor Text Widget geschrieben und der Cursor an die entsprechende Position gesetzt, um ein schnelles Eingeben der Bedingung zu ermöglichen.

```
def insert_while(editor, owner, row=None):
    val = adjust_row(editor, row)
    command = "while  :"
    command = formate_function(editor, row, command)
    editor.insert(val + " lineend", command)
    set_cursor_position(editor, command=command, row=val, offset=2)
```

Listing 8.1: Funktion für die Erstellung einer While Schleife im Code Editor

Der Funktion zur Formatierung der Einrückung werden das Code Editor Text Widget, die betroffene Zeile und die betroffene Syntax als Parameter übergeben. Es wird vorangehende Zeile ermittelt und dessen Text ausgelesen. Danach wird ermittelt wie viele Einrückungen die vorangehende Zeile enthält und ob sie eine Funktion enthält. Entsprechend dem Ergebnis werden der Syntax passende Einrückungen und eine neue Zeile mit ebenfalls passenden Einrückungen hinzugefügt.

```
def formate_function(editor, row, function_text):
    val = adjust_row(editor, row=row)

    prev_row = get_previous_row(val)
    prev_row_text = editor.get(prev_row + " linestart", prev_row +
                                                              " lineend")
    text = function_text + "\n\t"
    tabs_prev = check_tabs(editor, prev_row)
    for i in range(0, tabs_prev):
        text = "\t" + text + "\t"
    if check_for_function(prev_row_text):
        text = "\t" + text + "\t"
    text = "\n" + text
    return text
```

Listing 8.2: Funktion zur Ermittlung und Erstellung der passenden Einrückung

6.2.5 Pop-Up Benachrichtigungen

Im Hauptfenster werden Label Widgets erstellt, die als Pop-Up Benachrichtigungen dienen. Je nach Benachrichtigung tauchen diese Pop-Ups entweder für eine kurze Zeit auf oder bis der entsprechende Schritt durchgeführt wurde. Zum Beispiel erscheint eine Benachrichtigung, während das Sprachmodell des SA initialisiert wird;

dieses verschwindet erst wieder, wenn der Vorgang beendet wurde. Ein Pop-Up, das den Benutzer über eine nicht erkannte Anweisung informiert, verschwindet nach einer kurzen Zeit automatisch wieder. Listing 9 zeigt das Label für eine Fehlernachricht. Fehlernachrichten können tatsächliche Fehler sein, etwa wenn ein nicht unterstützter Hardware Beschleuniger ausgewählt wurde, aber auch Sprachanweisungen, die nicht zugeordnet werden konnten.
Über die Parameter können der angezeigte Text, die Hintergrundfarbe der Nachricht und die Lebzeit der Nachricht bestimmt werden. Es wird ein leeres Bildobjekt erzeugt, damit die Größe des Labels nicht mehr vom enthaltenen Text abhängt, sondern frei gewählt werden kann. Text und Farbe werden bei der Erstellung des Labels aus den Parametern übernommen und es wird die Größe in Relation zum Hauptfenster gesetzt. Das Label wird statt mit „pack()“, mit der „place()“ Funktion positioniert. Dies erlaubt es das Label vor, bzw. „über“ den anderen Elementen zu platzieren. Über die „after()“ Funktion wird das Label nach der angegebenen Zeit automatisch zerstört.

```
def create_error_info(self, text="", color="red", timer=3000):
    img = PhotoImage()
    self.error = Label(master=self.editor, text=text,
                       image=img, compound=CENTER,
                       font=('', self.info_text_size),
                       bg=color, width=self.winfo_width(),
                       height=self.info_height)
    self.error.place(relx=.5, y=int(self.editor.winfo_height()
                                    - self.info_height + 3)
                     , anchor=CENTER)
    self.after(timer, self.error.destroy)
```

Listing 9: Erstellung eines Labels für Pop-Up Nachrichten

6.3 Anwendung des Prototypen

Um die Anwendung des Code Editors und SA zu demonstrieren, werden zwei Beispiel Applikation entwickelt. Beide Anwendungen werden einmal mit und einmal ohne SA erstellt. Um einen Vorteil durch das wiederholte Schreiben des Codes zu verringern, wird die erste Applikation zuerst ohne dem SA erstellt und die zweite Applikation zuerst mit SA.
Listing 10 zeigt die Anweisungen der beiden Applikationen.

```
Applikation 1:
Erstelle die Variable alice=0 und bob=5.
Erstelle eine Endlosschleife.
Prüfe innerhalb der Endlosschleife of alice größer bob ist.
Wenn nicht, erhöhe den Wert von alice um eins.
Wenn ja, unterbreche die Endlosschleife und printe beide Variablen.
Erstelle und Starte einen Timer.
Erstelle eine For-Schleife von eins bis 100.
Printe in jedem Durchlauf die aktuelle Schleifenzahl.
Stoppe nach der Schleife den Timer und printe die Zeit.

Applikation 2:
Erstelle eine neue Funktion „suche“ mit dem Parameter „finde“.
Erstelle in der Funktion einen Match Case für den „finde“ Parameter.
Erstelle fünf Cases, je mit der Zahl von eins bis fünf.
Printe in jedem Case die entsprechende Zahl.
Erstelle außerhalb der Funktion einen Input.
Erstelle eine Try/Except Struktur.
Erstelle innerhalb der Try/Except Struktur einen neuen Thread.
Starte im Thread die „suche“ funktion und übergebe den Input als Parameter
```

Listing 10: Anweisungen für zwei Testsapplikationen

6.4 Evaluation

Bei der Erstellung der beiden Testapplikationen war die Arbeitsweise mit Sprachassistenz in beiden Fällen schneller. Auch wenn diese zwei Tests nicht repräsentativ sind, wird das Ergebnis durch den Eindruck bei der Codeerstellung bestärkt. Das Verwenden von Spracheingaben erfordert eine gewisse Umstellung, zeigt jedoch bei mehrzeiligen Programmstrukturen eine höhere Geschwindigkeit. Eine Schleife oder Abfrage kann per Sprachanweisung in eine Zeile „bestellt“ werden, währen selbst in einer anderen Zeile geschrieben wird. Auch das Erstellen von Strukturen ohne paralleles Schreiben war oft schneller, bzw. hat sich schneller angefühlt. Sehr simple Anweisungen wie das Erstellen eines Print Befehls, das Kopiere, Einfügen oder Löschen und das Navigieren zu bestimmten Zeilen, hat sich nicht als effizienter herausgestellt. Die Eingabe per Maus und Tastatur ist deutlich einfacher. Bei Anweisungen die den Import eines Moduls erfordern, wie ein Thread oder Timer, zeigte sich ebenfalls ein Vorteil. In den meisten Fällen werden solche Importe am Anfang einer Datei vorgenommen, dadurch muss der Benutzer einmal zum Anfang und zurück zu der Stelle navigieren, an der gerade gearbeitet wird. Über die Sprachbefehle wird der Import automatisch erzeugt und es ist kein Navigieren notwendig. Dieser Vorteil relativiert sich allerdings in Hinblick auf moderne Entwicklungsumgebungen, die fehlende Importe oft über ein Kontextmenü hinzufügen können.

Die Entwicklungsumgebung und der SA unterliegen einigen Begrenzungen. Die Entwicklungsumgebung kann Pythons Input() Funktion nicht korrekt ausgeben. Der Text, der die Eingabe einleiten soll, wird erst nach Absenden der Eingabe angezeigt. Der Algorithmus für die korrekte Einrückung einzufügenden Codes berücksichtigt keine mehrzeiligen Kommentare.
Die erweiterte Entität Extraktion der If-Anweisung, bzw. While Schleife unterstützt nicht die Operatoren „in“, „and“ und „or“.
Um die Hardware Beschleuniger nutzen zu können, ist eine Nvidia Grafikkarte notwendig. Weiterhin müssen das Programm CUDA 12 und die Bibliotheken cuDNN for CUDA12 und cuBLAS for CUDA 12 installiert sein. Ohne Hardware Beschleunigung dauert die Übersetzung länger, sodass der SA wohl keinen Vorteil mehr bietet.

7. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden die wichtigsten Aspekte der Entwicklung eines SA zusammengefasst, analysiert und Herausforderungen herausgearbeitet, mit dem Ziel einen SA zu schaffen, der Entwicklern bei Programmieraufgaben hilft. Im Besonderen für die Zuordnung von Anweisungen aus Sprachbefehlen wurden verschiedene Methoden verglichen und ausgewertet. Aufbauend auf dieser Auswertung wurde die für die Entwicklung eines SA mit genannter Funktion geeignetste Methode dargelegt. Für die Umsetzung wurde eine minimalistische grafische Entwicklungsumgebung vorgesehen, um möglichst viel Freiheit bei der Entwicklung des SA zu haben und um ihn praktisch anwenden zu können. Mit Hilfe der gewählten Methoden konnte ein Konzept, das dem Nutzer erlaubt Python Code zu programmieren und auszuführen und dabei bestimmte Arbeitsschritte per Sprachbefehl ausführen zu lassen. Dieses Konzept wurde anschließend in Form eines eigenständigen Systems umgesetzt.

Bei dem umgesetzten Programm handelt es sich um einen Prototypen, der gegebenenfalls um weitere Funktionalitäten ergänzt werden muss. So ermöglicht der SA nur exemplarisch für die If-Abfrage und While Schleife eine Einbindung eigener Parameter. Andere Funktionen sind auf die Angabe der Zeile begrenzt. Die Grundstruktur wird erstellt, jedoch muss die Bedingung der Schleife per Hand eingetragen werden. Eine Erweiterung des SA, die eine Einbindung eigener Parameter für alle Funktionen ermöglicht, würde den Nutzen weiter steigern.

Literaturverzeichnis

[1] Haslbeck, Andreas & Pecot, Katrina & Popova-Dlugosch, Severina & Eichinger, Armin & Bengler, Klaus. (2011). Menschliche Zuverlässigkeit bei der alphanumerischen Eingabe mittels unterschiedlicher Eingabemedien.
URL: https://www.researchgate.net/publication/258226621
(Besucht am: 14.01.2024)

[2] Sandra Keller Chandra, Ulrike Hoehne-Hückstädt, Rolf Ellegast, BGIA – Institut für Arbeitsschutz der Deutschen Gesetzlichen, Unfallversicherung, Sankt Augustin, Peter Schäfer, Verwaltungs-Berufsgenossenschaft – VBG, Ludwigsburg.
Ergonomische Anforderungen an Eingabemittel für Geräte der Informationstechnik.
URL: https://edocs.tib.eu/files/e01fn09/56018221X.pdf
(Besucht am: 16.01.2024)

[3] Serpil Tas, Christian Hildebrandt, René Arnold. Sprachassistenten in Deutschland.
URL: http://hdl.handle.net/10419/227052
(Besucht am: 14.01.2024)

[4] Oliver Nadig. "Hallo Computer" - Möglichkeiten und Grenzen der Spracheingabe als digitaler Arbeitstechnik.
URL: https://www.dvbs-online.de/index.php/publikationen-3/horus/horus-marburger-beitr%C3%A4ge-3-2020#5
(Besucht am: 17.01.2024)

[5] Matthias Wille, Christiane Adomeit, Marco Lehmann. Spracheingabe als Alternative zur Tastatur bei der Büroarbeit.
URL: https://doi.org/10.37307/j.2199-7349.2013.01.07
(Besucht am: 16.01.2024)

[6] M. Gupta, R. Kumar and H. Sardalia, "Voice Assistant Technology: The Case of Jarvis AI," 2023 4th International Conference for Emerging Technology (INCET), Belgaum, India, 2023, pp. 1-5,
DOI: 10.1109/INCET57972.2023.10170362.
(Besucht am: 14.04.2024)

[7] S. Dange, R. Kasture, A. Kadhao, A. Thorat and S. Mhamane, "Build: Web Services based Source Code Editor Integrate with Community Question Answer," 2020 Fourth International Conference on Inventive Systems and Control

(ICISC), Coimbatore, India, 2020, pp. 245-249,
DOI: 10.1109/ICISC47916.2020.9171162.
(Besucht am: 14.04.2024)

[8] Jochen Hartmann, Juliana Huppertz, Christina Schamp, Mark Heitmann, Comparing automated text classification methods, International Journal of Research in Marketing, Volume 36, Issue 1, 2019, Pages 20-38, ISSN 0167-8116,
URL: https://doi.org/10.1016/j.ijresmar.2018.09.009.
(Besucht am: 14.04.2024)

[9] Webb, Geoffrey. (2016). Naïve Bayes.
DOI: 10.1007/978-1-4899-7502-7_581-1.
(Besucht am: 15.04.2024)

[10] Jacob Eisenstein. Introduction to Natural Language Processing. Cambridge, MA: The MIT Press, 2019. S.22- 23.

[11] I. RishT.J. Watson Research Center. An empirical study of the naive Bayes classifier. URL:
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2825733f97124013e8841b3f8a0f5bd4ee4af88a
(Besucht am: 15.04.2024)

[12] Olson, D.L., Araz, Ö.M. (2023). Naïve Bayes Models in Healthcare. In: Data Mining and Analytics in Healthcare Management. International Series in Operations Research & Management Science, vol 341. Springer, Cham.
URL: https://doi.org/10.1007/978-3-031-28113-6_12
(Besucht am: 16.04.2024)

[13] Eyheramendy, S., Lewis, D.D. & Madigan, D.. (2003). On the Naive Bayes Model for Text Categorization. Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics in Proceedings of Machine Learning Research
URL: http://proceedings.mlr.press/r4/eyheramendy03a.html
(Besucht am: 16.04.2024)

[14] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).
URL https://doi.org/10.1023/A:1010933404324
(Besucht am: 20.04.2024)

[15] Steven J. Rigatti; Random Forest. J Insur Med 1 January 2017; 47 (1): 31–39.
DOI: https://doi.org/10.17849/insm-47-01-31-39.1
(Besucht am: 20.04.2024)

[16] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).
URL: https://doi.org/10.1023/A:1010933404324
(Besucht am: 20.04.2024)

[17] Harsh H. Patel, Purvi Prajapati, (2018). Study and Analysis of Decision Tree Based Classification Algorithms. International Journal of Computer Sciences and Engineering, 6(10), 74-78.
URL: http://dx.doi.org/10.26438/ijcse/v6i10.7478
(Besucht am: 23.04.2024)

[18] Schlenger, J. (2024). Random Forest. In: Memmert, D. (eds) Computer Science in Sport. Springer, Berlin, Heidelberg.
URL: https://doi.org/10.1007/978-3-662-68313-2_24
(Besucht am: 23.04.2024)

[19] Akuma, S., Lubem, T. & Adom, I.T. Comparing Bag of Words and TF-IDF with different models for hate speech detection from live tweets. Int. j. inf. tecnol. 14, 3629–3635 (2022).
URL: https://doi.org/10.1007/s41870-022-01096-4
(Besucht am: 23.04.2024)

[20] Steven J. Rigatti; Random Forest. J Insur Med 1 January 2017; 47 (1): 31–39.
DOI: https://doi.org/10.17849/insm-47-01-31-39.1
(Besucht am: 23.04.2024)

[21] Jaime Lynn Speiser, Michael E. Miller, Janet Tooze, Edward Ip,
A comparison of random forest variable selection methods for classification prediction modeling, Expert Systems with Applications, Volume 134, 2019, Pages 93-101, ISSN 0957-4174
URL: https://doi.org/10.1016/j.eswa.2019.05.028.
(Besucht am: 27.04.2024)

[22] I. Ahmad, M. Basheri, M. J. Iqbal and A. Rahim, "Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection," in IEEE Access, vol. 6, pp. 33789-33795, 2018.
DOI: 10.1109/ACCESS.2018.2841987.
(Besucht am: 27.04.2024)

[23] Suzuki, Kenji. (2011). Artificial Neural Networks - Methodological Advances and Biomedical Applications.
URL: https://www.researchgate.net/publication/319316102_Artificial_Neural_Networks_-_Methodological_Advances_and_Biomedical_Applications
(Besucht am: 27.04.2024)

[24] Siddharth Sharma, Simone Sharma, Anidhya Athaiya. International Journal of Engineering Applied Sciences and Technology, 2020. Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316
URL: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf
(Besucht am: 28.04.2024)

[25] Muhammad Zulqarnain, Rozaida Ghazali, Yana Mazwin Mohmad Hassim, Muhammad Rehan. A comparative review on deep learning models for text classification.
URL: http://dx.doi.org/10.11591/ijeecs.v19.i1.pp325-335
(Besucht am: 28.04.2024)

[26] Bodapati, S., Bandarupally, H., Shaw, R.N., Ghosh, A. (2021). Comparison and Analysis of RNN-LSTMs and CNNs for Social Reviews Classification. In: Bansal, J.C., Fung, L.C.C., Simic, M., Ghosh, A. (eds) Advances in Applications of Data-Driven Computing. Advances in Intelligent Systems and Computing, vol 1319. Springer, Singapore.
URL: https://doi.org/10.1007/978-981-33-6919-1_4
(Besucht am: 30.04.2024)

[27] Alon Jacovi, Oren Sar Shalom, Yoav Goldberg. Understanding Convolutional Neural Networks for Text Classification
URL: https://arxiv.org/abs/1809.08037
(Besucht am: 30.04.2024)

[28] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6,
DOI: 10.1109/ICEngTechnol.2017.8308186.
(Besucht am: 30.04.2024)

[29] Keiron O'Shea, Ryan Nash. An Introduction to Convolutional Neural Networks.

URL: https://arxiv.org/abs/1511.08458
(Besucht am: 30.04.2024)

[30] Ye Zhang, Byron Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.
URL: https://arxiv.org/abs/1510.03820
(Besucht am: 02.05.2024)

[31] Ahmed A. Elngar, Mohamed Arafa, Amar Fathy, Basma Moustafa, Omar Mahmoud, Mohamed Shaban, Nehal Fawzy. (2021). Image Classification Based On CNN: A Survey. Journal of Journal of Cybersecurity and Information Management, 6 (1), PP. 18-50
URL: http://dx.doi.org/10.54216/JCIM.060102
(Besucht am: 02.05.2024)

[32] Landesvatter, C., Behnert, J., & Bauer, P. C. (2023, October 10). Comparing Speech-to-Text Algorithms for Transcribing Voice Data from Surveys.
URL: https://doi.org/10.31235/osf.io/vk6wj
(Besucht am: 02.05.2024)

Listingverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Natural Language Processing
Natural Language Processing ist ein Teilbereich der Künstlichen Intelligenz (KI) mit dem Ziel menschliche, natürliche Sprache für den Computer verständlich zu machen. Dabei verwendet NLP Methoden aus verschiedenen Bereichen wie Computerlinguistik, Maschinelles Lernen und Statistik, um die Bedeutung eines Satzes zu verstehen, die Absicht und auch die Gefühle.

Feature
Im Kontext der Text Klassifizierung ist ein Feature eine Eigenschaft anhand der die Klassifizierung vorgenommen werden kann. Features können direkt im Text enthalten sein oder aus dem Text erzeugt werden.

Code Editor
Ein Code Editor ist ein Texteditor, der speziell für die Bearbeitung von Quellcode entwickelt wurde. Er bietet verschiedene Tools, um die Nutzung zu erleichtern.

A-priori-Wahrscheinlichkeit
In der Naturwissenschaft ist die A-priori-Wahrscheinlichkeit eine durch allgemeines Vorwissen oder eine vernünftig erscheinende Grundannahme vermuteter Wahrscheinlichkeitswert.

Bias und Varianz
Bias im Bereich der Künstlichen Intelligenz ist eine systematische Abweichung von der Realität, die dazu führt, dass Vorhersagen zu hoch oder zu niedrig ausfallen.
Die Varianz ist die Schwankung einer Modellvorhersage bei unterschiedlichen Datensätzen. Je höher die Varianz, desto stärker reagiert das Modell auf Veränderungen in den Trainingsdaten.

Rastersuche
Die Rastersuche – auch „Grid Search“ – ist eine Methode zur Optimierung von Hyperparametern; externen Konfigurationsvariablen, die für das Training im Maschinellen Lernen verwendet werden.
Die Rastersuche wird verwendet, um die besten Hyperparameter für ein gegebenes Modell zu finden, indem systematisch eine vordefinierte Menge möglicher Hyperparameterkombinationen durchlaufen wird.

CART-Methode
Cart (**C**lassification **a**nd **R**egression **T**rees) ist ein Algorithmus, der bei Entscheidungsbäumen eingesetzt wird und zur Entscheidungsfindung dient.

Bag-of-Words
Das Bag-of-Words Modell ist eine einfache Technik für die Feature Extraktion bei der Text Klassifizierung. Ein Bag-of-Words wird durch Zählen des Vorkommens einzigartiger Merkmale wie Wörtern und Symbolen in einem Dokument konstruiert.

TF-IDF
TF-IDF (**T**erm **F**requency and **I**nverse **D**ocument **F**requency) ist eine Methode zur Beurteilung der Relevanz von Termen in einem Dokument.

Backpropagation
Backpropagation – auch Rückwärtspropagation – ist ein Verfahren zum Einlernen künstlicher neuronaler Netze und gehört zu den überwachten Lernverfahren. Backpropagation ist ein Spezialfall eines allgemeinen Gradient Verfahrens in der Optimierung und basiert auf dem mittleren quadratischen Fehler.

Kernel
Kernel sind Filter, die in der digitalen Bildverarbeitung z.B. für die Faltung verwendet werden. Es handelt sich um quadratische Matrizen ungerader Abmessung in unterschiedlichen Größen.

ReLu Funktion
Die ReLu (**R**ectified **L**inear **U**nit) Funktion wird häufig in künstlichen neuronalen Netzen für Klassifizierungsaufgaben eingesetzt. Die Funktion gibt eine Null zurück, wenn sie eine negative Eingabe empfängt. Erhält sie eine positive Eingabe, gibt die ReLu Funktion diesen Wert weiter.

Word Embedding
Word Embedding – auch Worteinbettung - sind eine Reihe von Lernmethoden, die darauf abzielen, Wörter in einem Text durch Vektoren reeller Zahlen zu repräsentieren.

Intent
In der Text Klassifizierung ist ein Intent (Absicht) der Zweck und das Ziele, die ein Text zum Ausdruck bringt. Im Falle dieser Arbeit etwa die Absicht eine bestimmte Programmstruktur erstellen zu lassen.

Entity
Entities – Entitäten – sind zusätzliche Informationen, die aus einem Text gezogen werden können.

Strahlensuche
Die Strahlensuche – auch Beam Search – ist ein auf Heuristik basierender Suchalgorithmus. Er stellt eine Erweiterung der Bestensuche (Best First Search) dar.

Abkürzungsverzeichnis

SA Sprachassistent

OOB Out-of-Bag

KNN Künstliches Neuronales Netzwerk

RNN Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

FW Faster Whisper

RF Random Forest

NB Naive Bayes

Genutzte Software

PyCharm

Für die Entwicklung des Prototyps wurde die Entwicklungsumgebung PyCharm Community Edition 2024.1.1 verwendet.

Github

Während der Entwicklung wurde der Webdienst Github als Versionsverwaltung benutzt.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht.

Diese Erklärung erstreckt sich auch auf in der Arbeit enthaltene Grafiken, Skizzen, bildliche Darstellungen sowie auf Quellen aus dem Internet.
Die Arbeit habe ich in gleicher oder ähnlicher Form auch auszugsweise noch nicht als Bestandteil einer Prüfungs- oder Studienleistung vorgelegt.
Ich versichere, dass die eingereichte elektronische Version der Arbeit vollständig mit der Druckversion übereinstimmt.

Rene Wentzel
Matrikel Nummer: 5090763
Bremen, den 10.06.2024