Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

NAO Spracherkennung

Studienprojekt

Name, Vorname:	Anna Stabel,
	Caroline Sarah Schäfer,
	Sofie Wagner,
Semester:	WiSe 2024/25
Fachbereich:	Duales Studium (FB 2)
Studiengang:	Duales Studium Informatik
Modul:	Studienprojekt II
Betreuer Hochschule:	Dagmar Monet Diaz
Anzahl der Wörter:	0

Unterschriften

Ort, Datum	Anna Stabel
Ort, Datum	Caroline Sarah Schäfer
Ort, Datum	Sofie Wagner
Ort, Datum	Ausbilder*in SAP
Ort, Datum	Ausbilder*in HZB

Abstract

Hier kommt das Abstract hin

Inhaltsverzeichnis

Akronyme

KG Knowlegde Graphs

HWR Hochschule für Wirtschaft und Recht

OpenIE Open Information Extraction

ML Maschinelles Lernen

TSP Traveling Salesman Probelem

LSA Latent semantic analysis

SVD Singular Value Decomposition

DB Datenbank

POS-Tags Part-of-Speech-Tags

1 Einleitung

Einleitung

2 Problemstellung

TODO

3 IST Zustand des Programms

Das vorliegende Programm hat das Ziel, Antworten zu spezifischen Fragen aus Audioaufnahmen zu extrahieren. Im Folgenden wird der aktuelle Stand der Implementierung beschrieben. Die wichtigsten schritte erfolgen innerhalb der `get_answer` Methode.

3.1 Programmbeschreibung

3.1.1 Datenbank und Schlüsselwörter

Das System basiert auf drei Datenbank (DB): der Schlüsselwörter-Datenbank, der Synonyme-Datenbank und der Antworten-Datenbank. Die Schlüsselwörter (`generic_term`) sind Begriffe, die mit hoher Wahrscheinlichkeit darauf hinweisen, dass eine bestimmte Frage gestellt wurde. Ein Schlüsselwort kann mehrere Synonyme besitzen. Synonyme werden mithilfe von Oberbegriffen gruppiert, die durch SQL-Abfragen ermittelt werden.

Um eine effektive Zuordnung zu gewährleisten, sind Schlüsselwörter zusammen mit den zugehörigen Antworten in einer Tabelle organisiert. Jede Antwort wird durch eine eindeutige `case_id` identifiziert, was eine schnelle und effiziente Suche ermöglicht.

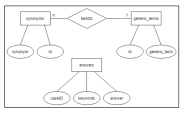


Abbildung 1: Datenbank-Diagramm

3.1.2 Satzanalyse

Die Verarbeitung der Audiodaten erfolgt durch eine Tokenisierung, bei der nicht signifikante Wörter (z. B. Artikel) aus dem Satz entfernt werden. Die verbleibenden Wörter werden mit dem NLP-Paket `spaCy` analysiert, das Part-of-Speech-Tags (POS-Tags) verwendet, um die jeweilige Wortart zu identifizieren. Die Gewichtung der Schlüsselwörter basiert auf der Häufigkeit ihres Auftretens. Dies wird in der Methode `distinct_list` umgesetzt:

$$\text{Gewichtungswert} = 1 - \frac{\text{Häufigkeit des Wortes}}{\text{Gesamtanzahl aller Schlüsselwörter}}$$

Dieses Verfahren ermöglicht eine priorisierte Analyse seltener, aber wahrscheinlich bedeutungsvoller Wörter. Seltener auftretende Wörter haben in der Regel eine höhere Bedeutung für die Zuordnung.

3.1.3 Antwortermittlung

Zur Bestimmung der passenden Antwort werden die analysierten Wörter mit der Datenbank abgeglichen. Die `case_id` mit den meisten Treffern wird durch die Methode `count_ids` ermittelt. Die Methode akzeptiert eine Liste von Wörtern als Eingabe und iteriert über diese. Mithilfe von `db_connector.get_caseIDs_by_keywords(word)` werden die zugehörigen `case_ids` für jedes Schlüsselwort aus der Datenbank abgerufen. Für jede gefundene `case_id` wird die Methode `check_list` aufgerufen, um die Gewichtung zu aktualisieren. Dabei wird das aktuelle Gewicht für jede `case_id` gespeichert. Schließlich wird die `case_id` mit dem höchsten Gewicht zurückgegeben. Falls mehrere `case_ids` mit identischer Gewichtung existieren, erfolgt eine zusätzliche Überprüfung anhand der primären Schlüsselwörter. Die finale Antwort wird mit der Methode `get_answer_from_db` abgerufen, welche die Antwort für die ermittelte `case_id` aus der Datenbank extrahiert und ausgibt.

3.2 Leistungsfähigkeit

Die momentane Performance des Algorithmus wird mit Hilfe der Python-Bibliothek *time* gemessen. Diese Bibliothek bietet Funktionen, die es Entwicklern ermöglichen, mit Zeiten zu arbeiten und verschiedene Zeitoperationen durchzuführen. Um die Zeit zu messen, die seit den Funktionsaufrufen vergangen ist, wird die Differenz zwischen der Endzeit und der Startzeit berechnet. Dazu wird der Timer mit der Funktion `time_nc()` initialisiert. Diese Funktion gibt die Anzahl der Nanosekunden seit der Initialisierung des Timers als Integer zurück **[pythonTimer]**. `time_nc()` wurde gewählt,

um potentiellen Präszisionsfehlern, die aufgrund einer Floating Nummer passieren können, zu vermeiden. Die folgende Tabelle zeigt die Messergebnisse:

Frage	**Antwort-Algorithmuszeit (ns)**	**Transkriptionszeit (ns)**	**Gesamtzeit (ns)**
Wie oft muss man einen PTB schreiben?	33659000	2432785000	2474059000
Welche Fachbereiche gibt es in der HWR?	18816000	388357000	411529000
Wie kann ich mich für ein duales Studium bewerben?	14467000	360204000	379107000
Erzähl mir über den Informatik-Studiengang.	13036000	308227000	325151000
Wann wurde die HWR Berlin gegründet?	22216000	379835000	405591000
Erzähl mir was über die HWR.	17349000	316136000	337483000
Welche Voraussetzungen gibt es für ein Informatik-Studium?	12492000	405805000	422220000
Was ist eine Studienarbeit?	13245000	291517000	308290000
Was ist ein Studiengang?	12957000	305227000	320896000
Was bedeutet PTB?	14044000	307653000	323718000

Durchschnittszeiten:

- Antwort-Algorithmuszeit: 17228100 ns
- Transkriptionszeit: 549574600 ns
- Gesamtzeit: 570804400 ns

Im folgenden wird die Antwort-Algorithmuszeit analysiert, um Verbesserungspotential im Suchalgorithmus festzustellen. Die Funktionsabfolge wurde in **??** erläutert.
Funktion `db_connector.get_generic_term`:

- Beschreibung: Für jedes relevante Wort wird eine Datenbankabfrage durchgeführt, um dessen generische Form zu finden. Dies ist der teuerste Schritt, da jede Abfrage Zeit beansprucht und die Abfragen in einer Schleife ausgeführt werden.
- Zeitaufwand: Dies hängt von der Anzahl der Wörter und der Geschwindigkeit der Datenbank ab, typischerweise im Bereich von Millisekunden.
- Laufzeitkomplexität: $O(n \cdot m)$, wobei n die Anzahl der Wörter und m die durchschnittliche Zeit für eine einzelne Datenbankabfrage ist. Falls für jedes Wort eine Abfrage ausgeführt wird, summieren sich die Datenbankoperationen linear zur Anzahl der Wörter.

Funktion `caseID = counter.count_ids`:

- Beschreibung: Dieser Schritt durchsucht die IDs basierend auf Gewichtungen der Wörter und sucht nach dem relevantesten caseID. Die Laufzeit hängt von der Implementierung von `count_ids` und der Anzahl der Wörter ab.
- Zeitaufwand: Variiert von Millisekunden bis Sekunden, abhängig von der Datenbank und der Anzahl der IDs in der Datenbank.
- Laufzeitkomplexität: $O(n \cdot k)$, wobei n die Anzahl der relevanten Wörter und k die Anzahl der verfügbaren caseIDs in der Datenbank ist.

Funktion `db_connector.get_answer_from_db`:

- Beschreibung: Die Funktion ruft die Antwort basierend auf einem einzelnen caseID ab. Da nur eine Datenbankabfrage erforderlich ist, ist die Laufzeit unabhängig von der Eingabelänge.
- Zeitaufwand: Typischerweise Millisekunden, aufgrund nur einer Abfrage.
- Laufzeitkomplexität: $O(1)$, da nur eine einzige Datenbankoperation ausgeführt wird, um die Antwort abzurufen.

Die zeitintensivsten Vorgänge sind demnach die Abfrage der generischen Form des Wortes und die Gewichtung der relevantesten caseIDs. Das Ziel dieser Studienarbeit besteht folglich darin, die Laufzeit der zuvor beschriebenen Funktionen zu mindern bzw. den Suchalgorithmus so zu modifizieren, dass diese obsolet werden.

4 NLP Suchalgorithmen

TODO: Intro für NLP Suchalgorithmen

4.1 Knowledge Graphs

4.1.1 Beschreibung

Knowlegde Graphs (KG) stellen eine strukturierte Darstellungsform von Informationen dar, welche aus unstrukturierten Texten gewonnen werden. Sie setzen sich aus Informationsentätiten, welche Knoten genannt werden, und Beziehungen zwischen den Informationsentätiten, welche Kanten genannt werden, zusammen. Diese werden aus Textdaten abgeleitet. Dadurch wird die Integration, der Abruf und die Analyse von Informationen erleichtert [**Hojas-Mazo2018A**]. Um einen KG aus einem Text zu konstruieren, werden verschiedene Methoden. Beispiele dafür sind Techniken wie Open Information Extraction (OpenIE), Maschinelles Lernen (ML) und semantische Analyse zum Einsatz [**OpenIEbased**]. Die strukturierte und semantische Darstellungsform von Informationen, wie sie in Knowledge Graphen erfolgt, ermöglicht eine präzisere und effizientere Beschaffung von textbasierten Informationen [**Dietz2017Utilizing**]. Dies wird durch folgende Faktoren begünstigt:

- Die verbesserte Textdarstellung ermöglicht die Rückgabe reichhaltiger semantischer Strukturen.
- Die automatische Strukturierung von Textinhalten wird durch KGs signifikant vereinfacht, da eine Kategorisierung von Textinformationen in kürzerer Zeit erfolgt [**Hojas-Mazo2018A**].
- Die Berechnung der semantischen Ähnlichkeit, welche eine Steigerung der Effizienz und Genauigkeit von Suchergebnissen zum Ziel hat, kann mittels KGs ohne großen Aufwand durchgeführt werden [**Wang2018Information**].
- Die Integration von KGs in multimediale Modelle, wie beispielsweise Richpedia, ermöglicht die Nutzung zusätzlicher Ressourcen, beispielsweise visueller Art, für die semantische Suche sowie die Beantwortung von Fragen [**Wang2020Richpedia:**].

4.1.2 Implementierung

Die Implementierung von KGs erfolgte unter Zuhilfenahme der Python-Bibliothek NetworkX in Kombination mit spaCy, einer bereits zuvor im Code verwendeten Python-

Bibliothek. Auf spaCy wurde bereits zuvor eingegangen. NetworkX ist ein Python-Paket, welches die Erstellung, Bearbeitung und Untersuchung der Struktur, Dynamik und Funktionen komplexer Netzwerke ermöglicht. Die Python-Bibliothek wurde als Werkzeug zur Umsetzung der vorliegenden Anforderungen gewählt. Das Paket ermöglicht es Entwicklern verschiedene Arten von Graphen (bspw. Diagraphen und Multigraphen) aus diversen Datenstrukturen wie Text oder XML zu erstellen oder zu generieren. Zudem können Operationen wie das Löschen von Knoten an Graphen durchgeführt, Graphen analysiert (beispielsweise die Anzahl der Knoten gezählt) oder Algorithmen wie z.B. zum Lösen des Traveling Salesman Probelem (TSP) implementiert werden [**networkX:Docs**].

4.2 Ontologie

4.2.1 Beschreibung

Ontologie im NLP beinhaltet die Verwendung eines strukturierten Rahmens zur Repräsentation von Wissen innerhalb einer bestimmten Domäne und erleichtert Aufgaben. Die Ontologie zeigt Eigenschaften und Beziehungen zwischen einer Reihe von Konzepten und Kategorien innerhalb der Domäne auf. Ein Beispiel für die Anwendung von Ontologie wäre, dass eine Maschine die Bedeutung des Wortes „Diamond“ in Bezug auf einen Baseballspieler, einen Juwelier oder eine Kartenfarbe genau interpretieren kann. In NLP wird Ontologie zum Beispiel zur Wiederauffindung von Informationen, dem Beantworten von Fragen und dem Annotieren von Entitäten eingesetzt, da sie semantisch angereicherte Antworten in ihren Domänen liefern [**Adelkhah2019The**] [**Naderian2018Ontology**].

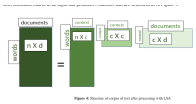
4.2.2 Implementierung

4.3 Vektor Suche

4.4 Latent semantic analysis

Latent semantic analysis (LSA) ist eine statistische Methode zur Schätzung der Wortbedeutungen. Diese Bedeutung basiert auf zugrunde liegenden Konzepten. Diese Konzepte werden durch Matrixoperationen extrahiert, die auf beobachteten Mustern der Wortverwendung basieren. Die grundlegende Idee hinter LSA ist, dass die Bedeutung jedes Textabschnitts als Summe der Bedeutungen der darin enthaltenen Einzelwörter ist, während eine Sammlung von Dokumenten (ein „Korpus“) als ein Ge-

leichhungssystem dargestellt wird, welches die Ähnlichkeit von Wörtern zu anderen Wörtern und Dokumenten zu anderen Dokumenten zueinander bestimmen kann. LSA stellt die Beziehung zwischen Dokumenten und Begriffen durch eine Term-Dokument-Matrix dar, die durch Singular Value Decomposition (SVD) weiter in das Produkt von drei Matrizen zerlegt wird. SVD ist das mathematische Werkzeug hinter LSA [**TheUseofLatentSemanticAnalysis**]. Es ist eine grundlegende Matrixfaktorisierungstechnik in der linearen Algebra mit vielseitigen Anwendungsbereichen [**Paige1981Towards**]. Es zerlegt eine Matrix A in drei Matrizen (siehe Bild). Für eine



gegebene Abfrage transformiert LSA diese in einen Pseudo-Dokumentenvektor und berechnet die Ähnlichkeiten mithilfe des SVD-Ergebnis aus der Term-Dokument-Matrix zwischen der Abfrage und dem durchsuchten Dokumenten [**SystematicReviewofSemanticA**
. Im Gegensatz zu präzisen Abgleichmethoden wird die Matrix durch SVD zerlegt, was sie in einen neuen Raum mit niedriger Dimension komprimiert.SVD kann nicht nur die Datenmenge reduzieren, sondern auch die zugrunde liegenden Beziehungen zwischen Begriffen erkennen. Aus den oben genannten Gründen, gilt LSA als eine sehr flexible Technik ist und wird oft in der Sprachsuche benutzt wird [**TextMiningUsingLatentSem**
.

1. **Fett 1** Nummerierte liste 1
2. *Kursiv 1* Nummerierte liste 1
3. Nummerierte liste 1

Eine Matheformel (Satz des Pythagoras):

$$a^2 + b^2 = c^2$$

wobei a und b die Längen der Katheten eines rechtwinkligen Dreiecks sind, und c die Länge der Hypotenuse.

```
1    Python Code Section 1
```

Listing 1: Pip Update

"'Deutsche Anführungszeichen"' **??** referenz Zu Label1 „Anführungszeichen unten, Anführungszeichen oben“

?? Bild Referenz

Column1	**Column2**
row1	row2

Tabelle 2: Your table caption here

5 Fazit

Fazit

5.1 Ergebnis

Ergebnis

5.2 Ausblick

Ausblick

Abbildungsverzeichnis

AI-Verzeichnis

- ChatGPT 4o und o1-preview
- Consensus AI
- Perplexity
- DeepL

Ehrenwörtliche Erklärung

Wir erklären hiermit ehrenwörtlich:

1. dass wir unsere Studienarbeit selbstständig verfasst habe,
2. dass wir die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe,
3. dass wir unsere Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Wir sind uns bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum	Anna Stabel
Ort, Datum	Caroline Sarah Schäfer
Ort, Datum	Sofie Wagner