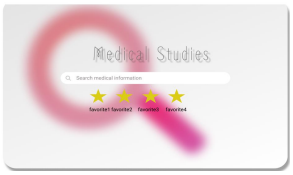
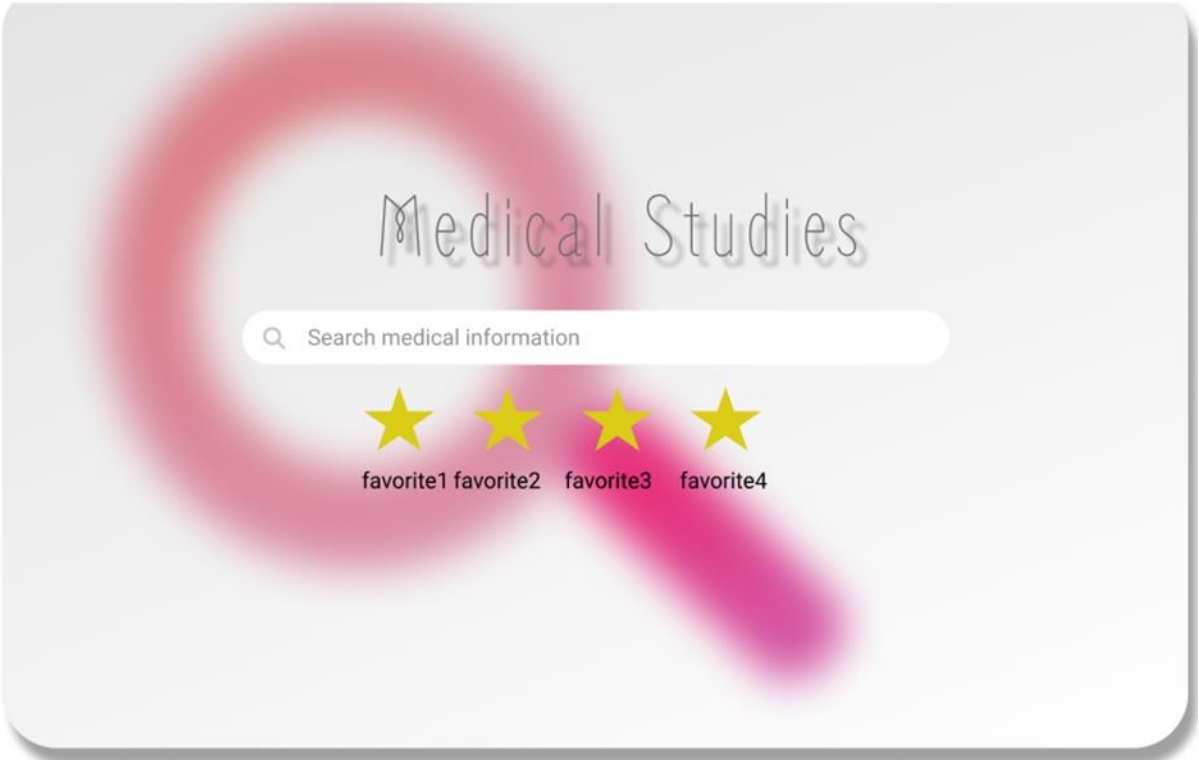


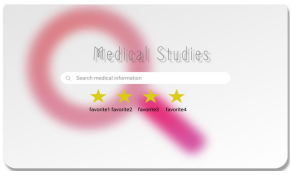
FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Medical Studies – Semesterarbeit
Optimierung eines Suchprogrammes

Dozentin: Ursula Deriu
Modul: BSC_INF_IR_NLP

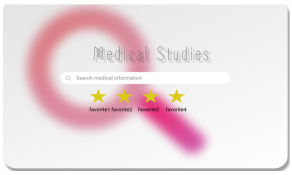
Autor: Chantale Gihara
Information Retrieval und Natural Language Processing





FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Inhaltsverzeichnis



1 Einleitung

Im Rahmen des Moduls Information Retrieval und Natural Language Processing sollte eine Semesterarbeit über die verschiedenen Themen aus den Präsenzveranstaltungen erstellt werden. Der Code ist unter https://github.com/Chantifa/medical_studies einsehbar.

1.1 Der Auftrag

Nach der ersten PVA mussten wir 2 Korpusse finden, welche die folgenden Konditionen erfüllt:

- Bilingual oder multilingual,
- Datenset von 5000 Daten.

Natürlich sollten es Daten sein, welche uns interessieren, damit wir auch spannende Queries daraus ziehen können.

Nach der 2. PVA muss die Semesterarbeit gestartet werden. Ein erster Index muss aus den Daten erstellt werden und wir sollten eine Idee haben, wie das GUI aussehen könnte. Hier wird noch kein GUI implementiert, sondern ein einfacher Prototyp erstellt z.B. mit Figma. Mit Solr müssen die Daten indexiert werden.

Im dritten Teil sollte ein Knowledge-Graph umgesetzt werden und zum Abschluss sollte noch Signal inkludiert werden, wenn möglich auch Learning to Rank und eine Automatische Klassifikation implementiert werden.

Im letzten Teil könnte noch eine Embedded Lösung verarbeitet werden, diese ist aber auf freiwilliger Basis.

1.2 Theorie zu Solr

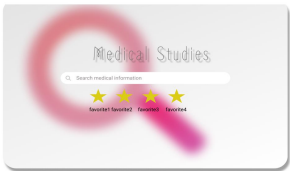
Gemäss (Tutorialpoint, 2022) ist Solr eine Open-Source-Suchplattform, die für die Entwicklung von Suchanwendungen verwendet wird. Sie wurde auf der Grundlage der Lucene Volltextsuchmaschine entwickelt. Solr ist unternehmenstauglich, schnell und hoch skalierbar.

Yonik Seely entwickelte Solr im Jahr 2004, um der Unternehmenswebsite von CNET Networks Suchfunktionen hinzuzufügen. Im Januar 2006 wurde es zu einem Open-Source-Projekt der Apache Software Foundation.

Solr kann zusammen mit Hadoop verwendet werden. Da Hadoop eine grosse Datenmenge verarbeitet, hilft Solr bei der Suche nach den benötigten Informationen aus einer solch grossen Quelle. Solr kann nicht nur für die Suche, sondern auch für die Speicherung verwendet werden. Wie andere NoSQL-Datenbanken ist es eine Technologie zur Verarbeitung und Speicherung nicht-relationaler Daten.

Kurz gesagt, Solr ist eine skalierbare, sofort einsatzbereite Such-/Speichermaschine, die für die Suche in grossen Mengen textbasierter Daten optimiert.

In meiner Arbeit arbeite ich mit der Solr Version 9.0.0 auf einer Windowsumgebung.



2 Korpus und Indexierung

Ich habe sehr lange für einen Korpus gesucht, ich wollte ein Themenkorpus finden, mit welchem ich mich identifizieren kann. Es soll einem Zweck dienen, eine solche Indexierung und Suchqueries vorzunehmen. Ich habe mich für einen Medizinischen-Daten-Korpus entschieden, weil mir von Freunden zu Ohren kam , dass es da viel zu wenig gibt. Nach langer Suche bin ich auf folgenden Korpus gestossen: https://clinicaltrials.gov/AllPublicXML.zip ist eine Datenbank mit privat und öffentlich finanzierten klinischen Studien aus aller Welt. Die Schwierigkeit hier ist definitiv die Mehrsprachigkeit. Ich konnte keinen nützlichen Mehrsprachigen Korpus zur Medizin finden. Da die Nutzer dieser Informationssuche zum grossen Teil Ärzte und Ärztinnen sein werden, erhoffe ich mir, dass Englisch ausreichen wird. Die Texte sind medizinischer Herkunft, da kann ich nicht einfach eine einfache Deepl Übersetzung nehmen, um das Ganze zu übersetzten, dass wäre sehr unprofessionell.

Ich kann also den Teil der Mehrsprachigkeit im Auftrag nicht einhalten und habe mich aus Interesse trotzdem für diesen Korpus entschieden.

1.1 Nutzer des «Medical – Studies Suche»

Bei einer Suchmaschine geht es darum zu verstehen, wer eigentlich meine Suchmaschine benutzt und was denn genau von diesem Nutzer als Ergebnis erwartet wird.

1.1.1 Ärzte oder Arbeiter in der Medizin

Die Hauptnutzer sind Ärzte, welche Studien zum Krankheitsbild eines Patienten finden möchten. Ein Arzt wird die Suche sehr spezifisch angehen. Er wird nach einer bestimmten Erkrankung suchen. Hier sollte die Suche eine sehr konkrete Studie zurückgeben, welche den gesuchten Term auch wirklich enthält.

1.1.2 Erkrankte Menschen / diagnostizierter Patienten

Da der Browser öffentlich für alle zugänglich ist, können sich natürlich auch erkrankte Personen über die Studien informieren und zu Ihrer Erkrankung eine mögliche Studie dazu finden wollen. Ein Patient wurde schon diagnostiziert, somit wird er auch nach einer Studie seiner diagnostizierten Krankheit suchen. Somit sollte auch hier die Suche genau ein Dokument zurückgeben, welches auch wirklich den gesuchten Satz/Term enthält.

Hier könnte sicher Signal-Boosting helfen, Studien dieser Themata, welche schon viel geklickt wurden zu derselben Erkrankung auszugeben.

1.1.3 Medizinisch interessierte Menschen

Medizinisch interessierte Menschen werden sich auch für die Fälle der verschiedenen Studien interessieren und sich auf dem Browser umsehen. Die Suche wird sich anders verhalten als bei einem Arzt oder eines diagnostizierten Patienten. Die Suche wird voraussichtlich etwas allgemeiner im Thema ausfallen als bei einer spezifischen Suche eines Arztes oder einer erkrankten Person, die schon

diagnostiziert wurde. Die Anfrage darf also auch Nachbarthemen, oder Dokumente in einer Kategorie enthalten.

1.2 Prototyp

Beim Erstellen eines Gui für eine Suche habe ich mich einfach von Google inspirieren lassen.

Eine einfache Sucheingabe, Favoriten, sowie Dokumente die ich bei vorherigen Suchen festgelegt habe und eine erweiterte Suche gemäss Kategorien.

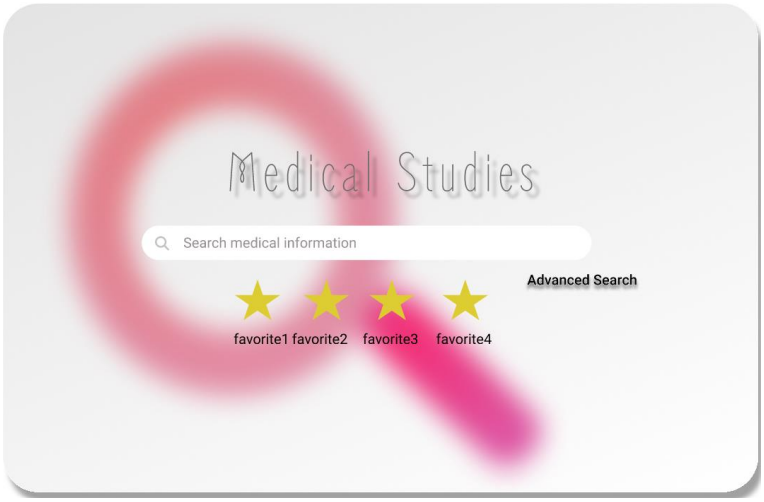


Abbildung 1: Frontpage Search medical information

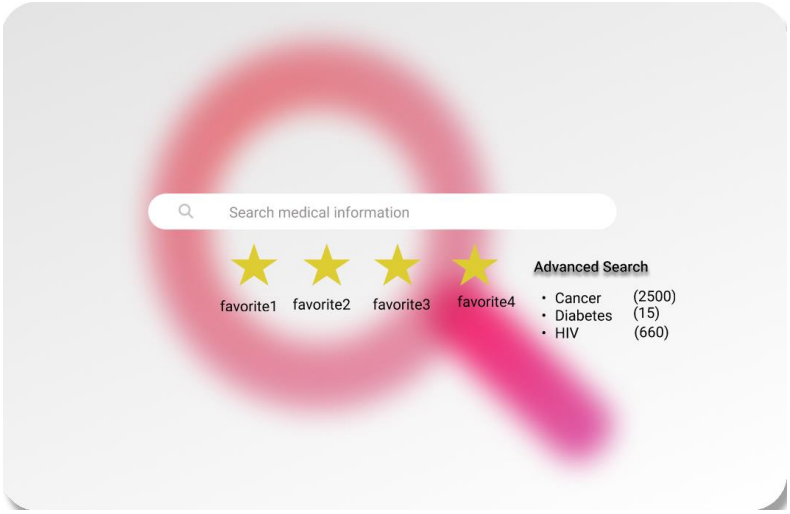
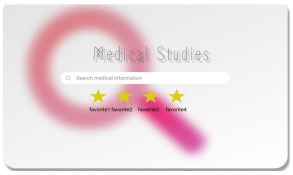


Abbildung 2:Advanced Search - Anzeigen der Kategorien



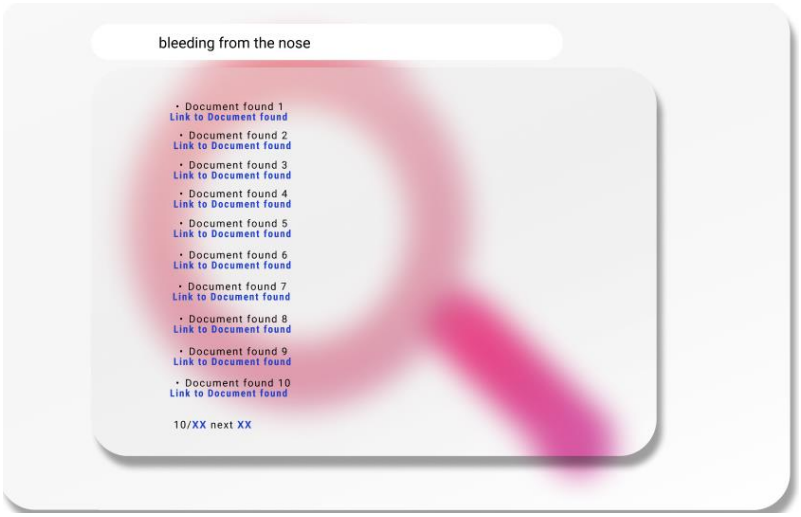


Abbildung 3: Gefundene Dokumente gemäss Eingabe

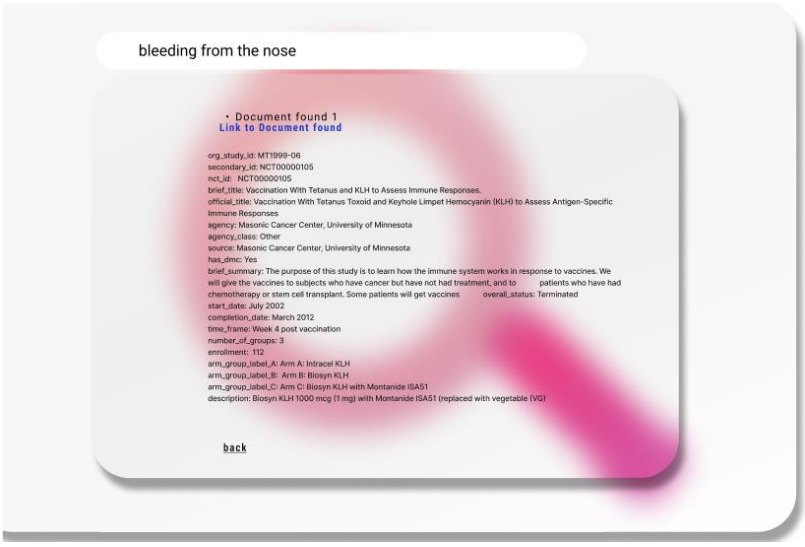


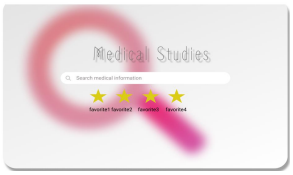
Abbildung 4: Anzeige des ausgewählten Dokuments

Der Prototyp ist über folgenden Link verfügbar:

https://www.figma.com/file/7ld7d3JNQ84o4scUnuGMXc/Search-Medical-Studies-(Copy)?node-id=0%3A1&t=MWdYrlRIUHUlbv7N-1

Der Prototyp sollte auch noch folgende Kategorien enthalten:

1. Geschlechtsauswahl (je nach Krankheit kann das von Vorteil sein, die richtige Studie zu finden)
2. Krebs (Es gibt viele verschiedene Krebsarten, die bekanntesten sollten hier aufgelistet werden zur Auswahl)
3. Alter (Je nach Alter kommen gewisse Studien gar nicht in Frage)



1.3 2.2 Indexierung

Bei der Indexierung gab es einige Schwierigkeiten, hier musste zuerst das Datenset transformiert werden, damit Solr dieses überhaupt einlesen konnte.

Dabei wurde folgender Index vorgeben:

```
<add>
    <doc>
        <field name='clinical_study'> </field>
        <field name='required_header'> </field>
        <field name='download_date'>    </field>
        <field name='link_text'>.</field>
        <field name='url'></field>
        <field name='id_info'> </field>
        <field name='org_study_id'> </field>
        <field name='nct_id'></field>
        <field name='brief_title'></field>
        <field name='sponsors'></field>
        <field name='lead_sponsor'></field>
        <field name='agency'></field>
        <field name='agency_class'></field>
        <field name='source'></field>
        <field name='brief_summary'> </field>
        <field name='textblock'></field>
        <field name='overall_status'></field>
        <field name='start_date'></field>
        <field name='completion_date'></field>
        <field name='study_type'> </field>
        <field name='has_expanded_access'> </field>
        <field name='condition'> </field>
        <field name='eligibility'></field>
        <field name='criteria'> </field>
        <field name='gender'> </field>
        <field name='minimum_age'></field>
        <field name='maximum_age'></field>
    </doc>
</add>
```

Die Daten waren in einem XML gemäss Vorgaben von einem bsd. Ich musste dafür Python Code schreiben, um die über 5000 XML-Dokumente einzulesen. Da einige nicht dem Standard entsprachen, kamen immer wieder Fehlermeldungen. Diese wurden einfach weggelassen und nicht eingelesen. In der realen Welt kann man das nicht, dabei muss das ganze Dokument analysiert werden. Der Korpus besteht aus über 100000 Datensätze, das wäre für meinen Rechner wie auch für die Analysen, die wir hier machen zu gross und zu umfangreich. Deshalb habe ich mich auf 5000 Dokumente beschränkt, wie im Auftrag vorgegeben.

1.4 In Solr - Suchalgorithmus BM25

Solr verwendet in der Version 9 folgender BM25 Algorithmus. Die Definition gemäss (Amati, 2009) ist folgende:

BM25 ist eine Ranking-Funktion, die eine Reihe von Dokumenten auf der Grundlage, der in jedem Dokument vorkommenden Suchbegriffe einstuft, unabhängig von der Beziehung zwischen den Suchbegriffen innerhalb eines Dokuments. Es handelt sich nicht um eine einzelne Funktion, sondern um eine ganze Familie von Bewertungsfunktionen mit leicht unterschiedlichen Komponenten und Parametern. Sie wird von Suchmaschinen verwendet, um übereinstimmende Dokumente nach ihrer Relevanz für eine bestimmte Suchanfrage einzustufen, und wird oft als "Okapi BM25" bezeichnet, da das Okapi Information Retrieval-System das erste System war, das diese Funktion implementierte. Die BM25-Retrievalformel gehört zur BM-Familie von Retrieval Modellen, das heisst die Gewichtung eines Terms t in einem Dokument d.
Bis vor der 8. Version hatte Solr TF-IDF [1]verwendet.

1.5 Goldstandard

Im Zusammenhang der Daten bezieht sich der Goldstandard auf einen manuell aufbereiteten Standard oder einen überprüften Datensatz, der die objektive Wahrheit so genau wie möglich darstellt.

Einer das wichtigste Nutzen des Goldstandards ist das Trainieren von Daten zur Erkennung von Datenmustern in verschiedenen Datentypen mit Hilfe von Algorithmen für maschinelles Lernen. Goldstandards können auch verwendet werden, um die Leistung solcher Algorithmen zu bewerten.

Wir werden den Goldstandard so aufbereiten, dass die ca. 5000 Daten die erwünschten Resultate zurück liefern:

Tabelle 1:Goldstandard

Nutzer	Query	Erwartetes Dokument	Id
Interessierter / nicht diagnostizierter Patient	Ovarian cancer	• A Multi-Institutional Phase II Study of Cyclophosphamide, Paclitaxel, Cisplatin With G-CSF for Patients with Newly Diagnosed Advanced Stage Ovarian Cancer • A Phase I Study of Taxol, Cisplatin, Cyclophosphamide and Granulocyte Colony-Stimulating Factor (G-CSF) in	• 6bbc514b-e457-4538-9498-db780b535a0b • d27b3faa-d520-4ec3-9097-d1e1fd194b9b

[1] TF-IDF: Term Frequency - Inverse Dokumenthäufigkeit. |D| ist die Gesamtzahl aller Dokumente, t ist der Begriff, und DF(t) ist die Anzahl aller Dokumente, die den Begriff enthalten. Je niedriger die Zahl ist, desto unbedeutender ist der Begriff, und je höher, desto mehr sollte ein Begriff in einer Abfrage für die Relevanzbewertung zählen.

		Previously Nontreated Ovarian Cancer Patients	
Arzt	lung cancer Stage 3	• Chemotherapy Plus Radiation Therapy with or Without Surgery in Treating Patients with Stage IIIA Non-small Cell Lung Cancer • Vinorelbine + Cisplatin or No Further Therapy in Non-small Cell Lung Cancer That Has Been Surgically Removed"	• 09d7056b-62cd-426f-ad02-8a75ebe4f7e0 • 57c5a4ed-2c2b-42c9-82d9-6951dce0be22
Diagnostizierter Patient	prostate cancer phase 2	• A Randomized Phase II Study of a PSA-Based Vaccine in Patients with Localized Prostate Cancer Receiving Standard Radiotherapy • Biological Therapy in Treating Patients with Prostate Cancer	• 71f22d71-cf80-47b6-81b8-e8a32b8b42a3 • b84d8011-af6b-49c5-b877-c4630f278255

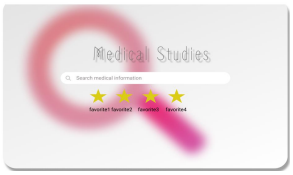
Das heisst von den 5000 Dokumenten sollte bei einem Versuch mit dem Query genau dieses Resultat zurück geliefert werden. Dafür müssen verschiedene Suchalgorithmen verwendet werden, die ich in dem nächsten Kapitel näherbringen möchte.

1.6 Validierungsverfahren

Um zu verstehen wie sich das Query mit den verschiedenen Schritten verhält kann nach jeder Implementation die Suche und der Output vergleichen werden. Dafür verwende ich die Cranfield Evaluation Methodologie. Diese Methode setzt voraus, dass für jedes Query eine Liste mit relevanten Dokumenten vorhanden ist. Aus dem Resultat der Abfrage, genauer der Anzahl relevanter und nicht relevanter Dokumente, kann die Precision und der Recall für das jeweilige System berechnet werden.

Die Precision sagt die Relevanz der Antworten voraus und der Recall, wie viele relevante Antworten vom Total der relevanten Dokumente erhalten wurden. (Shafi, 2014)

Precision:



$$\frac{\text{TRUE POSITIVES}}{\text{TRUE POSITIVES} + \text{FALSE POSITIVES}}$$

Abbildung 5:Formel Precision (Shafi, 2014)

Recall:

$$\frac{\text{TRUE POSITIVES}}{\text{TRUE POSITIVES} + \text{FALSE NEGATIVES}}$$

Abbildung 6: Formel Recall (Shafi, 2014)

2 Systemarchitektur

Der erste Schritt bestand darin den Korpus in ein Solr Format zu bringen, dazu musste ein eine Transformationsfunktion erstellt werden. Danach konnten die XML in Apache Solr eingelesen werden und indexiert werden.

Um die Sucheingabe zu prüfen, musste eine Query-Funktion implementiert werden, die die Abfrage im Solr abholt. Nun müssen die verschiedenen Rankings eingeholt werden. Dabei handelt es sich um den "Knowledge Graph", "Classification automation" und den "Signal Booster" und zum Schluss läuft dann noch die Embedded library darüber mit ClinicalBERT - Bio + Clinical BERT Model.

Was in den verschiedenen Schritten passiert wird im kommenden Kapitel beschrieben. Das GUI wurde hier nicht implementiert, weil es nicht in den Anforderungen enthalten war. Es geht hauptsächlich darum zu sehen, was den verschiedenen Schritten passiert und welchen Nutzen es einbringt für das gegebene Query.

Am Schluss werden die Queries gemäss dem Goldstandard verifiziert.

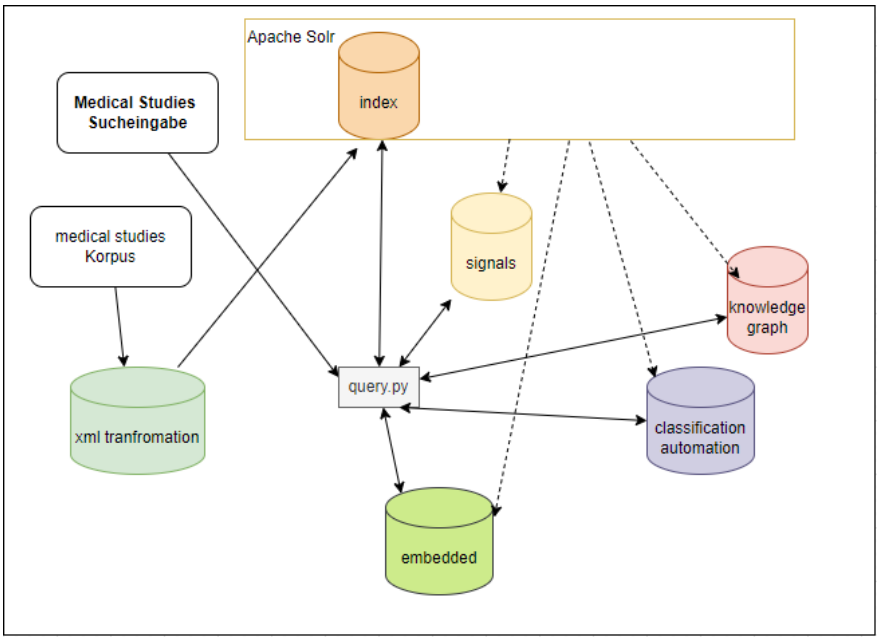


Abbildung 7: Architektur Medical Study Search

3 Knowledge Graph

Wie von (Joshi, 2022) beschrieben wird kann ein Knowledge Graph als eine Menge von Knoten und Kanten definieren, wie in der Abbildung ersichtlich:

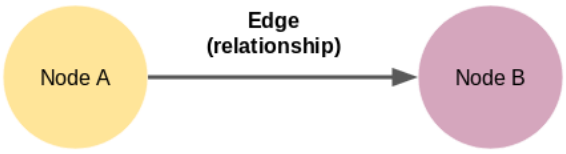


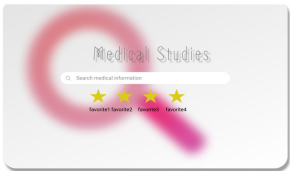
Abbildung 8: Ein einfacher Knowledge Graph (Joshi, 2022)

(Trey Greiniger, Doug Turnbull, Max Irwin, 2022) haben in dem Buch «AI Powered Search» folgendes definiert zu dem Knowledge Graph:

Es gibt drei Möglichkeiten mit einem Knowledge Graph zu arbeiten.:

1. Erstellen von eigenen Knowledge Graph von Grund auf mit Hilfe einer Graph Datenbank (Neo4j, ArangoDB, etc.)
2. Einbinden eines bereits vorhandenen Knowledge Graph (ConceptNet, DBPedia, etc.)
3. Automatisches Generieren eines Knowledge Graph aus eigenen Daten, dabei werden die Inhalte direkt benutzt, um das Wissen zu extrahieren.

Jeder Ansatz hat seine Stärken und Schwächen, und schliessen sich nicht unbedingt gegenseitig aus. (Trey Greiniger, Doug Turnbull, Max Irwin, 2022) zeigen den Graphen wie folgt:



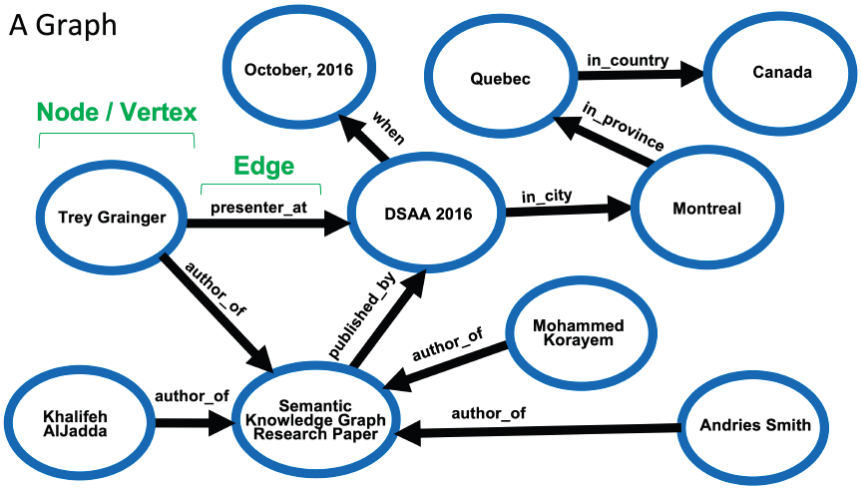


Abbildung 9:Knowledge Graph

In dieser Arbeit werde ich nun so ein Knowledge Graph erstellen nach Punkt 3, also aus eigenen Daten, um Informationen aus den Medical-Studies zu gewinnen. Dabei werde ich mit Python arbeiten, um einen Knowledge Graph mit der Anwendung der spaCy-Bibliothek.

In meinem Beispiel können wir ein Query setzten, welches voraussetzt sicher einem Geschlecht zugeteilt zu sein, also setzten wir das «facet.query=gender» und dazu möchte ich ein Knowledge Graph setzten, dabei sollte in den Dokumenten das Wort «Ovarian Cancer» im «brief_title» vorkommen.

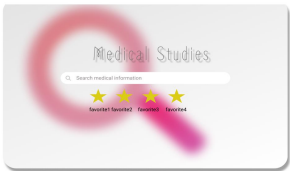
Ich werden nun das Query in meinen Code integrieren. Durch meine Recherchen bin ich auf eine gut erklärte Quelle von (Joshi, 2022) gestossen. Der Code ist sehr gut aufgeschlüsselt und lieft das Resultat wie ich mir das für meine Daten vorstelle.

Ich passe den Code etwas an, weil ich die Daten direkt von Solr beziehen möchte.

Folgende Libraries werde ich hinzuziehen und dabei wird das Spacy Model «en_core_web_sm» verwendet

```
import json
from urllib.request import urlopen

import pandas as pd
import spacy
from spacy.matcher import Matcher
import networkx as nx
import matplotlib.pyplot as plt
from tqdm import tqdm
nlp = spacy.load("en_core_web_sm")
# get query match title cancer
connection =
urlopen("http://localhost:8983/solr/med_studies/select?facet.query=gender&facet=tru
e&indent=true&q.op=OR&q=brief_title%3A%22*breast*%22cancer*%22&wt=json")
response = json.load(connection)
# get query match brief title diabetes
connection2 =
urlopen("http://localhost:8983/solr/med_studies/select?facet.query=gender&facet=tru
e&indent=true&q.op=OR&q=brief_title%3A%22diabetes%22&wt=json")
response2 = json.load(connection2)
```



Ich werde hier einige Sätze aus den gefundenen Dokumenten als Entität einsetzen, diese werden danach mit der Liste der Sätze aller gefundenen Sätze dieses Queries verglichen.

Um zur Beziehung zwischen den Texten und der Entität zu gelangen, muss noch ein Pattern festgelegt werden:

```
def get_relation(sent):
    doc = nlp(sent)
    # Matcher class object
    matcher = Matcher(nlp.vocab)
    #define the pattern
    pattern = [{'DEP':'ROOT'},
               {'DEP':'prep','OP':"?"},
               {'DEP':'agent','OP':"?"},
               {'POS':'ADJ','OP':"?"}]
    matcher.add("matching_1", [pattern])

    matches = matcher(doc)
    k = len(matches) - 1

    span = doc[matches[k][1]:matches[k][2]]

    return(span.text)
```

ROOT: Root

OP: Operator oder Quantifizierer, bestimmt wie oft ein Token-Muster übereinstimmen soll.

DEP: Nicht klassifizierte Abhängigkeit

ADJ: Adjektiv

prep: Präpositionsmodifikator

agent: Agent

Zum Schluss wird alles visuell in einem Graphen dargestellt. Hier ist der Knowledge Graph von meinem Knowledge Query «Cancer»:

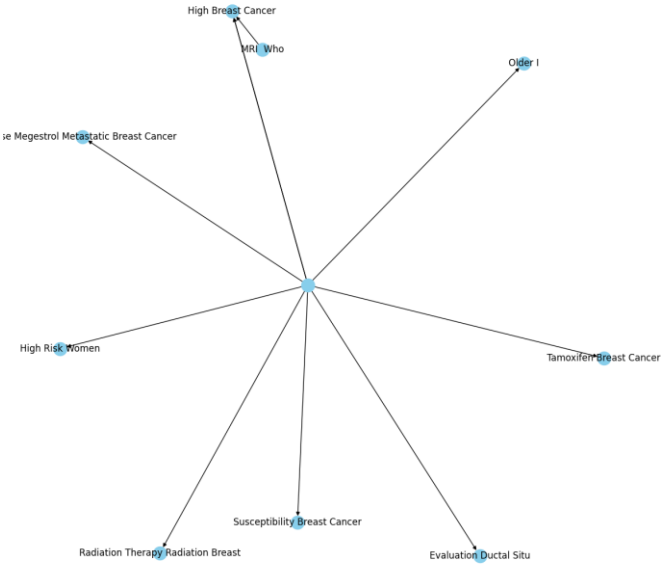


Abbildung 10: Knowledge Graph Breast Cancer Query

Ganz gut ersichtlich hat «Cancer» verschiedene Beziehungen zu anderen Eingaben. Ich habe mir das schon gedacht. Hier sieht man wie gross die Möglichkeit ist. Es sind Beziehungen zwischen Krankheiten und deren gemachten Studien auffindbar.. Das Potenzial hier ist riesig und spannend.

Nun habe ich das gleich noch für Diabetes gemacht, wollen wir doch sehen, wie der Graph sich hier darstellt:

Abbildung 11: Knowledge Graph Diabetes

Die Beziehungen der Wörter machen auf jeden Fall Sinn

Ein Knowledge Graph kann für verschiedene Krankheiten erstellt werden. Somit könnten Kategorien nach Klassen erstellt werden. Mittels Score-Index kann noch die Verwandtschaft im Knowledge Graph favorisiert werden.

Ich werden nun die Klassen erstellen, das heisst, das Wort «Cancer» ist der Klassennamen, dabei werden aber auch die zugehörigen Namen des Knowledge Graph berücksichtig.

Es könnten auch noch unter Kategorien erstellt werden, da die Nutzer Ärzte wie auch die Patienten nach spezifischen Klassen suchen möchten.

Zum Beispiel verschiedene Krebs-Erkrankungskategorien:

- «Lung-Cancer»
- «Prostate-Cancer»
- «Ovarian-Cancer»
- «Breast-Cancer»
- Etc.

Innerhalb dieser Klassen können auch verschiedene gerankte Dokumente mit Signalen geboostet werden. Dazu mehr im Kapitel 4 «Signal Boosting Modell».

3.1 Schlussfolgerung des Knowledge Graph

Der Knowledge Graph kann in verschiedenen Anwendungen seinen Platz finden. Die Verwandtschaft der Terme zu evaluieren ist sehr aufschlussreich für die Erweiterung des Rankings, welches im nächsten Abschnitt folgt. In meinem Beispiel wurde nur einem Dokument mit mehreren verglichen und nicht alle miteinander. Sonst wäre der Graph zu unübersichtlich geworden. Hierbei können Hierarchien dargestellt werden und somit auch Unterkategorien erkannt werden. Dabei könnten die Graphen auch auf einer Seite angezeigt werden, damit die Ärzte auch verwandte Krankheiten in Betracht ziehen können.

4 Signal Boosting Model

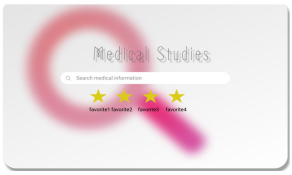
4.1 Grundlagen zur Theorie

Bei Signal Boosting werden alle Dokumente betrachtet, die jemals für eine bestimmte Abfrage angeklickt wurden. Dabei wird dann einen Boost für die Gesamtzahl der bisherigen Klicks auf dieses Dokument abgefragt.

Leider ist es anfällig für Datenverzerrungen und sogar Manipulationen. Es können einige Techniken zur Entfernung von Störungen in den Signalen erarbeitet werden, um die Qualität der Signalverstärkungsmodelle zu maximieren und die Möglichkeit unerwünschter Verzerrungen zu verringern. (Trey Greiniger, Doug Turnbull, Max Irwin, 2022)

4.2 Signal Boosting gewisser Dokumente zur Indexierungszeit

Boosten von populären Abfragen für Dokumente zur Indizierungszeit.



Dies geschieht durch Hinzufügen beliebter Abfragen zu einem Feld in jedem Dokument, zusammen mit ihrem Boost-Wert. Zur Abfragezeit suchen wir dann einfach im neuen Feld, und wenn das Feld den Begriff aus unserer Abfrage enthält, wird es automatisch auf der Grundlage des für den Begriff indizierten Boost-Wertes, der für den Begriff indexiert wurde, geboostet.

Bei der Implementierung des Boosting zur Indexierungszeit wird die Signalaggregationen genutzt, um Paare von Dokumenten und Boost-Gewichten für jede Abfrage zu erzeugen. Sobald diese Signal Boosts generiert wurden, muss nur noch ein zusätzlicher Schritt in den Arbeitsablauf eingebaut werden, das Aktualisieren der Info.

In einer Sammlung wird jedem Dokument ein Feld zugeteilt, welches die Begriffe mit gleichen oder ähnlichen Dokumenten enthält, die geboostet werden soll, zusammen mit der zugehörigen numerischen Boost-Gewichtung.

(Trey Greiniger, Doug Turnbull, Max Irwin, 2022)

4.3 Praktische Umsetzung

Ich habe mich in der Medical Studies Suche für Boosting gewisser Dokumente zur Indexierungszeit entschieden, weil es so gut eingebaut werden kann.

Dabei wurden folgendes hinzugefügt:

- Dokumenten ID
- Klicks
- Gewichtung
- Target ("condition")

Die Daten werden aus Solr geladen. Danach werden den Daten Random an einem Zeitpunkt und an einem Gewicht zugeteilt.

```
connection_bc = urlopen(

"http://localhost:8983/solr/med_studies/select?defType=lucene&facet.contains
=cancer&facet.field=official_title&facet.sort=count&facet=true&indent=true&q
.op=OR&q=brief_title%3Abreast%20cancer&wt=json")
relevantDocument_bc = json.load(connection_bc)


def breast_cancer():
    breast_cancer_array = []

    for i in relevantDocument_bc['response']['docs']:
        query = i['brief_title']
        condition = i['condition']
        target = i['id']
        weight = random.randint(1, 4)
        timestamp = datetime.now()
    breast_cancer_array.append(
        str('["query":"{}" , "timestamp":"{}", "condition":"{}" ,
"type":"click" , "weight":"{}", "target":"{}"]').format(
            query,
            timestamp,
            condition,
```

```
                    weight, target))
        return relevantDocument_bc
with open(f'signal_ovarian_cancer.json', 'w') as f:
    json.dump(ovarian_cancer, f)
```

5 Ranking Model

Ranking-Modelle arbeiten in der Regel mit der Vorhersage einer Relevanzbewertung s = f(x) für jede Eingabe x = (q, d), wobei q eine Anfrage und d ein Dokument ist. Sobald wir die Relevanz jedes Dokuments kennen, können wir die Dokumente nach diesen Werten sortieren (d.h. einordnen). Das Scoring-Modell kann mit verschiedenen Ansätzen implementiert werden. (Casalegno, 2022)

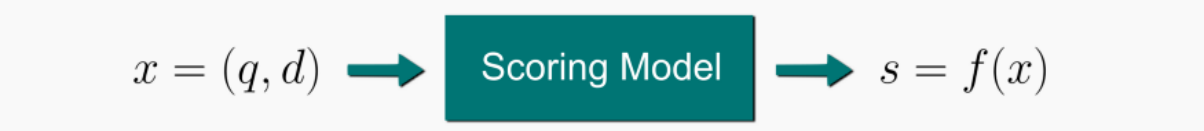


Abbildung 12:Ranking Modell (Casalegno, 2022)

5.1 Vektorraummodelle:

Berechnung einer Vektoreinbettung (z. B. mit Tf-Idf oder BERT) für jede Anfrage und jedes Dokument und anschliessende Berechnung der Relevanzbewertung f(x) = f(q, d) als Kosinusähnlichkeit zwischen den Vektoreinbettungen von q und d. (Casalegno, 2022)

5.2 Learning to Rank:

LTR[2]-Systeme wandeln unsere Trainingsdaten in Modelle um, in ein sogenanntes Relevanz-Ranking. Mit dieser Art von System können die zugrunde liegenden Muster in den Trainingsdaten gefunden werden.

(Trey Greiniger, Doug Turnbull, Max Irwin, 2022)

(Casalegno, 2022) hat folgendes geschrieben: Das Scoring-Modell ist ein Modell des maschinellen Lernens, das während einer Trainingsphase, in der eine Art von Ranking-Verlust minimiert wird, lernt, einen Score für eine Eingabe x = (q, d) vorherzusagen.

(Casalegno, 2022)

6 Automatische Klassifikation

Die Automatische Klassifikation Ist die Suche nach Mustern anhand eines Klassifikationsmerkmal. Dies kann zum Beispiel die Modellierung einer Produktaffinität sein. Durch die antrainierten Muster lassen sich beispielsweise Produktaffinitäten vorhersagen. (Wuttke, 2021)

[2] LTR: learning to rank

Es liegt eine Klassifizierung vor, wenn das vorherzusagende Merkmal Kategorien von Werten enthält.

Jede dieser Kategorien wird als eine Klasse betrachtet, in die der vorhergesagte Wert fällt.

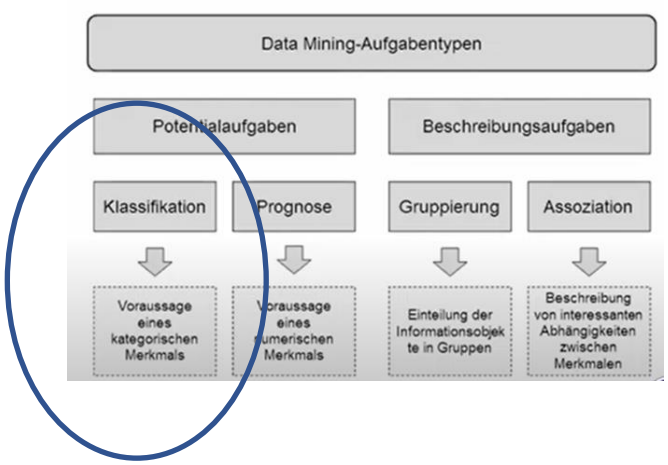


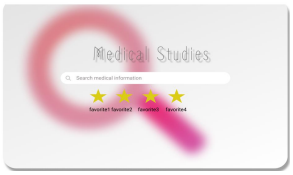
Abbildung 13: Datamining Aufgabentypen (Wuttke, 2021)

Zu den Klassifizierungsalgorithmen gehören:

- Naive Bayes
- Logistische Regression
- K-Nächste Nachbarn
- (Kernel) SVM
- Entscheidungsbaum
- Ensemble-Lernen

6.1 Klassifizierung in der Praxis mit dem K-Nächsten Nachbarn

Zuerst werden die Klassifizierungen herausgepickt, bei der Schema-Analyse der «Condition» sind diese schon sehr gut ersichtlich:



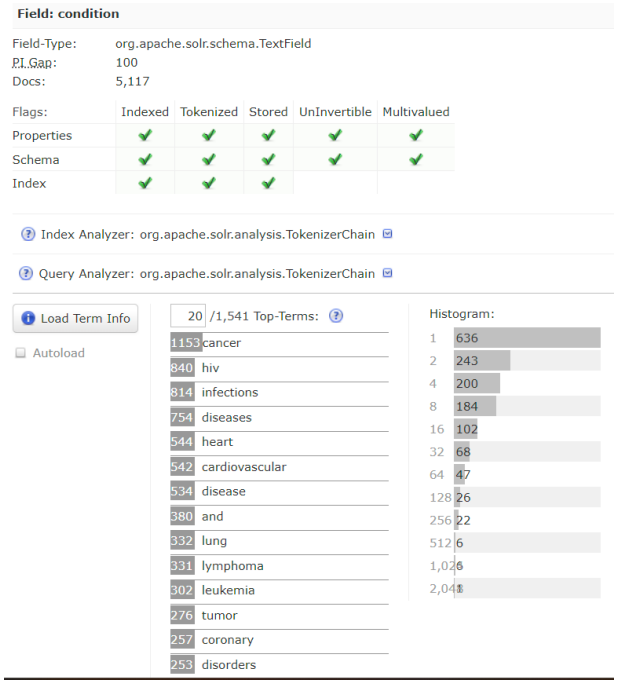


Abbildung 14:Analyse "condition" für Klassifizierungen

Die Kategorien werden per Request in Solr Facets abgerufen und in eine Liste eingefügt. Das Datensätze können jetzt trainiert werden, aus den Dokumenten, die einer Kategorie zugeteilt wurden. Dabei könnte der Abgleich der Dokumente gleicher Kategorie gemacht werden und deren Muster trainieren. Die Kategorien müssten immer gegenüber der Testdaten gegenübergestellt werden, damit die Fehlerrate evaluiert werden kann. Bei einer Abfrage von einem Nutzer würden die Dokumente, die gerankt wurden, mit den trainierten Daten abglichen werden und einem Klassifizierungsmuster zugeteilt werden. Dabei werden nur die wichtigen Dokumente dieser Kategorie auftauchen. Nächster Schritt wäre dann noch die Signal Boosting zu beachten.

Dabei könnten vortrainierte Datensets gebraucht werden wie zum Beispiel Clinical BERT von «Hugging Face»:

```
def getClassification(query):
    model =
AutoModelForSequenceClassification.from_pretrained("tarasophia/Bio_Clinical
BERT_medical")
    tokenizer =
AutoTokenizer.from_pretrained("tarasophia/Bio_ClinicalBERT")
    inputs = tokenizer(query, return_tensors="pt")
    with torch.no_grad():
        logits = model(**inputs).logits
    predicted_class_id = logits.argmax().item()
    model.config.id2label[predicted_class_id]
    predicted_token_class_ids = logits.argmax(-1)
    predicted_tokens_classes = [model.config.id2label[t.item()] for t in
predicted_token_class_ids[0]]
    return predicted_tokens_classes
```

Dabei wird das Query einer vorhergesagten Token Klasse zugeteilt.

6.2 Fazit Ranking Modelle

Bei der Automatischen Klassifizierung hätte das Suchwort oder die Sucheingabe die Klassifizierung vorausgesagt. Dabei kann man die Daten trainieren und mit einen Testdatenset verifizieren wie die Fehlerrate liegt. Leider hat die Zeit dazu nicht mehr gereicht. Aber das würde die Advance Search, der Kategorien bereichern und das Ganze Ranking enorm verbessern.

7 Embedded Search

Gemäss (Bronznick, 2022) sind «Embedded Words /Search» mathematische Strukturen, die eine Sammlung von Wörtern darstellen. Die Strukturen sollten den semantischen und syntaktischen Kontext eines Wortes erfassen. Dabei sollte jede Beziehung zu anderen Worten via Vektor dargestellt werden, so dass die Beziehung binär kalkuliert werden kann. Dokumentensammlungen werden auch als Vektoren abgebildet in einem sogenannten Vektorraum. Sammlung von Dokumenten werden ebenfalls als Vektoren dargestellt, dabei wird alles in einem Vektorraum dargestellt. Die Länge wie auch die Richtung des Vektors eines Wortes wird ähnlich repräsentiert wie ein Wort mit einer ähnlichen Bedeutung. Das heisst, die Vektorräume sind sehr nahe beieinander.

Mit einem Trainierten System wie Bert sieht es wie folgt aus zum Beispiel von (Sharma, 2020)

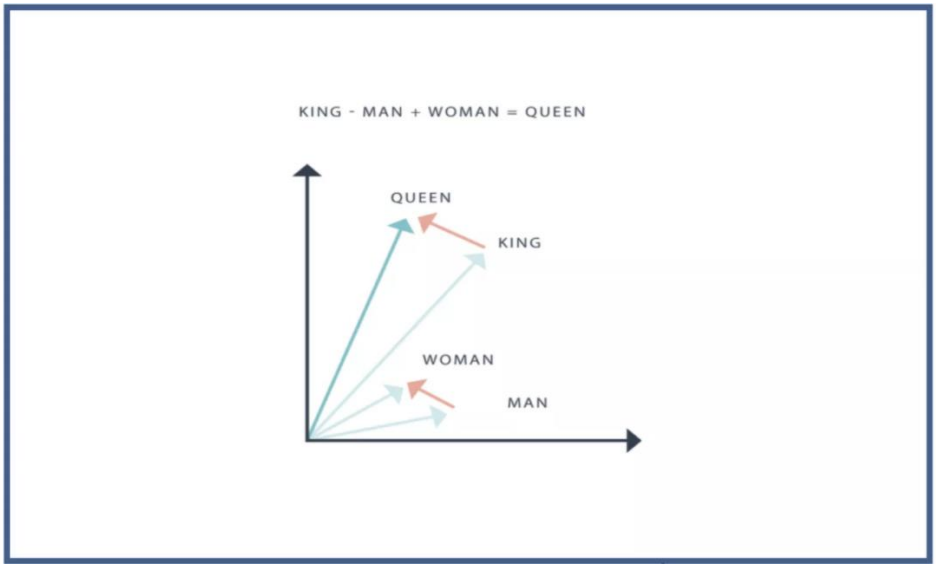
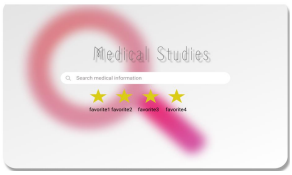


Abbildung 15: Beispiel Embedded Wörter in Vektoren dargestellt

7.1 Mögliche Umsetzung Embedded Search

Leider hat die Zeit nicht mehr ausreichend gereicht. Dabei könnten ein Vortrainiertes Modell von «Hugging Face - emilyalsentzer/Bio_ClinicalBERT» verwendet werden.




```
import nlu

def embedding(query):
    embeddings_df =
nlu.load('en.embed_sentence.bert_base_uncased').predict(query,
output_level='sentence')
    return embeddings_df
embedding("breast cancer")
```

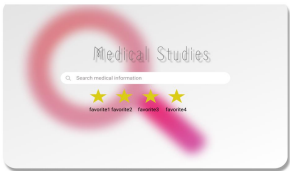
Dabei werden die Wörter schon erkannt und dem Richtigen Vektorraum zugeteilt. Das ist sehr nützlich wenn ein Suchvorschläge ergänzt werden. Eine sehr angenehme Funktion für die Nutzer.

8 Recap /Fazit

Ziel dieser Arbeit war es, die Realisierung eines Suchsystems für einen selbst ausgewählten Textkorpus. Ich hatte einen Korpus mit medizinischen Studien ausgewählt und bin mit meiner Wahl sehr zufrieden, auch wenn die Datenaufbereitung nicht perfekt war und auch sehr aufwändig, habe ich dabei so einiges gelernt. Solr zum Laufen zu bringen war am Anfang bei mir etwas aufwändiger, jedoch nachdem es einmal lief, hatte ich wirklich Spass damit. Die Indexierung brauchte am meisten Zeit , weil die Daten in einer so unbrauchbaren Verfassung waren, dass ich 5000 Dokumente anpassen musste. Klar habe ich dazu eine Automatisierung programmiert, jedoch musste diese auch zuerst funktionieren und programmiert werden. Wurden die Dokumente einmal transformiert konnte ich die Indexierung im Solr vornehmen und die Daten laden.

Ich musste einmal den Goldstandard definieren, dieser war zwar optimal erfüllt, jedoch könnte noch viel mehr vorgenommen werden. Aktuell liegt ein Overfitting für den Goldstandard vor. Dies ist daran zu sehen, dass der Index nahezu perfekt die gewünschten Dokumente zuoberst auflistet, aber trotz der gewählten Vorgehensweise noch nicht optimal generalisiert. Das Signal Boosting habe ich zwar gemacht, bin mir aber nicht sicher, ob es in diesem medizinischen Fall sinnvoll ist, weil ja Krankheiten nicht wirklich generalisiert werden können oder wie oft eine Krankheit vorkommt, oder ob genau diese Studie die Richtige ist, nur weil viele auf diese klicken. Auch könnte ein Signalspam von falsch generierten Klicks die Signale verändern. Bei einer Suche wird es dem Signal angepasst und nicht dem Query. Zusätzlich könnte dem Nutzer ähnliche Studien angezeigt werden, die zu ähnlichen eingegebenen Queries von Nutzern zu Klicks geführt haben. Es könnte auch ein Timetracker genutzt werden, um zu sehen, wie lang ein Dokument nach dem Klick geöffnet war. Es gäbe noch viel spannendes, um die ganze Suche zu optimieren.

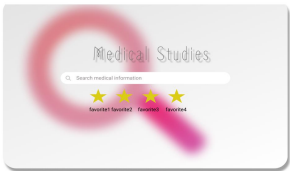
Ich habe in dieser Arbeit einen guten Eindruck bekommen, was es dazu braucht, um an einen idealen Suchalgorithmus zu gelangen. Es braucht verschiedene Schritte, und einiges auszuprobieren. Wichtig dabei ist es zu beachten, den Fokus auf die Nutzer nicht zu verlieren, . Durch die verschiedenen Möglichkeiten vergisst man schnell, ob diese Implementierung wirklich von Nutzen ist für den Nutzer, z.B. Signal Boosting in dieser Suche, fand ich nicht so optimal. Es handelt sich hier um Studien,



zum Teil Einzelfälle. Eine Studie könnte sehr wichtig sein für einen Arzt, weil sie jedoch nie geklickt wurde, erscheint diese nicht in den Top 10. Das Ganze Signal Boosting sollte also mit Vorsicht genutzt werden

Wie im Kapitel 5.25.2 beschrieben könnte man dem entgegenwirken, so dass das Problem des optimalen Rankings mit Hilfe eines Klassifizierungsalgorithmus aus dem Bereich des maschinellen Lernens löst.

In dieser Arbeit hat die Datenaufbereitung sehr viel Zeit in Anspruch genommen, so dass für die Algorithmen und die wichtigen Ranking Themen gar nicht mehr so viel Zeit geblieben ist. Das fand ich etwas schade. Data-Cleansing ist definitiv nicht meine Leidenschaft, das ist aber doch auch schon eine Erkenntnis. Ich hätte jetzt wirklich Lust das Ganze zu implementieren und produktiv laufen zu lassen. Dafür ist vielleicht nach dem Studium Zeit dazu.



Literaturverzeichnis

Amati, G. (2009). *Encyclopedia of Database Systems.* New York, NY: Springer New York, NY.

Bronznick, A. (2022). *Using Word Embedding method to improve information.* The Open University.

Casalegno, F. (6. 12 2022). *learning to rank*. Von towardsdatascience: https://towardsdatascience.com/learning-to-rank-a-complete-guide-to-ranking-using-machine-learning-4c9688d370d4 abgerufen

ClinicalTrials.gov . (Mai 2021). Von Downloading Content for Analysis: https://clinicaltrials.gov/AllPublicXML.zip abgerufen

Joshi, P. (14. June 2022). *analyticsvidhya*. Von how-to-build-knowledge-graph-using-spacy: https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/ abgerufen

Shafi, A. (1 2014). *precision-and-recall*. Von towardsdatascience : https://towardsdatascience.com/precision-and-recall-88a3776c8007 abgerufen

Sharma, N. (26. 11 2020). *how-to-use-bert-sentence-embedding-for-clustering-text*. Von https://techblog.assignar.com: https://techblog.assignar.com/how-to-use-bert-sentence-embedding-for-clustering-text/ abgerufen

Trey Greiniger, Doug Turnbull, Max Irwin. (2022). AI Powered Search. In *AI Powered Search.* Manning.

Tutorialpoint. (2022). Von Apache Solr: https://www.tutorialspoint.com/apache_solr/apache_solr_overview.htm abgerufen

Wuttke, L. (2021). *https://datasolut.com*. Von https://datasolut.com/was-ist-data-mining/ abgerufen