Universität Leipzig

Modul 10-201-2316

„Information Retrieval“

Sommersemester 2019

Information Retrieval: Simpsons Search

Dozent: Junior-Prof. Dr. Martin Potthast

Autor: Sonja Heinze
sh27deny@studserv.uni-leipzig.de
Matrikel-Nr.: 3735039

Autor: Carlos König
cifo@studserv.uni-Leipzig.de
Matrikel-Nr.: 3733455

Autor: Paula Müller
pm39funo@studserv.uni-leipzig.de
Matrikel-Nr.: 3709196

Autor: Esther Prause
ep59tono@studserv.uni-leipzig.de
Matrikel-Nr.: 3717865

Autor: Robin Seidel
rs92vegi@studserv.uni-leipzig.de
Matrikel-Nr.: 3735157

Inhaltsverzeichnis

1. Einleitung

Für den Bau einer Suchmaschine im Rahmen des Moduls 10-201-2316 „Information Retrieval“ entschied sich unsere Gruppe für das Erstellen einer Suchmaschine über einen Datensatz, der sich mit der berühmten Zeichentrickserie „The Simpsons“ befasst. Diese Entscheidung wurde vorrangig von unserem persönlichen Interesse motiviert, da unsere Generation mit der gelben Familie aufgewachsen ist, allerdings ist auch die kulturelle Relevanz und Präsenz der Serie bedeutend genug, dass die Suchmaschine auch ein breiteres Publikum ansprechen könnte. Des Weiteren zeichnen sich die “Simpsons” sowohl durch ein hohes Maß an Intertextualität und -medialität aus als auch durch, häufig sarkastische oder satirische, Bezugnahme auf gesellschaftspolitische Themen. Für Forschungsarbeiten in den Kultur- und Medienwissenschaften, die solche Bezüge untersuchen, kann es eine enorme Zeitersparnis bedeuten, gezielt nach relevanten Stellen (Szenen) suchen zu können.
Der Datensatz stammt von der Website kaggle.com[1] und basiert auf den Skripten der einzelnen Folgen, speziell besteht er aus CSV-Dateien, die unter das aus der Vorlesung bekannte Konzept „1 File – Many Units“ fallen. Die CSV-Dateien werden also in diesem Falle in Dokumentenunits aufgeteilt. Dabei werden von uns vor allem die Dialoge in den einzelnen Szenen als Grundlage des Suchprozesses verwendet, sodass sowohl eine interpretative Suche, aber dank der Datenstruktur auch datenbankähnliche Abfragen möglich sind. Letztere nehmen in diesem Projekt jedoch nur einen kleinen Teil der Anfragen ein.
Als Grundlage für unsere Suchmaschine nutzen wir die in C# geschriebene Open-Source-Software Apache Lucene.Net, in der neuesten Version 4.8 von 2015.[2] Unsere Implementierung verwendet ebenfalls C#.

2. Beschreibung des Datensatzes

Der ausgewählte Datensatz, der vor allem die Skripte der 600 Folgen in den Staffeln 1 bis 28 (bis Folge 4) der Fernsehserie „The Simpsons“ umfasst, besteht aus 4 CSV Dateien[3]. Die

1https://www.kaggle.com/wcukierski/the-simpsons-by-the-data/, abgerufen am 07.05.2019. Zum Zeitpunkt der Erstellung dieses Berichts scheint der Datensatz gelöscht worden zu sein. Eine v.a. statistisch orientierte Aufbereitung der Daten befindet sich jedoch weiterhin unter: https://www.kaggle.com/gpreda/simpsons-world-revealed, abgerufen am 13.08.2019.

2 http://lucenenet.apache.org/download/version-4.html, abgerufen am 14.08.2019.

3 Die Originaldateien sind im Ordner „Data“ im github unseres Projektes zu finden.

tabellenartige Struktur unserer Daten ermöglicht prinzipiell nicht nur interpretative Suche, sondern auch reine Datenbankabfragen. Die Details der einzelnen Dateien sind in folgender Tabelle aufgeführt:

Dateiname	**Zeilen**	**Spalten (Anzahl)**
simpsons_script_lines.csv	157.462	id, episode_id, number, raw_text, timestamp_in_ms, speaking_line, character_id, location_id, raw_character_text, war_location_text, spoken_words, normalized_text, word_count (13)
simpsons_characters.csv	6.177	id, name, normalized_name, gender (3)
simpsons_episodes.csv	600	id, title, original_air_date, production_code, season, number_in_season, number_in_series, us_viewers_in_millions, views, imdb_rating, imdb_votes, image_url, video_url (13)
simpsons_locations.csv	3.144	id, name, normalized_name (4)

Tab. 1: Datensatz

Der klare Fokus für dieses Projekt liegt auf der Datei script_lines, die den gesprochenen Text eines jeden Charakters pro Redeanteil in den einzelnen Folgen umfasst und auf der sich somit inhaltlich nach bestimmten Topics[4] suchen lässt. In dieser Datei sind allerdings auch Regieanweisungen verzeichnet, in diesem Fall ist der Wert der Spalte speaking_line „FALSE“, sodass über diesen Wert nicht in Regieanweisungen, sondern in tatsächlich gesprochenen Worten gesucht werden kann. Getan wird dies in der Spalte „normalized_text“. Zur Präsentation werden über die IDs der Charaktere, Orte und Folgen jedoch auch deren Werte erfasst. Als Dokumente des Datensatzes hätten einzelne Redeanteile, ganze Folgen oder Szenen deklariert werden können. Dabei erschien uns Letzteres als das beste Maß, sodass weder nur ein einzelner Redebeitrag noch eine Folge in ganzer Länge präsentiert wird. Die Unterteilung in Szenen erfolgt dabei allerdings nicht durch den bereits erwähnten „FALSE“-Wert der Spalte speaking_line, da Szenen so womöglich vorzeitig beendet würden. Beispielsweise könnte eine Regieanweisung heißen, dass ein Charakter lacht, was jedoch die

4 Siehe Abschnitt 4, Evaluation.

Szene nicht unterbrechen würde. Als Lösung dieses Problems wird der Wert in der Spalte “character_id” zu Rate gezogen, denn nur sobald dieser leer ist, findet gerade keine Handlung, sei es ein Sprechakt oder besondere Gestik und Mimik, statt (und damit ist die Szene abgeschlossen). Nach dieser Unterteilung erhalten wir insgesamt 15.618 Szenen als Dokumente.

3. Architektur der Suchmaschine

Unsere Suchmaschine ist eine .NET Core-Webanwendung. Das Back-End ist in C# geschrieben. Zur Indexierung des Datensatzes und zur Suche wird die Suchmaschinen-Bibliothek Lucene.net in der aktuellen Version 4.8.0 verwendet. Diese ist ein Clone der Apache Lucene-Bibliothek für die Programmiersprache C#.
Zum Einlesen unseres CSV-Datensatzes wurde dabei die Bibliothek CsvHelper[5] verwendet. Weiterhin wird für die Erstellung der Evaluationsdokumente im JSON Format die Bibliothek Newtonsoft.Json[6] genutzt. Die Anbindung an das Front-End erfolgt mittels Razor Pages[7], welches ein Teil des .NET Core-Frameworks ist. Weitere genutzte Technologien sind Jquery und Javascript. Zum Deployment der Webanwendung verwenden wir Docker.

3.1 Retrieval-Modell unserer Baseline Suchmaschine

Ausgehend von dem Konzept *Retrieval Model* als Tupel ‹D,Q,ρ›, beschreiben wir im Folgenden die drei Komponenten des Retrieval-Modells, auf dem die Implementierung der Baseline unserer Suchmaschine aufbaut.

<D>: Für die Erstellung der Dokumentrepräsentationen lesen wir die Dateien *simpsons_script_lines.csv* und *simpsons_episodes.csv* mithilfe von Methoden der .NET-Library CsvHelper zeilenweise ein. Eine Zeile entspricht bei der *simpsons_script_lines.csv* einer ScriptLine und bei der *simpsons_episodes.csv* einer Episode. Zur weiteren Verarbeitung

5 https://joshclose.github.io/CsvHelper/, abgerufen am 15.08.2019.

6 https://www.newtonsoft.com/json, abgerufen am 15.08.2019.

7https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-2.2&tabs=visual-studio, abgerufen am 15.08.2019.

werden die Dateien so zu Objekten und als Listen gespeichert (siehe Datei *ConversionService.cs* im Ordner *Services*). Die Einteilung in Dokumente für die Indexierung erfolgt in *SimpleSearchBase.cs* anhand von Heuristiken. Hierbei definieren wir ein Dokument als eine Szene. Zur Unterteilung in Szenen verwenden wir die Spalte ‚character_id', der Annahme folgend, dass Szenenwechsel im Allgemeinen durch einen Ortswechsel bestimmt werden können. Im Skript bedeutet dies, dass eine Zeile mit der Beschreibung der neuen Location vorhanden ist, ohne dass gleichzeitig in der Spalte ‚character_id' ein Charakter eingetragen wird. Entsprechend lässt sich die Einteilung der ScriptLines in Szenen gut über die Abfrage nach leeren Feldern als Trennungsmarker implementieren.
Für die Baseline werden dann folgende Felder dieser Dokumente indexiert: Text, Location und Characters. In Text ist der ‚normalized_text' einer Szene als String gespeichert, in der Location der Ort der Szene und unter Characters werden alle in der Szene vorkommenden Figuren als Set gespeichert.
Im Rahmen der Indexierung unserer Dokumentenkollektion nutzen wir schließlich den StandardAnalyzer von Lucene mit seinen Default-Einstellungen. Dieser setzt sich zusammen aus dem StandardTokenizer von Lucene.Net und den Filtern: StandardFilter, LowerCaseFilter, sowie StopFilter. Der StandardFilter normalisiert die extrahierten Tokens. Der LowerCaseFilter normalisiert alle tokens zu Lower-Case-Zeichen. Der StopFilter entfernt Stoppwörter. Für den StopFilter nutzen wir die von Lucene bereitgestellte Stoppwort-Liste für die englische Sprache. Außerdem wird ein Stemming über den extrahierten Tokens durchgeführt.

<Q>: Für die eingegebenen Queries nutzen wir denselben StandardAnalyzer wie für die Dokumente und zusätzlich den „MultiFieldQueryParser" zur Verarbeitung. Dieser erweitert den einfachen QueryParser von Lucene insoweit, dass in mehreren Feldern eines Dokuments gesucht werden kann. Dies ist für unserer Suche erforderlich, da die Felder Text, Location und Characters relevante Suchfelder für unsere Queries sind. Ansonsten unterscheiden sich die Parser nicht.
Somit sind durch den Parser alle gängigen Verfahren in ihrer Default-Einstellung verwendbar, dazu gehören u.a. die logischen Operatoren (*Homer AND Marge*), direkte Suchen in spezifischen Feldern (z.B. *location: school*), Wildcard-Suchen (z.B. *te?t* oder *wo*o*) und reguläre Ausdrücke. Eine vollständige Auflistung der Möglichkeiten liefert die

Dokumentation[8].
Die Default-Einstellung des MultiFieldQueryParser behandelt darüber hinaus zwei einfach eingegebene Worte, d.h. ohne Zusätze in der Suchanfrage, als „OR“ Anfrage und sucht beide Begriffe jeweils in allen Feldern. In den Results werden also auch Dokumente ausgegeben, in denen nur einer der beiden Begriffe vorkommt.

<ρ>: Um die Scores zu erhalten, übernehmen wir für die Relevanzfunktion unverändert die Defaulteinstellungen, die Lucene.Net nutzt, wenn nicht aktiv andere Methoden zugewiesen werden. Damit liegt unserem Modell eine Kombination aus dem Boolean-Modell und dem Vektor-Space-Modell zugrunde. Allgemein findet Lucene zunächst die Dokumente, die gescored werden müssen, anhand von Boolean-Logik entsprechend der Query Spezifikationen. Dann werden diese Ergebnisse dem implementierten Vektor-Space-Modell entsprechend gescored und gerankt. Die Dokumente und Queries werden als gewichtete Vektoren in einem multi-dimensionalen Raum abgebildet. Als Gewichte werden standardmäßig tf-idf Values genutzt. Für das Scoring wird dann im Wesentlichen mit der Kosinusähnlichkeit gerechnet. Lucene verfeinert das Scoring an dieser Stelle noch um einige Faktoren. So wird standardmäßig ein Score-Faktor hinzugenommen, der darauf basiert, wie viele Begriffe der Query im Dokument gefunden wurden, so dass Dokumente, in denen mehrere Suchbegriffe vorkommen, einen höheren Score erhalten sollten. Die Dokumentation beschreibt die einzelnen Faktoren an dieser Stelle noch detaillierter[9].

3.2 Relevance Judgement

Unsere Suchmaschine ermöglicht direktes Relevanz-Feedback. Mittels zwei Buttons neben dem Textfeld (‘+’ und ‘-’) können Suchergebnisse mit der Wertung ‘relevant’ oder ‘nicht relevant’ belegt werden. Die Anwendung befindet sich somit permanent im Evaluationsmodus, welcher aber natürlich ignoriert werden kann.
Für das Logging des Relevance Judgements, das User über das Anklicken der Buttons abgeben können, steht eine Log-Datei zur Verfügung, in der alle Suchanfragen mit der

8 https://lucenenetdocs.azurewebsites.net/api/Lucene.Net.QueryParser/Lucene.Net.QueryParsers.Classic.html, abgerufen am 15.08.2019.

9 https://lucenenetdocs.azurewebsites.net/api/Lucene.Net/Lucene.Net.Search.Similarities.TFIDFSimilarity.html, abgerufen am 15.08.2019.

jeweiligen Anzahl der Ergebnisse und jede einzelne Bewertung gespeichert werden. User werden über die IP-Adresse identifiziert, welche ebenfalls gespeichert wird. Zusätzlich löst ein Klick auf einen Button das Anlegen einer JSON-File im Ordner ‘Evaluation Files’ aus. Der Prozess ist gezielt auf unsere Topics zugeschnitten.
Im aktuellen Zustand der Suchmaschine haben die geloggten Bewertungen keinen Einfluss auf eine Query, die mehrmals ausgeführt wird. Sie dienen vor allem der Bewertung des implementierten Status Quo, um die Suchmaschine evaluieren zu können. Das Log für Optimierungen zu nutzen ist jedoch bereits jetzt grundsätzlich möglich und wäre ein logischer nächster Schritt für Erweiterungen.

3. 3 Refinement der Baseline

Da wir unsere Suchmaschine speziell darauf ausrichten wollten, nach Szenen zu bestimmten Themen (Topics) zu suchen, haben wir zur Optimierung der Baseline zusätzlich eine Form von QueryExpansion implementiert. Dafür haben wir zu jedem Topic in der Datei dictionaries.cs ein dictionary mit ähnlichen/verwandten Wörtern angelegt (z.B. *travelDic*, *schoolDic*). Wird ein Suchbegriff als eines unserer Topics erkannt, fügt die Methode GetSynonyms() in der Datei SynonymAnalyzer.cs die Strings aus dem entsprechenden dictionary einem neuen Suchstring hinzu (z.B. für Topic *affair*:

```
sBuilder.Add(new CharsRef("affair"), new CharsRef(dictionaries.affairDic[key]), true).
```

Gesucht wird dann also nach allen Begriffen.
Die Wörter für die dictionaries haben wir teils von bereits existierenden Wordclouds und thematischen Wortlisten online übernommen, teils manuell erstellt. Sie enthalten Synonyme, Konnotationen und ähnliche Begriffe aus den Wortfeldern der Topics, im Sinne eines Thesaurus. In einer nicht-fingierten, realen Anwendersituation können solche Wortsammlungen beispielsweise aus früherem Query-Logging zustande gekommen sein (“wurde häufig zusammen gesucht mit”) oder aus der Analyse der Dokumente im Document Store in Kombination mit Relevanzfeedback (“Dokumente, die Nutzer für Query A relevant fanden, enthielten auch folgende Begriffe”).

4. Evaluation

Für die Evaluation unserer Suchmaschine haben wir 45[10] Topics erstellt, die sich grob in zwei Anfragetypen unterteilen lassen: Zum einen Topics, von denen wir erwartet haben, dass einfache Datenbankabfragen ohne Optimierungen bereits gute Ergebnisse liefern werden (wie die Suche nach bestimmten Charakteren, *Homer*, oder die Kombination aus Charakter und Location, *Homer tavern*); zum anderen thematisch angelegte Topics mit teilweise abstrakten Suchbegriffen (*environment*, *romance*, *education*). 9 der Topics fallen eher unter datenbankorientierte, die restlichen 36 unter interpretative Suche. Tab. 1 zeigt exemplarische Beispiele.

Das Feld *title* entspricht der eingegebenen Query. Nachdem die Baseline der Suchmaschine lauffähig war, wurden in einem ersten Evaluationsdurchgang die Suchergebnisse für alle Topics binär mit *relevant* oder *nicht relevant* bewertet. Als Bewertungsmaß haben wir uns für *Precision@k* mit $k = 10$ entschieden[11]. Hier erreicht unsere Baseline eine MeanAveragePrecision (MAP) von 0,56.

Dabei schwankt die AveragePrecision (AvgP) der einzelnen Topics enorm. Bei einigen werden Top-Werte von ≥ 0,9 erzielt. Interessant ist der Vergleich der Anfragen *Homer* und *Krusty* (der Charakter muss jeweils in der Szene vorkommen): Obwohl man erwarten könnte, dass Homer als wichtigster und redestärkster Charakter bessere Ergebnisse liefert, liegt die AvgP nur bei 0,69, Krusty als Nebencharakter aber bei 0,95. Dies erklärt sich dadurch, dass die Figur des Homer Simpson in der Serie so dominant ist, dass sein Name häufig erwähnt

10 In der Topiclist in unserem git-Repository und im Code sind 50 Topics angelegt. Dies spiegelt unser ursprüngliches Vorhaben wieder, auch statistische Anfragen an den Datensatz zu ermöglichen. Aufgrund mangelnder Relevanz für die spezifischen Problemstellungen des Information Retrieval, besonders in Bezug auf die inhärente Subjektivität von Relevanzbewertungen, haben wir diese Funktionen nicht mehr implementiert. Diese fünf Topics sind somit nicht evaluiert worden: „How many times is Bart in the detention room?“, „Who says donut most often“, „Where is bart‘s favorite location“, „top words in episode 100“, „most relevant words in season 1“. Es sei angemerkt, dass diese Queries dennoch Funde liefern; die interessierte Userin müsste dann eigenhändig die gefunden Szenen, oder bestimmte Begriffe in ihnen, durchzählen. Dies ordnen wir nicht als bewertbares Suchergebnis ein.

11 Recall-Werte messen wir nicht. Würde unsere Suchmaschine auch Episodensuche anbieten, bestünde die Möglichkeit, das online verfügbare Fan-wiki zu den Simpsons (https://simpsons.fandom.com/wiki/Category:Episode_themes, abgerufen am 15.08.2019) als Benchmark zu nutzen, so dass die dort aufgelisteten Episoden der maximal möglichen Menge an relevanten Resultaten in einer bestimmten Kategorie entsprechen. Mit der Szenensuche ergäbe ein solcher Abgleich jedoch wenig Sinn, da das Vorhandensein einer Szene, die ein bestimmtes Thema anspricht, noch lange nicht bedeutet, dass die ganze zugehörige Episode den entsprechenden Themenschwerpunkt setzt

wird, obwohl er selbst in der Szene nicht auftritt.

Eine zweite Gruppe von Topics hat überhaupt keine, oder sehr wenige (<10), Ergebnisse (*computing*, *lgbt*, *mobbing*, *doomsday*, *patriotism*). Da in der Baseline nur nach reinem Vorkommen der Querybegriffe im Datensatz gesucht wird, kann ein Topic nicht gefunden werden, wenn es in keinem Dialog direkt geäußert wird und auch sonst nirgendwo im Skript enthalten ist.

Eine dritte Gruppe liefert zwar Funde, aber mit sehr schlechter AvgP. Ursachen liegen in der Ambiguität von Queries wie *work* (Bedeutungsfeld arbeiten vs. funktionieren) oder *musical* (das Musical vs. Musik/musikalisch) und im hohen Abstraktionsgrad mancher Topics: *action* im Sinne unserer Topicdefinition fasst eine Vielzahl einzelner Handlungselemente zusammen und abstrahiert sie zum Genre. Im Gegensatz zu kontroversen Themen wie *politics* oder *religion*, bei denen zu erwarten ist, dass sie explizit in Dialogen erwähnt und diskutiert werden (der Erwartung entsprechend liefern beide recht gute Ergebnisse), wird selbst in action-lastigen Szenen natürlich nicht *über* action *gesprochen*. Das generelle Fehlen von Handlungsanweisungen im Skript verhärtet diese Probleme.

Die Evaluation der Refined Search erfolgte für alle Topics bis auf diejenigen, die eine Location im *title* haben (z.B. *Homer tavern*), ausschließlich über die Dialoge (*SearchAdvancedOnlyInText*). Insbesondere für die selten/nie vorkommenden, abstrakten und ambigen Topics haben wir uns eine Verbesserung durch das Hinzufügen von Wortfeldern erhofft. Tatsächlich konnte eine Verbesserung der MAP auf 0,68 festgestellt werden. Es gibt keine Query mehr, die sehr wenige oder keine Ergebnisse liefert.

In der zweiten Gruppe bedeutet der Einsatz der Wortfelder besonders für die Topics *mobbing* und *lgbt* eine enorme Effektivitätssteigerung (AvgP 0,54 bzw. 0,63). *Computing* dagegen hat nun zwar Ergebnisse, aber schlechte (AvgP 0,17). An diesem Fall zeigt sich eine Schattenseite des ‚naiven' Einsatzes von Wortlisten, denn hier ist insbesondere der Begriff *bug* aus der Wortliste verantwortlich für viele nicht-relevanten Funde. *Bug* ist nicht nur ambig, sondern die für uns relevante Bedeutung des Begriffs stammt aus der Fachsprache, wodurch die Wahrscheinlichkeit, ihn tatsächlich in dieser spezifischen Bedeutung anzutreffen, sehr gering ist.

Den sprunghaften Anstieg der AvgP bei bspw. *action*, *crime*, *fantasy*, *work* werten wir als erfolgreiche Konkretisierung bzw. Disambiguierung. Bei anderen Topics scheinen zu viele Elemente in den Wortlisten eher in die Beliebigkeit zu führen oder zeugen davon, dass die

heuristische Auswahl der ‚richtigen' alternativen Suchbegriffe auch ein wenig Glückssache ist (vgl. die stark gesunkene AvgP von *super-heroes* und unser entsprechendes dictionary, das personell einen starken Fokus auf die Avengers setzt).
Abschließend betrachten wir exemplarisch einige zusammengesetzte Queries. Im Falle von *road trip* hat die Baseline eine eher schlechte AvgP, da der Begriff *road* auch als Location vorkommt und dadurch Szenen hoch gerankt werden, die an dieser Location stattfinden. Das Refinement hat die Ergebnisse für *road trip* signifikant verbessert, schon allein dadurch, dass die Suche nun auf die Dialoge beschränkt ist. *gun control* dagegen verschlechtert sich ebenso stark, hier sind wie auch in der Baseline viele nicht-relevante Szenen auf das isolierte Auffinden des Begriffs *control* zurückzuführen. o*kily dokily* schließlich als zusammengesetzter Neologismus hält seine AvgP von 1, da ein Bestandteil der Phrase nie ohne den jeweils anderen geäußert wird.
Die Suchergebnisse aller zusammengesetzten Queries können durch die gezielte Anwendung von Operatoren („*road trip*") und Keywords (*characters: Homer AND location: tavern*) stark spezifiziert werden. Dies erfordert jedoch erfahrene und versierte User.

5. Fazit

Die Suchmaschine SimpsonsSearch ermöglicht die thematische Suche nach Szenen innerhalb der TV-Serie und stellt somit eine sinnvolle Ergänzung zu anderweitig verfügbaren Themensammlungen zur Serie dar, wie z.B. erwähntem Fan-Wiki, in dem nur Episoden in Gänze erfasst sind. Besonders die Angabe des timestamps im Suchergebnis ermöglicht es Suchenden, beim Anschauen der gefundenen Folgen direkt an die passende Stelle springen zu können. Um als alleiniges Suchwerkzeug zu dienen, könnte die Suchmaschine noch mit weiteren Features und Optimierungen ausgestattet werden, die nicht auf die von uns definierten Topics beschränkt sind. Besonders bieten sich die Nutzung des Logs für die Verbesserung der Suchergebnisse und die Implementierung weiterer Ranking-Funktionen (beispielsweise BM25) an.