Qualitätsanalyse von inhaltsbasierten Empfehlungssystemen für Journals

Fachhochschule Graubünden
University of Applied Sciences

Churer Schriften
zur Informationswissenschaft

Herausgegeben von
Wolfgang Semar, Bernard Bekavac, Ivo Macek, Armando Schär

Arbeitsbereich Master of Science in Business Administration
Studienrichtung Information and Data Management

Schrift 162

Qualitätsanalyse von inhaltsbasierten Empfehlungssystemen für Journals

Mahmoud Hemila

Chur 2023

Churer Schriften zur Informationswissenschaft

Herausgegeben von Wolfgang Semar,
Bernard Bekavac, Ivo Macek, Armando Schär

Schrift 162

Qualitätsanalyse von inhaltsbasierten Empfehlungssystemen für Journals

Mahmoud Hemila

Diese Publikation entstand im Rahmen einer Thesis zum Master of Science FHGR in Business Administration, Studienrichtung Information and Data Management.

Referent: Prof. Dr. Heiko Rölke

Korreferent: Prof. Dr. Rolf Assfalg

Verlag: Fachhochschule Graubünden

ISSN: 1660-945X

Ort, Datum: Chur, Juni 2023

Abstract

Diese Arbeit analysiert, wie geeignet Empfehlungssysteme für die Auswahl von passenden Journals für die Publikation wissenschaftlicher Artikel sind. Weiterhin gibt sie Einblick auf Basis welcher Feature-Engineerings und Klassifikationssysteme Empfehlungssysteme qualitativ die besten Empfehlungen treffen können. Hierzu wurde basierend auf dem Verfahren «Klassifikation» eine Analyse von 12 verschiedenen Kombinationen aus drei Feature-Engineerings – term frequency - inverse document frequency (tf-idf), word2vec, Embedding from Language Model (ELMo) – und aus vier Klassifikationssystemen – Logistic Regression (LR), Multi-Layer Perceptron Classifier (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) – durchgeführt. Diese Kombinationen wurden mit drei verschiedenen Datensätzen aus den Fachgebieten Physik, Chemie und Biologie getestet. Die Ergebnisse dieser Analyse zeigen, dass mit der Kombination tf-idf und LR die höchste Genauigkeit von 87% (Physik), 77% (Chemie) und 73% (Biologie) für top-20 erzielt werden kann. Bei isolierter Betrachtung der erreichten Genauigkeit unterschiedlicher Klassifikationssysteme erzielte LR in Kombination mit allen drei Feature-Engineerings bei allen drei Fachgebieten durchschnittlich eine höhere Genauigkeit als jedes der anderen Klassifikationssysteme. Bei den Feature-Engineerings zeigt ein Vergleich zwischen den durchschnittlich erzielten Genauigkeiten, dass die Verwendung von word2vec und ELMo in Kombination mit allen Klassifikationssystemen zu ähnlichen Genauigkeiten führt. Die Genauigkeiten dieser Kombinationen sind zudem höher als jene von tf-idf.

Schlagwörter: Content-based Filtering, Convolutional neural Network, ELMo, Empfehlungssystem, Feature-Engineering, Journal, Klassifikationssystem, Long Short Term Memory, Recurrent Neural Network, Inhaltsbasierte Empfehlungssysteme, Logistic Regression, tf-idf, Wissenschaftliche Publikation, word2vec

Vorwort

An dieser Stelle möchte ich mich bei allen bedanken, die durch fachliche oder persönliche Unterstützung zum Gelingen dieser Arbeit beigetragen haben.

Insbesondere danke ich meinen Vorgesetzten Matthias Töwe und David Johann sowie meinem Arbeitskollegen Fabian Schneider für ihre wertvolle Unterstützung zum Verfassen bzw. Abschliessen dieser Arbeit.

Einen Ebenso grossen Dank möchte ich meinen Freunden und meiner Familie aussprechen, für ihre motivierenden Worte und die zahlreichenden inspirierenden Diskussionen. Allen voran gilt dieser Dank meiner Ehefrau Melanie für ihre grosse Unterstützung.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BIO	Biologie
BOW	Bag of Words
CB	Content-based Filtering
CF	Collaborative-based Filtering
CH	Chemie
CNN	Convolutional Neural Network
ELMo	Embedding from Language Model
LR	Logistische Regression
LSTM	Long Short Term Memory Networks
MLP	Multi Layer Perceptron Classifier
PH	Physik
RNN	Recurrent Neural Network
tf-idf	Term Frequency - Inverse Document Frequency

1 Einleitung

Für Forschende ist es von großer Bedeutung, ihre Studien in passenden und möglichst hochrangigen Journals publizieren zu können (Feng et al., 2019; Medvet et al., 2014). Die Auswahl eines passenden Journals ist aus verschiedenen Gründen jedoch nicht immer einfach (Medvet et al., 2014). Die zunehmende Zahl von Journals machen die Wahl für Forschende schwierig (Feng et al., 2019; Huynh et al., 2020; Medvet et al., 2014; Peiris & Weerasinghe, 2015; Safa et al., 2017; Wang et al., 2018). AutorInnen haben oft Mühe zu entscheiden, in welchem Journal sie publizieren möchten oder welche Journals Interesse für ihre Arbeit haben könnten, da jedes Journal bestimmte Themen und Fachgebiete abdeckt (Feng et al., 2019; Huynh et al., 2020; Safa et al., 2017; Yang & Davison, 2012).

Die Auswahl eines Journals für das eigene Manuskript ist allerdings von entscheidender Bedeutung. Wenn eine Arbeit in einem passenden Journal eingereicht wird, steigt die Wahrscheinlichkeit einer positiven Begutachtung und schliesslich der Publikation des Artikels (Medvet et al., 2014). Wenn eine Publikation hingegen abgelehnt wird, weil sie thematisch nicht passt, geht wertvolle Zeit verloren (Luong et al., 2012; Wang et al., 2018). Dies kann in bestimmten Fachgebieten wie z.B. Computer Science, wo das Innovationstempo hoch ist, ein grosses Problem sein (Luong et al., 2012). Darüber hinaus führt die Wahl des optimalen Journals dazu, dass die Arbeit interessierte Forschende erreicht und die Publikation öfter zitiert wird (Medvet et al., 2014). Um herauszufinden, welches Journal thematisch passt, wird viel Zeit in Anspruch genommen; trotzdem gibt es keine Garantie, die optimale Wahl getroffen zu haben (Safa et al., 2017). Qualitativ hochwertige Empfehlungssysteme können diese Probleme lösen und bei der schnellen Entscheidungsfindung sowie der optimalen Auswahl eines Journals helfen (Peiris & Weerasinghe, 2015; Safa et al., 2017).

In der Regel kennen AutorInnen die relevanten Journals in ihrem Fachgebiet (Feng et al., 2019; Medvet et al., 2014) und sie tendieren dazu, ihre Publikationen bei einem dieser Journals einzureichen, insbesondere in einem solchen, in welchem sie bereits erfolgreich publiziert haben (Yang & Davison, 2012). Das führt dazu, dass sie auch ihre neuen Publikationen häufig wieder im selben Journal einreichen, auch wenn für ihre neu geschriebenen Arbeiten andere Journals geeigneter sein könnten. Dies kann insbesondere dann problematisch sein, wenn sich die neue Arbeit thematisch erheblich von älteren Publikationen unterscheidet (Peiris & Weerasinghe, 2015). Genügend Vorkenntnisse, die zu einer optimalen Auswahl von Journals führen, sind nur bei recht erfahrenen AutorInnen vorhanden (Luong et al., 2012). Problematisch bleibt allerdings auch bei dieser

Personengruppe, dass neue, möglicherweise passendere Journals nicht berücksichtigt werden (Luong et al., 2012). In dieser Hinsicht könnten daher auch erfahrene AutorInnen von einem erfolgreichen Empfehlungssystem profitieren. Unerfahrene AutorInnen, die sich neu in einem Forschungsgebiet bewegen, haben aufgrund der weiter oben genannten Gründe bereits bei der Auswahl von einem geeigneten Journal Schwierigkeiten (Huynh et al., 2020; Luong et al., 2012; Safa et al., 2017; Yang & Davison, 2012). Für sie können Empfehlungssysteme bzw. die eingesetzten automatischen Mechanismen eine besonders grosse Unterstützung bieten (Huynh et al., 2020; Luong et al., 2012; Medvet et al., 2014; Safa et al., 2017).

Empfehlungssysteme sind in den letzten Jahrzehnten ausgiebig untersucht worden (Khusro et al., 2016; Mohamed et al., 2019). Es hat sich gezeigt, dass sie eine wirksame Unterstützung bei der Entscheidungsfindung in unterschiedlichen Bereichen bieten können, zum Beispiel bei der Auswahl von Musik, Filmen und Informationsmedien (Linden et al., 2003). Im akademischen Publikationsbereich haben sie ebenfalls einen Platz gefunden. Sie werden beispielsweise für Empfehlungen von Publikationen (Sun et al., 2013), von Gutachtern im Peer-Review-Prozess (Conry et al., 2009) und für Zitationsempfehlungen eingesetzt (Huang et al., 2014). Dies bedeutet aber nicht, dass sie für diese Zwecke ausreichend untersucht worden sind. Im Gegenteil, sie stellen eine Neuheit dar und haben immer noch Entwicklungsbedarf (Safa et al., 2017). Dies gilt auch für Empfehlungssysteme für Journals und Proceedings, deren Einsatz die Auswahl bzw. den Veröffentlichungsprozess beschleunigen kann (Safa et al., 2017).

Die vorliegende Studie möchte dazu beitragen, diese Lücke zu schliessen, und untersucht, wie geeignet Empfehlungssysteme für die Auswahl von passenden Journals für wissenschaftliche Publikationen sind und auf Basis welcher Techniken und Verfahren sie qualitativ gute Empfehlungen machen können.

Zunächst wird unter Kapitel 2 definiert, was Empfehlungssysteme beinhalten und welche Kategorien es gibt. Kapitel 3 widmet sich dem aktuellen Stand der Forschung. Davon werden die daraus hervorgehenden Forschungsfragen abgeleitet und in Kapitel 4 vorgestellt. Kapitel 5 beschäftigt sich schliesslich mit der zur Beantwortung dieser Fragen gewählten Methodik, mit den geprüften und letztlich verwendeten Verfahren und Vorgehensweisen zur Datenverarbeitung sowie mit dem Modellaufbau, der Durchführung und Evaluation. Anschliessend werden in Kapitel 0 die Ergebnisse vorgestellt und unter Kapitel 7 ausführlich diskutiert. Kapitel 8 zeigt die Implikationen dieser Studie für Forschung und Praxis. Kapitel 9 fasst abschliessend diese Studie zusammen und zieht ein Fazit zu den wesentlichen Erkenntnissen.

2 Empfehlungssysteme

Empfehlungssysteme sind Softwaretools und -techniken (Ricci et al., 2010; Yang & Davison, 2012), die eine große Menge von Objekten wie zum Beispiel Filme, Bücher, wissenschaftliche Publikationen, Musik usw. automatisch filtern, um diejenigen zu ermitteln, die das Interesse der Nutzenden am ehesten treffen (Luong et al., 2012). Dafür werden verschiedene Arten von Informationen, benutzerbezogenen Daten sowie die vorhandenen Artikel benutzt, damit den Nutzernden die bestgeeignete Option geliefert bzw. empfohlen wird (Ricci et al., 2010). Die Empfehlungen helfen somit bei der Entscheidungsfindung, indem eine Auswahl geeigneter Optionen basierend auf dem Interesse des Nutzers bzw. der Nutzerin präsentiert wird. So kann diese/r das gewünschte Ziel trotz der grossen Menge von Möglichkeiten und Information schneller erreichen (Ricci et al., 2010; Yang & Davidson, 2012; Safa et al., 2017).

Empfehlungssysteme können grundsätzlich in drei Kategorien unterteilt werden. Diese sind Collaborative-based Filtering, Content-based Filtering und hybride Empfehlungssysteme (Luong et al., 2012). 2019 haben Pradhan und Pal zusätzlich eine vierte Kategorie definiert: netzwerkbasierte Empfehlungssysteme.

- **Collaborative-based Filtering (CF)** Empfehlungssysteme versuchen anhand des bisherigen Verhaltens des Nutzers bzw. der Nutzerin und von anderen NutzerInnen Muster zu bilden, mit welchen der Person bei der Entscheidung geholfen bzw. für das neue Anliegen ein Element empfohlen wird, welches interessant sein könnte (Melville & Sindhwani, 2011; Safa et al., 2017). Diese Systeme sind in zwei Kategorien unterteilt. Die erste wird Memory-based Collaborative Filtering genannt und zeichnet sich dadurch aus, dass den Nutzenden ein Element empfohlen wird, welches auf Basis bereits gespeicherter Entscheidungen von anderen NutzerInnen mit ähnlichem Verhalten ausgewählt wird (Breese et al., 2013). Die zweite Kategorie ist das Model-based Collaborative Filtering, in dem das beobachtete Entscheidungsverhalten (NutzerIn-Artikel-Paare) erlernt wird und dadurch Elemente empfohlen werden, die dem NutzerInverhalten ähnlich sind (Yang & Davison, 2012). Obwohl CF weit verbreitet sind (Yang & Davison, 2012) und oft in Empfehlungssystemen eingesetzt werden (Breese et al., 2013) haben sie einige Schwächen. Diese kommen auch zum Tragen, wenn sie für die Empfehlung von Journals verwendet werden. Sun et al. (2013) sind der Meinung, dass für die Empfehlung von Journals die semantische Beziehung zwischen dem Inhalt der Publikation und dem Journal berücksichtigt werden sollte. CF sind jedoch nicht in der Lage, diese Beziehungen zu extrahieren, weil CF nur die Beziehung

zwischen dem Nutzer bzw. der Nutzerin und dem Artikel bzw. Journal erkennen kann (Sun et al., 2013). Es ist daher möglich, dass einer Person immer wieder die gleichen Journals empfohlen werden, unabhängig vom Inhalt der geplanten Publikation (Medvet et al., 2014). CF haben darüber hinaus das Problem des «Cold-Start», das heisst Ungenauigkeiten des Systems in der Anfangsphase, weil noch keine Daten über den Nutzer bzw. die Nutzerin vorhanden sind, auf deren Basis Empfehlungen gemacht werden können (Pradhan & Pal, 2019). Ein weiteres Problem liegt darin, dass durch diese Technik Artikel bzw. Journals nicht empfohlen werden, sofern sie neu und noch von niemandem ausgewählt worden sind (Pradhan & Pal, 2019).

- **Content-based Filtering (CB)** Empfehlungssysteme nutzen Inhaltsmerkmale von Artikeln, um weitere Artikel mit ähnlichen Eigenschaften zu empfehlen (Mooney & Roy, 2000). Sie basieren auf dem Konzept, dass Artikel, Elemente oder Publikationen mit ähnlichen Eigenschaften ähnlich bewertet bzw. behandelt werden sollen (Wang et al., 2018). Wang et al. (2018) gehen davon aus, dass die bei dieser Technik am meisten genutzte Informationsquelle Textdokumente sind. Ein grosser Vorteil dieser Systeme liegt darin, dass sie keine benutzerbezogenen Daten benötigen, um eine Empfehlung zu geben (Wang et al., 2018). Dies könnte eine Erklärung für die Feststellung von Cabanac (2011) sein, dass solche Empfehlungssysteme oft im wissenschaftlichen Bereich Anwendung finden. Systeme, die diese Technik einsetzen, leiden aber in der Regel unter einer begrenzten Inhaltsanalyse, was die Qualität der Empfehlung negativ beeinflussen kann (Hornick & Tamayo, 2012). So ist die Verfügbarkeit von Volltexten, welche für Empfehlungen am sinnvollsten wäre, aufgrund urheberrechtlicher Gründe eingeschränkt und die verfügbaren Abstracts sind von geringer Qualität (Pradhan & Pal, 2019; Safa et al., 2017). Ausserdem werden neue Elemente, wie zum Beispiel neu erschienene Journals oder Artikel, nicht empfohlen, weil das System diese in der Aufbauphase nicht «gesehen» hat bzw. nicht mit ihnen trainiert wurde (Pradhan & Pal, 2019).
- **Hybride Systeme** kombinieren CF und CB, um die Nachteile der jeweiligen Systeme zu vermeiden bzw. die Vorteile beider Systeme zu nutzen. Dadurch wird die Qualität der Empfehlungen erhöht (Pradhan & Pal, 2019; Safa et al., 2017). Ein solches System kann zum Beispiel für die Empfehlung von wissenschaftlichen Publikationen nach Bai et al. (2020) wie folgt aussehen: Mit CB können Profile der Forschenden erstellt werden, in denen ihre Forschungsinteressen basierend auf früheren Veröffentlichungen erfasst sind. Die CF bildet darauf aufbauend

Muster durch Entdeckung ähnlicher Forschungsinteressen zwischen den Forschenden, analysiert die von den Forschenden zitierte Literatur und empfiehlt ihnen dadurch potenzielle zitierfähigen Publikationen (Bai et al., 2020).

- **Netzwerk-basierte Empfehlungssysteme** sind eine Option, um die Probleme der CF und CB zu verringern (Pradhan & Pal, 2019). Hier wird beispielsweise in Bezug auf Empfehlungen von Journals die soziale Beziehung zwischen den AutorInnen basierend auf der Co-Autorschaft anhand eines Graphs aufgebaut. Die Knoten beinhalten Daten von AutorInnen und Journals, in denen sie in der Vergangenheit publiziert haben (Luong et al., 2012). Eine Empfehlung wird gemacht, indem analysiert wird, mit welchen AutorInnen der/die systemnutzende AutorIn Verbindungen hat und in welchen Journals diese jeweils publiziert haben. Das Journal mit den meisten Artikeln von AutorInnen dieses Netzwerks wird empfohlen (Pradhan & Pal, 2019). Laut Pradhan & Pal (2019) weist diese Technik für die Empfehlung von Journals allerdings verschiedene Probleme auf. So werden beispielsweise unabhängig vom Inhalt der Publikationen immer dieselben Journals empfohlen, solange die Arbeit vom selben Autor bzw. von der selben Autorin kommt. Zudem ist es schwierig, für neue AutorInnen, die noch keine Beziehungen zu etablierten AutorInnen haben, sowie bei neu erschienenen Journals, qualitative Empfehlungen zu generieren (Pradhan & Pal, 2019).

Für die Evaluation der Qualität von Empfehlungssystemen kommen in der Regel Genauigkeitsmetriken wie z.B. Recall, Precision und Fall-out und Fehlermetriken wie beispielsweise Root of the Mean of the Square of Errors (RMSE) und Mean of Absolute value of Errors (MAE) zum Einsatz (Cremonesi et al., 2011).

3 Stand der Forschung

Obwohl sich Empfehlungssysteme für verschiedene Zwecke etabliert haben, wie zum Beispiel für die Empfehlung von Filmen, Tags in sozialen Netzwerken, Links (Yang & Davison, 2012), Musik (Schedl & Hauger, 2015), Nachrichten, (Bobadilla et al., 2013; Turcotte et al., 2015), Webseiten und Dokumenten (Lu et al., 2015), werden sie im akademischen Publikationsbereich bislang kaum eingesetzt (Feng et al., 2019). Das traf lange auch bei Empfehlungen von Journals und Proceedings zu (Yang & Davison, 2012). Allerdings hat sich dies in den letzten Jahren langsam verändert und es sind Ansätze von Empfehlungssystemen aufgetaucht (Medvet et al., 2014). Tabelle 1 vermittelt einen chronologischen Überblick über die relevanten verfügbaren Studien bzw. (kommerziellen) Produkten in diesem Bereich. Anschliessend wird deren Inhalt und Relevanz ausführlicher erläutert. In diesen Studien wurde mehrheitlich die Metrik Genauigkeit für die Evaluation der Qualität des jeweils betrachteten Empfehlungssystems verwendet. Die Genauigkeit wurde meist an dem Auftreten bzw. Fehlen (True/False) eines definierten Targets in einer bestimmten Treffermenge («Top») bemessen. Gängige Treffermengen in diesen Studien umfassen 1, 3, 10, 15 oder 20 Treffer (z.B. Top-10). Wenn das Target-Journal in den Top-n vorhergesagt wird, bedeutet dies eine korrekte Vorhersage. Die Genauigkeit berechnet sich aus der Anzahl richtigen Vorhersagen dividiert durch die Anzahl aller Vorhersagen.

Pham et al. (2010) haben ein CF-Empfehlungssystem basierend auf dem Nutzer-Netzwerk Clustering entwickelt. Ihr Ansatz liegt darin, NutzerInnen zu gruppieren, die bei der Wahl der Journals ähnliche Muster aufweisen. Die Zuweisung von AutorInnen zu den Journals basiert auf der Anzahl Publikationen, die sie in jedem Journal publizieren. Die Gruppierungen bzw. das Netzwerk von AutorInnen basiert auf Merkmalen wie gemeinsame Publikationen (Co-Authoring), das gegenseitige Zitieren oder das Publizieren in denselben Journals bzw. die Teilnahme an den gleichen Workshops. Für ihr Experiment haben sie Daten von DBLP – einer Online-Referenzplattform über Computer Science Journals und Proceedings (dblp, 2022) – mit einem Datensatz von 1'226'412 Publikationen von 788'259 AutorInnen, die in 3'490 Journals oder Proceedings publiziert wurden, analysiert. Ihr Empfehlungssystem erreichte in der durchgeführten Studie eine Genauigkeit von 35%, wobei die Top-n nicht berichtet wurde.

AutorIn	Methode	Fachgebiet	Datensatz	Höchste Genauigkeit
Pham et al. (2010)	Nutzer-Netzwerk + Clustering	keine Angabe	1'226'412 (Train & Test)	35%
Luong et al. (2012)	AutorInnen- und Co-AutorInnennetzwerk	Computer Science	640 (Train) 154 (Test)	91.56 % - (Top-3)
Yang & Davison (2012)	Clustering + Klassifikation	keine Angabe	173'890 (Train) 10'000 (Test) + 129'927 (Train) 10'000 (Test)	78% - (Top-20)
Medvet et al. (2014)	Topic Modelling + Klassifikation	Computer Science	58'000 (Train & Test)	45.6% - (Top-10)
Safa et al. (2017)	Ähnlichkeit + AutorInnen- und Co-AutorInnennetzwerk	keine Angabe	Train: keine Angabe 20'000 (Test)	48.53% - (Top-20)
Wang et al. (2018)	Klassifikation	Computer Science	14'000 (Train & Test)	61.37% - (Top-3)
Pradhan & Pal (2019)	AutorInnen-, Zitation- und Co-Zitationsnetzwerk + Themenmodellierung + Ähnlichkeit	Computer Science (CS) + Biologie (BIO)	13'402'547 (CS Train) 2'977'587 (CS Test) + 12'848'227 (BIO Train) 2'886'334 (BIO Test)	70.8% - (Top-15)
Feng et al. (2019)	Klassifikation	Medizin	880'165 (davon Train 90%; Publikationen aus 2008-2016, Test 10%; Publikationen aus 2017)	86% - (Top-10)
Huynh et al. (2020)	Klassifikation	Computer Science	14000 (Train & Test)	89% - (Top-3)
Elsevier - Find Journals	noun phrases + Okapi BM25+	keine Angabe	1980000 (Train & Test)	keine Angabe
Springer - Journal suggester	keine Angabe	keine Angabe	keine Angabe	keine Angabe

Tabelle 1: Tabellarische Übersicht der Literatur zu Journal-Empfehlungssystemen

Luong et al. (2012) untersuchten hingegen die Eignung eines von ihnen erstellten Empfehlungssystems, das anhand von AutorInnen- und Co-AutorInnennetzwerken ein Proceeding empfiehlt. Die der Studie zugrunde liegende Idee ist, dass pro AutorIn und Co-

AutorIn ein Netzwerk von Co-AutorInnen erstellt und analysiert wird, in welchen Proceedings Manuskripte der AutorInnen in diesem Netzwerk publiziert wurden. Für das Ranking von empfohlenen Proceedings wurden drei Methoden verwendet: Zunächst wird das Proceeding empfohlen, in welchem die meisten AutorInnen publiziert haben. Als zweite Methode wird zusätzlich zur Methode 1 pro AutorIn normalisiert, da die AutorInnen, die viel publizieren, einen grösseren Einfluss auf das Ranking haben als die anderen. So hat jeder Autor bzw. jede Autorin den gleichen Einfluss auf das Ergebnis. In der dritten Methode wird neben den Massnahmen aus Methode 2 auch die Stärke der Verbindungen zwischen den AutorInnen miteinbezogen.

Für das Trainieren bzw. Aufbauen des Netzwerks wurden bei Luong et al. (2012) die Angaben über AutorInnen und Co-AutorInnen von 640 Publikationen aus vier Subdomains im Bereich Computer Science verwendet, die in 16 Proceedings publiziert wurden. Für das Netzwerk der AutorInnen und Co-AutorInnen der 640 Publikationen wurden Daten von Microsoft Academic Search (MAS) verwendet. Damit konnten Verbindungen von 115'000 AutorInnen zu 160'000 Publikationen hergestellt werden. Zum Testen des Konzepts haben Luong et al. (2012) 154 Publikationen benutzt. Das beste Ergebnis erreicht die dritte Methode mit einer Genauigkeit von 56% für Top-1, 79.87% für Top-2 und 91.56% für Top-3. Die Studie zeigt allerdings nicht, wie mit neuen AutorInnen ohne Publikation, die also noch keine Verbindungen zum Journal haben, umgegangen wird. Ausserdem besteht in diesem System die Einschränkung, dass jede Publikation desselben Autors bzw. derselben Autorin, unabhängig vom eigentlichen Thema der Arbeit, die gleiche Empfehlung erhält (Medvet et al., 2014).

Yang & Davison (2012) haben sich mit der Frage beschäftigt, ob Publikationen mit ähnlichem Schreibstil eher von ähnlichen Journals angenommen werden. Ihre Annahme ist, dass Publikationen, die ähnliche Themen behandeln und somit in denselben oder vergleichbaren Journals publiziert werden, ähnlich strukturiert sind bzw. einen ähnlichen Schreibstil haben. So kann anhand des Schreibstils einer Publikation eine Kategorie zugewiesen und eine Verbindung zu einem Journal hergestellt werden, welches diese Kategorie abdeckt. Dieses Journal gilt dann als Empfehlung des Systems. Das Experiment wurde zweimal mit unterschiedlichen Datensätzen zum Trainieren verwendet. Ein Datensatz stammte von ACM Digital Library mit 172'890 Publikationen, der zweite Datensatz von CiteSeer Digital mit 119'927 Publikationen. Zum Testen wurden für beide Experimente Daten im Umfang von 10'000 Publikationen benutzt. Das beste Ergebnis in dieser Studie konnte mit den Daten von ACM Digital Library mit Genauigkeit von 55.7% für Top-5, 69% für Top-10 und 78% für Top-20 erreicht werden.

Medvet et al. (2014) gingen davon aus, dass jedes Journal und jedes Proceeding nur auf bestimmte Themen in einem Fachgebiet fokussiert ist. Sie haben ein System aufgebaut, welches eine Publikation einem bestimmten Thema zuordnet und dementsprechend ein Journal oder ein Proceeding empfiehlt, welches dieses Thema abdeckt. Für ihre Studie haben sie einen Datensatz von 58'000 Publikationen im Bereich Computer Science nur unter Berücksichtigung von Titel und das Abstract verwendet. Der Datensatz deckte 300 Klassen bzw. Proceedings und Journals ab. Für ihr System haben sie mit drei unterschiedlichen Methoden experimentiert: Canvar-Trenkle – eine N-Gramm-basierte Text-Kategorisierung (Canvar & Trenkle, 1994) –, Two-step-LDA (Collection Mode und Item Mode) – eine Methode, welche auf dem Konzept eines probabilistischen Themenmodells und auf der Latent Dirichelt Allocation basiert (Blei et al., 2003) – und LDA+Clustering, womit Publikationen anhand ihrer Themenwahrscheinlichkeit in Cluster eingeteilt werden. Die Ergebnisse der Studie von Medvet et al. (2014) haben gezeigt, dass die höchste Genauigkeit mit der Canvar-Trenkle-Methode erreicht werden kann, eine Quote von 26.08% für die Top-3, 34% für die Top-5 und 45.6% für die Top-10. Die Ergebnisse der Studie können allerdings nicht verallgemeinert werden, weil ausschliesslich Journals im Bereich Computer Science und keine weiteren Fachgebiete berücksichtigt wurden.

Eine ähnliche Studie wie Luong et al. (2012) haben Safa et al. (2017) durchgeführt. Das System von Safa et al. (2017) berücksichtigt aber nicht nur AutorIn und Co-AutorIn, sondern auch den Titel der Publikationen. Safa et al. (2017) geben an, ihr System basiere auf der CB-Technik. Nach Einschätzung des Autors dieser Studie wäre es allerdings als hybrid (CF/Netzwerk + CB) zu klassifizieren. Ihr System versucht ähnliche Publikationen von Co-AutorInnen durch Abgleich von Titeln mit dem eingegebenen Titel zu finden. Die Journals, in denen ähnliche Publikationen vorhanden sind, werden in eine Rangfolge gebracht und empfohlen. Diese Herangehensweise bzw. dieses Verfahren kann rekursiv für weitere Co-AutorInnen von Co-AutorInnen durchgeführt werden. Die Ähnlichkeit zwischen den Titeln ermitteln sie mit der Term-Häufigkeit und dem Ansatz der Cosinus Similarity. Das Ranking der Journals basiert auf der Gewichtung jedes Journals im Zusammenhang mit der gesuchten Publikation. Die Gewichtung eines Journals wird aus der Summe der Cosinus Similarities zwischen dem Titel der jeweiligen Publikation in diesem Journal und dem gesuchten Titel berechnet. Journals werden nach Gewichtung sortiert und dem Nutzer bzw. der Nutzerin präsentiert. Safa et al. (2017) haben für diese Studie Daten aus dem Digital Bibliography & Library Project (DBLP) sowie der Computer Science bibliographic Website benutzt. Als Basis für dieses Konzept wurden Daten aus den Jahren 2008 bis 2012 sowie für den Aufbau des AutorInnen-Netzwerks Daten über AutorInnen von Publikationen aus den Jahren 2003 bis 2012 genutzt. Für die Evaluation

des Konzepts wurden 20'000 Publikationen aus dem Jahr 2013 verwendet. Das beste Ergebnis mit ihrem Konzept erreichte eine Genauigkeit von 48.53% für Top-20 mit der Tiefe 1, wobei Tiefe 1 ausdrückt, dass das Netzwerk nur Co-AutorInnen von Co-AutorInnen berücksichtigt und nicht tiefer geht. Es wurde nicht diskutiert bzw. gezeigt, wie mit der verbreiteten Problematik neuer AutorInnen, die keine oder neue Co-AutorInnen haben, umgegangen wird und ob das System in diesen Fällen nur anhand des Titels eine Empfehlung ausgeben kann.

Wang et al. (2018) haben ein Empfehlungssystem für Journals und Proceedings im Bereich Computer Science vorgestellt, welches gute Ergebnisse lieferte. Ihr System basiert auf der CB-Technik. Um das Modell zu trainieren, haben sie Abstracts von Publikationen von einem Datensatz bestehend aus etwa 14'000 Publikationen aus 28 Journals und 38 Proceedings verwendet. Wang et al. (2018) haben für die Extraktion der Features die Methode «Term Frequency - Inverse Document Frequency» (tf-idf) verwendet, um die Relevanz eines Begriffs für ein spezifisches Dokument in einem Pool von Dokumenten festzulegen. Die Feature-Extraktion ist auch als Feature-Engineering bekannt und bezeichnet Modelle oder Methoden, die Texte in Form von nummerischen Vektoren darstellen (Sarkar, 2019). Wang et al. (2018) haben ausserdem die statistischen Methoden "Chi-Quadrat", "Mutual information" und "Information gain" verwendet, um die Anzahl der Features auf diejenigen Begriffe zu reduzieren, die eine klare Abhängigkeit zu den Kategorien bzw. zu den Journals haben (Wang et al., 2018). Ihr Modell nutzt zur Klassifizierung die Funktion «Softmax Regression», eine Erweiterung der logistischen Regression, welche «Multi-Class»-Probleme lösen kann (Wang et al., 2018). Das Ergebnis dieser Studie zeigt, dass das Modell mit der Verwendung von tf-idf und Chi-Quadrat die höchste Genauigkeit von 61.37% für die Top-3 erreicht.

Pradhan und Pal (2019) haben ein umfassendes Empfehlungssystem namens DISCOVER entwickelt. Dieses berücksichtigt mehrere Aspekte, die eine Rolle für die Auswahl eines Journals spielen können. Es sind Aspekte wie das soziale Netzwerk des Autors bzw. der Autorin, Zitationen und Co-Zitationen, Themenmodellierung basierend auf der kontextuellen Ähnlichkeit und die Identifizierung von Schlüsselrouten auf der Grundlage der Hauptpfadanalyse eines bibliografischen Zitationsnetzwerks. Für ihr System verwendeten sie Daten von Microsoft Academic Graph (MAG). Die Daten enthalten Publikationen in den Bereichen Computer Science (CS) und Biologie (BIO), die zwischen 1982 und 2012 publiziert wurden. Zum Trainieren des Systems sind Daten von Publikationen zwischen 1982 und 2011 (10'4249'60 (CS) und 9'961'893 (BIO)) benutzt worden. Für das Testen sind Daten aus dem Jahr 2012 (2'977'587 (CS) und 2'886'334 (BIO)) verwendet

worden. Das beste Ergebnis liefert das System mit einer Genauigkeit von 70.8% (CS) und 69.7% (BIO) für Top-15.

Im Jahr 2019 haben Feng et al. Pubmender, ein Deep Learning-basiertes Empfehlungssystem, entwickelt, welches Journals im Fachgebiet Biomedizin empfiehlt. Pubmender basiert auf der CB-Technik, wobei nur das Abstract als Analyseobjekt für die Studie verwendet wurde. Feng et al. (2019) haben word2vec als Methode für das Feature-Engineering und ein Convolutional Neural Network (CNN) mit drei Schichten und softmax-Operation zum Trainieren des Modells verwendet. Für die Studie haben Feng et al. (2019) einen Datensatz mit 880'165 Publikationen von 1130 Journals benutzt. Diese Studie hat gezeigt, dass dieses Modell eine Genauigkeit von 50% für Top-1 und 86% für Top-10 erreicht. Feng et al. (2019) haben ihr System selbstkritisch betrachtet und sind der Meinung, dass man die Evaluationen des Modells nur mit Vorsicht geniessen sollte. Als grosses Problem betrachten sie die Aufteilung von Daten. So gab es im Datensatz Journals wie «PLOS One», das mit 153'608 Publikationen extrem häufig vertreten war, während andere Journals wie «Horticulture Research» mit nur 100 Publikationen deutlich seltener vertreten war (Feng et al., 2019). Ausserdem haben sie 90% ihres Datensatzes (Publikationen zwischen 2008 und 2016) zum Trainieren und nur 10% (Publikationen von 2017) zum Testen benutzt. Durch diese Aufteilung könnte es sein, dass das Modell mit Journals trainiert wurde, auf die es nicht getestet wurde. In den Testdaten (2017) kann es sein, dass nicht alle Zeitschriften, die in Trainingsdaten (2008-2006) vorhanden sind, abgedeckt sind. Hintergrund dafür ist, dass im Jahr 2017 möglicherweise nicht in allen der 1130 Journals publiziert wurde. Umgekehrt könnte es auch sein, dass das Modell mit Journals getestet wurde, auf die es nicht trainiert wurde, wenn im Jahr 2017 ein neues Journal im Fachbereich erschienen ist. Der Datensatz ist somit nicht balanciert aufgeteilt, was die Verlässlichkeit der Ergebnisse in Frage stellt (Mazumder, 2021).

Huynh et al. (2020) haben die Studie von Wang et al. (2018) erweitert und ebenfalls ein Empfehlungssystem für Journals und Proceedings im Bereich Computer Science aufgebaut, welches eine noch höhere Genauigkeit erzielt. Wie das in der Studie von Wang et al. (2018) vorgestellte System basiert ihr Modell ebenfalls auf der CB-Technik und es wird derselbe Datensatz zum Trainieren und Testen des Systems verwendet. Im Unterschied zu Wang et al. (2018) haben Huynh et al. (2020) für die Entwicklung ihres Empfehlungssystems aber den Titel, das Abstract und die Keywords verwendet. Sie haben das Modell mit sieben unterschiedlichen Kombinationen trainiert und getestet. Die Kombinationen sind «Titel», «Abstract + Keywords», «Titel + Abstract», «Abstract + Keywords», «Keywords + Titel», sowie «Titel + Abstract + Keywords». Für die Feature-

Extraktion haben sie die Methode «tf-idf» und «Chi-Quadrat» für Titel und Abstracts sowie «One hot encoding» und «Chi-Quadrat» für Keywords verwendet. Für das Experiment wurden die zwei Klassifikatoren Linear Logistic Regression und Multi-Layer Perceptron trainiert und getestet. Das beste Ergebnis hat ein auf Multi-Layer Perceptron basierendes System in der Kombination von «Titel + Abstract + Keyword» mit einer Genauigkeit von 89.07% für die Top 3 geliefert. Das System, das in dieser Studie vorgestellt wurde, arbeitete genauer als das von Wang et al. (2018). Beide Studien weisen aber einige Schwachpunkte auf:

- Die entwickelten Modelle sind fachbereichsabhängig. Sie eignen sich also nur für den Bereich Computer Science, weil sie nicht mit Daten von anderen Fachgebieten trainiert oder evaluiert sind.
- Der verwendete Datensatz deckt nur 66 Kategorien bzw. Journals und Proceedings ab. Dies ist nur ein kleiner Anteil von allen Journals und Proceedings in diesem Bereich, welche mehr als 13'000 umfassen (Huynh et al., 2020).
- In der Anwendungsform der Studie von Huynh et al. (2020) ist der Datensatz unvollständig. Er beinhaltet lediglich Angaben über etwa 14'000 Publikationen, enthält aber nicht alle dazugehörigen Keywords (Huynh et al., 2020). Huynh et al. (2020) haben auch nicht erwähnt, ob diese fehlenden Keywords beim Trainieren und Testen bei der Verwendung der Kombinationen (Keywords + Abstract + Keywords), sowie (Titel + Keywords) weiter (Abstract + Titel + Keywords) berücksichtigt worden sind oder wie mit ihnen umgegangen wurde.

Verlage wie zum Beispiel Elsevier und Springer haben eigene Empfehlungssysteme für Journals aufgebaut. Das System von Elsevier nennt sich *Find Journals* und verlangt Titel, Abstract, Keywords (nur eine bereits vordefinierte Auswahl) und das Fachgebiet als Input. Es liefert als Output eine Liste von Journals mit Angaben wie Journal Impact Factor und Akzeptanzrate (Elsevier, 2022). Das System ist mit Daten von 1’980’000 Publikationen der Datenbank Scopus trainiert und getestet worden. Find Journals nutzt die «noun phrases» für das Feature-Engineering und Okapi BM25+ für das Ranking bzw. für die Empfehlung. Die Empfehlungen sind nur auf Werke der Elsevier-Verlage beschränkt (Kang et al., 2015).

Springer bietet ebenfalls ein Empfehlungssystem für Journals, mit dem Namen *Journal suggester*. Dieses erwartet einen Titel, einen Abstract und das Fachgebiet als Input. Als Output liefert das System eine Liste von Journals mit Angaben zum Impact Factor, der durchschnittlichen Dauer bis eine Antwort vom Journal auf eine Publikationsanfrage

erwartet werden kann, sowie zur Akzeptanzrate. Dieses System ist beschränkt auf Journals von Springer und BioMed Central (BMC) (Springer Nature, 2020).

Beide Verlagsprojekte eint, dass die Empfehlungen auf ihre eigenen Publikationen beschränkt sind.

4 Forschungsziele

Der aktuelle Stand der Forschung bietet ein Bild verschiedener Systeme mit unterschiedlicher Qualität. Diese Qualitätsunterschiede können unter anderem dadurch begründet sein, dass die Systeme auf verschiedenen Techniken basieren, auf unterschiedlich grossen Datensätzen aufgebaut sind und zum Teil auf unterschiedlichen Klassifikationssystemen basieren. Bislang blieb jedoch die Frage unbeantwortet, was tatsächlich die Qualität eines Empfehlungssystems für Journals beeinflusst. Die vorliegende Studie soll dazu beitragen, dies zu ändern und die Forschungsgrundlagen schaffen, um ein qualitativ hochwertiges Empfehlungssystem für wissenschaftliche Journals zu entwickeln. Allgemein kann mit Blick auf die bestehende Studienlage festgehalten werden, dass folgende Systeme tendenziell am vielversprechendsten sind:

- CB Systeme basierend auf dem Verfahren «Klassifikation»
- CF oder netzwerkbasierte Systeme, basierend auf sozialen Netzwerken der AutorInnen

Da CF und netzwerkbasierte Systeme den grossen Nachteil haben, dass sie keine hinreichend guten Empfehlungen für Publikationen von neuen AutorInnen sowie von neu aufgenommenen Journals bieten können, wurde sie im Rahmen dieser Studie ausgeschlossen. Untersucht wurde daher der Einsatz von CB Empfehlungssystemen basierend auf dem Verfahren Klassifikation. Zur Entwicklung eines erfolgreichen Empfehlungssystems dienten die folgenden forschungsleitenden Fragen:

- Wie gut können CB-Empfehlungssysteme für Journals sein?
- Mit welchen technischen Rahmenbedingungen können CB-Empfehlungssystemen qualitativ gute Empfehlungen bieten?

Auf Basis der bestehenden Forschung ergeben sich folgende konkreten Fragestellungen:

- In den vorhandenen Studien wurden unterschiedliche Methoden für das Feature-Engineering verwendet. Es fehlt allerdings eine Aussage darüber, ob die verwendete Methode eine Rolle für die Qualität des Systems spielt, und wenn ja, welche Methode die beste Qualität für ein Empfehlungssystem liefern könnte. Dies führt zu folgender Forschungsfrage:
  - *F1: Welches Feature-Engineering erzielt die beste Qualität für ein Empfehlungssystem für Journals?*
- Die bisher durchgeführten Studien haben jeweils unterschiedliche Klassifikationssysteme genutzt. Neben der Analyse von Huynh et al. (2020) gibt es keine weitere

Studie, welche die gleichen Experimente mit mehreren Klassifikatoren durchgeführt hat, um zu zeigen, mit welchem Klassifikator die höchste Qualität erreicht werden kann. Huynh et al. (2020) haben in ihrer Analyse nur die Klassifikationssysteme Logistic Regression und Multi-Layer Perceptron mit einem kleinen Datensatz verwendet. Um eine generalisierbare Aussage über den Einfluss der Klassifikationssysteme treffen zu können, wird im Rahmen dieser Arbeit als Erweiterung zu den Ergebnissen von Huynh et al. (2020) der mögliche Einfluss der Klassifikationssysteme auf die Qualität der Empfehlungssysteme untersucht. Daraus ergibt sich die folgende Forschungsfrage:

  - *F2: Auf Basis welches Klassifikationssystems kann ein Empfehlungssystems die beste Qualität erzielen?*

- Es fehlt ebenfalls ein Überblick oder Hinweise, ob und welchen Einfluss die Kombinationen bestimmter Feature-Engineering und Klassifikationssysteme auf die Qualität eines Empfehlungssystems haben. Auch die möglichen Wechselwirkungen zwischen Feature-Engineering und Klassifikationssystem verdienen Beachtung auf der Suche nach den Rahmenbedingungen, welche die beste Qualität ermöglichen. Deshalb lautet die dritte Forschungsfrage:
  - *F3: Mit welcher Kombination von Feature-Engineering und Klassifikationssystem kann ein Empfehlungssystem die beste Qualität erzielen?*
- Die Mehrheit der Arbeiten bzw. der Experimente beschränkte sich auf die Nutzung von Daten aus dem Bereich Computer Science. Nur eine Studie nutzte ausschliesslich Daten aus dem medizinischen Bereich und eine andere neben Computer Science auch Daten aus der Biologie. Es gibt bisher noch keine Studie, die versucht hat, herauszufinden, was für eine Rolle Daten aus unterschiedlichen Fachgebieten bei der Qualität eines Empfehlungssystems spielen könnten. Dieses Wissen ist sehr relevant für den Aufbau von Empfehlungssystemen für Journals. Sollte es zwischen Fachgebieten Unterschiede geben, könnte dies bedeuten, dass für unterschiedliche Fachgebiete bessere Kombinationen von Feature-Engineerings und Klassifikationssystemen gesucht bzw. untersucht werden müssen. Dies führt zur folgenden Forschungsfrage:
  - *F4: Inwiefern ändert sich die Qualität eines Empfehlungssystems für Journals durch die Verwendung von Datensätzen aus unterschiedlichen Fachgebieten?*

5 Methodik

Um entsprechend den gestellten Forschungsfragen Vergleiche zu ermöglichen, wurden in dieser Studie mehrere Feature-Engineerings, Klassifikationssysteme und Daten von unterschiedlichen Fachgebieten verwendet. Im Folgenden werden die Hintergründe der jeweiligen Auswahl erläutert sowie die gewählten Methoden erklärt.

5.1 Auswahl der Methoden

Zur Durchführung dieser Arbeit musste aus einer breiten Palette möglicher Engineerings, Klassifikationssysteme und Fachgebiete eine Auswahl getroffen werden.

5.1.1 Feature-Engineerings

Feature-Engineerings sind Modelle oder Methoden, die Texte in Form von nummerischen Vektoren darstellen, damit sie in maschinellen Lernverfahren viel einfacher angewendet werden können (Sarkar, 2019). Entscheidender Faktor bei der Auswahl der verwendeten Feature-Engineerings war, dass sie auf jeweils unterschiedlichen, in der Anwendung für diese Fragestellung erfolgsversprechenden Logiken basieren sollten. Dies führte zu einer Auswahl von insgesamt drei Methoden. Je eine aus den folgenden verschiedenen Kategorien von Feature-Engineerings wurde verwendet:

A) Traditional Feature Engineering

Engineerings dieser Art repräsentieren für jedes Dokument in einem Korpus einen Vektor, welcher genauso viele Werte besitzt wie die Anzahl einzigartiger Wörter in einem Korpus (Sarkar, 2019). Jeder Wert repräsentiert die Gewichtung eines Wortes in einem Dokument (Sarkar, 2019). Basierend auf dieser Logik standen beispielsweise die Engineerings Bag of Words (BOW) und tf-idf zur Auswahl. BOW hat den Nachteil, dass die Gewichtung eines Wortes nur auf der Häufigkeit des Wortes innerhalb eines Dokuments basiert, nicht aber auf der Häufigkeit in anderen Dokumenten in einem Korpus (Siddharth M, 2021). Tf-idf löst dieses Problem (Siddharth M, 2021). Darüber hinaus bot sich die Verwendung von tf-idf an, da es von Huynh et al. (2020) verwendet wurde und sich dort mit einer hohen Genauigkeit (89%) bewährte.

B) Word Embedding

Es handelt sich hier um Engineerings, die jedes Wort in einem Dokument durch einen n-dimensionalen Vektor repräsentieren (Kathrani, 2020). Mit Feature-Engineering basierend auf dieser Logik werden die Wörter im Zusammenhang mit ihrem Kontext

berücksichtigt (Siddharth M, 2021). Dadurch haben Wörter mit ähnlicher Bedeutung (z.B. boat und ship) sowie Wörter mit semantischem Zusammenhang (z.B. boat und water) ähnliche vektorielle Repräsentationen, weil sie in der Regel einen ähnlichen Kontext haben (Ogundepo, 2021).

Die meistverbreiteten Modelle bzw. Engineerings, die auf dieser Logik basieren, sind word2vec und GloVe (Kathrani, 2020). Beide Modelle liefern normalerweise ähnliche Ergebnisse, obwohl sie unterschiedlich trainiert werden (MLNerds, 2019). Es wurde für vorliegende Studie die Methode word2vec verwendet. Word2vec wurde bereits im Rahmen der Analyse von Feng et al. (2019) verwendet und hat dort eine hohe Genauigkeit (86%) erreicht. Es sollte daher überprüft werden, ob mit word2vec eine ähnlich hohe Genauigkeit würde erreicht werden können.

C) Sentence Embedding

Ein Nachteil der Modellgruppe - des sogenannten Word Embedding - ist, dass der Kontext von Wörtern, also die Sätze, in denen sie erwähnt werden, nicht berücksichtigt wird (Kathrani, 2020). Das heisst, Wörter, die mehr als eine Bedeutung haben (z.B. Bank, Apple), werden immer die gleiche vektorielle Repräsentation haben, unabhängig von der eigentlichen Bedeutung oder dem Kontext (Kathrani, 2020). Durch die Verwendung von Modellen, die auf dem sogenannten Sentence Embedding basieren, sollte dieses Problem vermieden werden (Kathrani, 2020). Modelle, die auf vorgenannte Logik basieren, sind beispielsweise ELMo, InferSent und Sentence-BERT (Kathrani, 2020). Da in keiner der für diese Arbeit zur Analyse verfügbaren Studien eines dieser Modelle verwendet wurde, gab es keine konkreten Anhaltspunkte, welches davon erfolgreich auf ein Empfehlungssystem für Journals angewendet werden könnte. Es wurde für die Studie aus rein experimentellen Gründen das Verfahren ELMo gewählt. Die anderen beiden Modelle wurden ausgeschlossen, weil mehrere Modelle, die auf einer ähnlichen Logik basieren, vermieden werden sollen.

5.1.2 Klassifikationssysteme

Nach der Festlegung, mit welchen Feature-Engineerings Features extrahiert werden, wurden die Klassifikationssysteme, mithilfe derer die Analyse durchgeführt wird, ausgewählt. Klassifikationssysteme sind Modelle, die mit gelabelten Daten trainiert werden. Das heisst, es ist bekannt, welcher Datenpunkt zu welcher Klasse – welchem Label – gehört. Damit ist man in der Lage, Datenpunkte (z.B. Abstracts) vordefinierte Kategorien bzw. Klassen (Journals) zuzuordnen (angelehnt an die Definition von Müller, 2022). Bei der Auswahl wurden zwei Kriterien berücksichtigt. Erstens musste das

Klassifikationssystem in der Lage sein, Multi-Klassen-Probleme zu lösen. Das bedeutete in diesem Zusammenhang, dass eine Vorhersage in Form einer Liste mit Wahrscheinlichkeiten aller Klassen ausgegeben werden sollte. Dies war notwendig, weil in der vorhandenen Analyse nicht nur eine Klasse, sondern mehrere vorhergesagt werden sollten. Zweites Kriterium war, dass die Klassifikationssysteme sowohl auf den klassischen Machine Learning-Klassifikationsverfahren als auch auf Deep Learning-Algorithmen basieren sollten. Es erfüllten mehr Verfahren diese Kriterien, als im Rahmen dieser Arbeit abgebildet werden konnten. Daher wurde die Auswahl hier auf insgesamt vier Klassifikationsverfahren mit unterschiedlichen Mechanismen beschränkt.

Für die Auswahl wurden zuerst 14 Klassifikationssysteme von *Scikit-learn* (in Python: sklearn) – ein Python-Module für Machine Learning – als dahingehend untersucht, ob sie das erste Kriterium erfüllten; Das war bei fünf Verfahren der Fall. Die Klassifikationssysteme wurden mit einem Sample-Datensatz von 15’019 Einträgen bzw. Abstracts und 575 Klassen mit den Standardeinstellungen trainiert und getestet, um die besten zwei auszuwählen. Die Features wurden für diesen Zweck mit dem Feature-Engineering tf-idf extrahiert. Die höchsten Genauigkeiten konnten die Verfahren MLPClassifier (23%) und LogisticRegression (18%) erzielen. Es wurde daher entschieden, für diese Analyse diese beiden Verfahren zu verwenden. Tabelle 2 bildet diesen Auswahlprozess ab.

Klassifikationsverfahren	**Kriterium 1**	**Genauigkeit**	**Klassifikationsverfahren**	**Kriterium 1**	**Genauigkeit**
SGDClassifier	Nein		NearestCentroid	Nein	
Perceptron	Nein		RandomForestClassifier	Ja	15%
PassiveAggressiveClassifier	Nein		LogisticRegression	Ja	18%
BernoulliNB	Nein		MLPClassifier	Ja	23%
MultinomialNB	Ja	10%	LinearSVC	Nein	
GaussianNB	Nein		GradientBoostingClassifier	Ja	11%
CategoricalNB	Nein		KNeighborsClassifier	Nein	

Tabelle 2: Auswahl Klassifikationssysteme von Scikit-learn

Bezüglich Deep Learning Klassifikationsverfahren wurde für diese Arbeit entschieden, mit Convolutional Neural Networks (CNN) sowie mit Recurrent Neural Networks (RNN) - ganz spezifisch RNN mit Long Short Term Memory Networks (LSTM) - zu arbeiten. CNN kam in dieser Studie zur Anwendung, weil es in der Analyse von Feng et al. (2019)

verwendet wurde und eine hohe Genauigkeit (86%) erzielen konnte. Es soll untersucht werden, ob damit auch bei Daten aus anderen Fachgebieten und mit unterschiedlichen Feature-Engineerings eine ähnlich hohe Genauigkeit erzielt werden kann. RNN kam in dieser Studie aus experimentellen Gründen zum Einsatz, da sie in keiner der dargestellten Analysen verwendet wurde. Bei RNN bzw. LSTM spielt im Gegensatz zu anderen neuronalen Netzen die Reihenfolge der betrachteten Elemente eine Rolle (colah's blog, 2015). Daher sollte untersucht werden, ob RNN aufgrund dieser Eigenschaft im Vergleich zu den anderen Klassifikationssystemen eine bessere Qualität erzielen würde. Das könnte erreicht werden, indem das Verfahren die Reihenfolge der Wörter in einem Text erkennen und im Zusammenhang mit den Klassen Muster bilden könnte.

Als Alternative zu den beiden ausgewählten neuronalen Netzen gab es weitere Optionen zur Auswahl: zum Beispiel Generative Adversarial Networks (Croce et al., 2020), Radial Basis Function Networks (Shao, 2020). Da aus Kapazitätsgründen nur vier Verfahren angewendet werden konnten, wurden die vorgenannten beiden Optionen ausgeschlossen.

5.1.3 Fachgebiete

Zuletzt wurden die einzubeziehenden Fachgebiete festgelegt. Aus Kapazitätsgründen wurde diese Wahl auf drei Fachgebiete beschränkt. Eine Limitierung auf nur zwei Fachgebiete würde nicht ausreichen, um Unterschiede oder Ähnlichkeiten in allen Fachgebieten nachzuweisen. Diese könnten ihre Ursache in bestimmten Eigenheiten dieser beiden Fachgebiete haben. Die Nutzung von Daten aus drei Fachgebieten sollte diese Problematik reduzieren, indem geprüft wurde, ob die Ergebnisse in allen Fällen vergleichbar wären oder ob sich das Ergebnis in einem Fachgebiet erheblich unterscheiden würde. Alles in allem sollte die Anzahl von drei Fachgebieten eine genügend grosse Generalisierbarkeit der Ergebnisse gewährleisten.

Die drei Fachgebiete, die in dieser Studie berücksichtigt werden, sind Physik (PH), Chemie (CH) und Biologie (BIO). Die Auswahl der Fachgebiete erfolgte aus praktischen Gründen und vor dem Hintergrund, dass die Genehmigung der Plattform «Dimensions» (Digital Science) zur Nutzung ihrer sehr umfassenden Datenbank auf Betreiben der Bibliothek der ETH Zürich, der Arbeitgeberin des Autors, zustande kam. Die Institution ETH Zürich ist auf Naturwissenschaften fokussiert und plant, auf Basis dieser Forschungsergebnisse ein Empfehlungssystem zu entwickeln.

5.2 Verwendete Verfahren

Im Folgenden werden die Verfahren vorgestellt, die auf Basis der in Kapitel 5.1 erläuterten Überlegungen für diese Studie ausgewählt wurden.

5.2.1 Feature-Engineerings

In diesem Kapitel werden die drei ausgewählten Feature-Engineering beschrieben und deren Funktionsweisen erläutert.

5.2.1.1 Term frequency - Inverse document frequency (tf-idf)

Tf-idf ist eine häufig verwendete Methode im Text Mining[1] (Huynh et al., 2020). Jedem Wort in einem Text wird von tf-idf ein Gewichtungswert zugewiesen (Sarkar, 2019). Die Gewichtung basiert auf der Häufigkeit des Wortes in einem Dokument unter Betrachtung der Anzahl Dokumente, die dieses Wort ebenfalls enthalten, in Relation mit sämtlichen vorhandenen Dokumenten in dem Korpus (Huynh et al., 2020; Sarkar, 2019). Die Gewichtung wird verwendet, um die Wichtigkeit eines Wortes in einem Dokument zu bewerten (Huynh et al., 2020). Das Resultat des Engineerings ist eine Feature Matrix. Das bedeutet, dass pro Dokument ein Vektor mit so vielen nummerischen Werten erstellt wird, wie es einzigartige Wörter im ganzen Korpus gibt. Wenn ein Wort im Dokument vorhanden ist, bekommt dieses einen Gewichtungswert, ansonsten bekommt dieses Wort den Wert Null. So entstehen am Ende so viele Vektoren wie es Dokumente gibt und alle Vektoren sind exakt gleich lang (Sarkar, 2019). Tf-idf kann auch für die Erstellung einer Sparse Matrix mit n-grams zur Kombination von mehreren Wörtern in einer Reihenfolge verwendet werden (scikit-learn, 2022c). Der folgende Code, sowie die darunter dargestellte Ergebnistabelle, enthalten ein einfaches Beispiel mit 4 kleinen vorbereiteten Texten:

```
docs=['cilium like membrane protrusion', 'emanate surface vertebrate cell
classify', 'motile cell primary cilium motile cilia', 'move fluid flow cell']
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer()
matrix_docs = tv.fit_transform(docs)
matrix_array = matrix_docs.toarray()
vocab = tv.get_feature_names_out()
pd.DataFrame(np.round(matrix_array, 2), columns=vocab)
```

[1] «[…] text mining is defined as the methodology and process followed to derive quality and actionable information and insights from textual data.» (Sarkar, 2019, S. 66)

	cell	cilia	cilium	classify	emanate	flow	fluid	like	membrane	motile	move	primary	protrusion	surface	vertebrate
0	0.00	0.00	0.41	0.00	0.00	0.00	0.00	0.53	0.53	0.00	0.00	0.00	0.53	0.00	0.00
1	0.30	0.00	0.00	0.48	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.48
2	0.24	0.38	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0.75	0.00	0.38	0.00	0.00	0.00
3	0.35	0.00	0.00	0.00	0.00	0.54	0.54	0.00	0.00	0.00	0.54	0.00	0.00	0.00	0.00

5.2.1.2 Word2vec

Die klassischen «vector space»-Modelle wie beispielsweise tf-idf können zu höherer Dimensionalität und Sparse-Problemen[2] führen, was die Qualität eines Modells negativ beeinflussen könnte (Feng et al., 2019). Durch die Verwendung einer «word embedding»-Methode wie z.B. word2vec kann dieses Problem vermieden werden (Feng et al., 2019). Ausserdem ist die Nutzung dieser Methode für das Trainieren von neuronalen Netzen im Vergleich zu den traditionellen Feature-Engineering-Modellen von Vorteil (Collobert et al., 2011).

Word2vec ist eine auf einem neuronalen Netz basierende Methode für das Transformieren von Wörtern zu nummerischen Werten. Als Input wird eine Liste mit einzelnen Wörtern verwendet und als Output wird ein Vektor für jedes Wort geliefert. Dieser Vektor beinhaltet nummerische Werte, die das jeweilige Wort repräsentieren (Ayyadevara, 2018). Die Länge der Werte kann vorgängig definiert werden. Vektoren von Wörtern, die einen gemeinsamen Kontext in einem Korpus haben, haben weniger Abstände; sie sind sich also ähnlicher (Di Pietro, 2020). Man kann ein bereits trainiertes word2vec Modell oder ein Modell mit den Wörtern vom eigenen Korpus trainieren und verwenden (Di Pietro, 2020). Als Beispiel zeigt der untenstehende Code ein Modell, welches mit jedem Wort von 100 Dokumenten trainiert ist. Dieses soll für jedes Wort einen Vektor mit 5 Werten generieren. Das heisst, ein Dokument, welches aus 5 Wörtern besteht, wird mit einer Matrize mit dem Shape (5,5) repräsentiert.

```
from gensim.models import Word2Vec
import numpy as np
docs=[text.split() for text in corpus[:100]]
w2v_model = Word2Vec(docs, vector_size=5, min_count=1)
doc = 'move fluid flow propel cell'.split()
vectors = np.array([w2v_model.wv.get_vector(word) for word in doc])
print (vectors)
```

[2] Die meisten Dokumente beinhalten normalerweise eine sehr kleine Teilmenge der Wörter eines Korpus. In der Regel sind 99% der Werte der vektoriellen Repräsentation dieser Dokumente Nullen. Repräsentation dieser Dokumente beinhalten daher Werte für Wörter, die diese Dokumente nicht haben. Dementsprechend werden schnell für nicht allzu grosse Korpusse recht lange Vektoren entstehen, wofür der Speicherplatz und Zwischenspeicher bei der Verwendung berücksichtigt werden muss (scikit-learn, 2022b).

```
[[-0.10319845 -0.18262771  0.15827577 -0.12256435 -0.18999851]
 [ 0.03966755  0.01054277  0.03038067 -0.12689959 -0.1680822 ]
 [ 0.21738556  0.26883683  0.24592462 -0.07920758  0.16445838]
 [-0.03602478 -0.1619114   0.02348583  0.14331649 -0.06367458]
 [ 0.06074653 -0.09321033 -0.13082878  0.18563427 -0.03147945]]
```

5.2.1.3 Embedding from Language Model (ELMo)

ELMo ist eine weitere Methode zur Darstellung von Wörtern mittels Vektoren (Joshi, 2019). Diese Methode löst das Problem der Semantik bzw. der verschiedenen möglichen Bedeutungen eines Wortes. Sie kann den Kontext eines Wortes effektiv erfassen, indem es eine dynamische Repräsentation auf Wortebene liefert, welche die Kontextinformation enthält (En et al., 2019). Ein Wort kann somit unterschiedliche Vektoren haben, wenn es in unterschiedlichen Kontexten vorhanden ist. Beispielsweise kann der Vektor des Wortes «Apple» als Frucht anders sein als «Apple» als Marke (En et al., 2019). ELMo ist ein Modell, das auf einem neuronalen Netz mit zweischichtigem bidirectional language model (biLM) aufgebaut ist (Joshi, 2019). Es ist so aufgebaut, dass die Information über das Wort sowie über den Kontext der vorangehenden Wörter und der nachfolgenden Wörter erfasst werden und als Vektor bzw. Vektoren für das Wort zurückgegeben werden (Joshi, 2019). ELMo ist auf einem grossen Korpus von 5,5 Billionen Wörtern und auf einem kleineren Korpus von einer Billion Wörtern vortrainiert (Wei, 2020). Das Modell kann von *tensorflow-hub* heruntergeladen werden. Es nimmt einen Text oder eine Liste von Texten als Input und gibt einen Tensor zurück, welcher so viele Matrizen beinhaltet wie die Anzahl Texte in der Liste. Jede Matrize beinhaltet so viele Vektoren wie die Anzahl Wörter des grössten Texts in der Liste. Jeder Vektor beinhaltet genau 1024 Werte. Der untenstehende Code zeigt beispielhaft eine Liste mit drei Dokumenten. Das erste Dokument hat 4, das zweite 5 und das dritte 6 Wörter. ELMo wird als Output einen Tensor geben, welcher 3 Matrizen hat, jede mit 6 Vektoren und jeder Vektor mit 1024 Werten.

```
docs=['cilium like membrane protrusion',
'emanate surface vertebrate cell classify',
'motile cell primary cilium motile cilia']
import tensorflow_hub as hub
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)
elmo_vectors = elmo(docs, signature="default", as_dict=True)["elmo"]
print(elmo_vectors.shape)
```

```
(3, 6, 1024)
```

5.2.2 Klassifikationssysteme

In diesem Kapitel werden die vier ausgewählten Klassifikationssystemen beschrieben und erläutert, wie sie funktionieren.

5.2.2.1 Logistic Regression (LR)

Die logistische Regression (LR) ist ein Klassifizierungsverfahren, welches auf dem Konzept der Schätzung von Wahrscheinlichkeiten basiert. Für die Berechnung einer Wahrscheinlichkeit wird die Sigmoid-Funktion verwendet. Sigmoid ist eine mathematische Funktion, die beliebige Werte in Wahrscheinlichkeitswerte zwischen 0 und 1 konvertiert (Naeem, 2021). Das bedeutet für Klassifikationen, dass jede Klasse bzw. jede Kategorie durch einen Wahrscheinlichkeitswert repräsentiert wird. Die Kategorie mit dem höchsten Wahrscheinlichkeitswert wird als Vorhersage ausgegeben (Géron, 2019).

5.2.2.2 Multi Layer Perceptron Classifier (MLP)

Multi Layer Perceptron Classifier (MLP) ist ein Verfahren, das für die Klassifizierung von binären oder Multi-Klassen verwendet werden kann. Es handelt sich dabei um ein neuronales Netz, das im Fall der Multi-Klasse in der Output-Schicht die gleiche Anzahl Neuronen wie Klassen besitzt. Jede Klasse bekommt einen Wahrscheinlichkeitswert zwischen 0 und 1 (Géron, 2019); Die Summe aller Werte muss dann 1 ergeben (Brownlee, 2020; Géron, 2019). Die Werte werden mithilfe der Funktion «Sigmoid» berechnet (scikit-learn, 2022d) und die Klasse mit dem höchsten Wert wird als Vorhersage ausgegeben (Géron, 2019). MLP Classifier kann direkt von der Bibliothek *Scikit-learn* abgerufen und verwendet werden (scikit-learn, 2022d).

5.2.2.3 Convolutional Neural Network (CNN)

CNN ist ein neuronales Netz, welches versteckte Schichten, sogenannte Faltungsschichten besitzt. Jede dieser Schichten hat die Aufgabe, komplexere Muster in den Daten zu erkennen (Janakiev, 2018), was letztendlich dazu führen soll, gute Klassifikationsergebnisse zu erzielen (Maheshwari, 2018). Die Output-Schicht enthält analog zu MLP so viele Neuronen wie Klassen. Für die Auswahl der richtigen Klasse wird im Fall der Multi-Klasse Klassifikation in der Regel die "softmax" -Funktion verwendet. «Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector» (Brownlee, 2020). Jeder Wert im Vektor mit Wahrscheinlichkeiten repräsentiert eine Klasse. Die Summe aller Werte muss dann 1 ergeben (Brownlee, 2020; Géron, 2019). Die Klasse mit dem höchsten Wert wird als Vorhersage ausgegeben (Géron, 2019).

5.2.2.4 Recurrent Neural Network (RNN)

Bei Recurrent Neural Network (RNN) handelt es sich um Netzwerke mit aufeinander aufbauenden Schleifen von neuronalen Netzen (colah's blog, 2015). Die Ausgabe eines

neuronalen Netzes wird als Eingabe für das nächste verwendet (Amidi & Amidi, 2022). Ein bekannter Vertreter dieser Methode sind die Long Short Term Memory Networks (LSTM). LSTM vermeiden das Problem der praktischen Anwendung der RNN, mit Langzeit-Abhängigkeit umzugehen, indem sie Information über lange Zeit speichern können (colah's blog, 2015). Das ist ein wichtiger Grund dafür, dass sie oft für die Textklassifikation verwendet werden (Shekhar, 2021). Genauso wie bei CNN wird auch hier für die Auswahl der richtigen Klasse - im Fall von Multi-Klasse Klassifikation - für die Ausgabeschicht die Aktivierungsfunktion "softmax" verwendet. Auch hier wird die Klasse mit dem höchsten Wahrscheinlichkeitswert als Vorhersage ausgegeben.

5.3 Datenverarbeitung

Im Folgenden werden die Datengrundlage sowie die Aufbereitung der Daten Schritt für Schritt dargestellt.

5.3.1 Datengrundlage

Zur Durchführung dieser Analyse bzw. zum Trainieren der Klassifikationssysteme wurden gelabelte Daten benötigt, welche über die Plattform «Dimensions» bezogen werden konnten. Diese Daten sollten aus Publikationen extrahiert werden. Von jeder Publikation wurde das Abstract sowie der Titel des Journals, in dem sie publiziert wurde, extrahiert. Die Daten sollten von drei Fachgebieten stammen und von ihrem Umfang her ähnlich gross sein.

Unter Berücksichtigung dieser Kriterien wurden die Daten für diese Studie von der Plattform Dimensions via SQL-Abfrage erhoben. Die Abfrage von Publikationen wurde für jedes der drei Fachgebiete einzeln durchgeführt, um drei separate Datensätzen zu erhalten. Abgefragt wurden der Dimensions-Identifikator (ID), das Publikationsjahr, das Abstract und das publizierende Journal pro Publikation. Für die Analyse wurden nur die Abstracts der Publikationen als Input und die Titel der Journals als Labels verwendet. Für diese Analyse wurde die Verwendung von Volltexten ausgeschlossen, weil dabei sehr viel Abklärung betreffend Urheberrechte notwendig wäre (Pradhan & Pal, 2019; Safa et al., 2017). Um besser Ergebnisse zu erzielen, könnten als Input neben den Abstracts noch der Titel und die Keywords der Publikationen verwendet werden (Huynh et al. 2020). Das war im Rahmen dieser Studie nicht möglich, weil die Keywords der Publikationen nicht vorhanden sind und die Nutzung von Abstracts und Titeln in der Studie von Huynh et al. (2020) nicht viel bessere Ergebnisse lieferte als die alleinige Nutzung von Abstracts.

Aus Kapazitätsgründen wurden die Subdomänen der drei gewählten Fachgebiete PH, CH und BIO nicht berücksichtigt. Für CH und BIO wurden Publikationen zwischen den Jahren 2015 und 2021 abgefragt und für Physik zwischen 2010 und 2021, damit alle Datensätze eine ähnliche Anzahl von Publikationen aufweisen. Damit sollte verhindert werden, dass die Ergebnisse der Analyse vom quantitativen Umfang der Datengrundlage beeinflusst und fälschlicherweise der Eindruck entstehen würde, ein Empfehlungssystem für ein Fachgebiet funktioniere besser als bei anderen Fachgebieten. Es ist davon auszugehen, dass Empfehlungssysteme grundsätzlich besser arbeiten, je grösser die Datengrundlage ist (Adey, 2021). Die Verwendung von Datensätzen, die unterschiedliche Zeiträume abdecken, löst die Problematik der unterschiedlichen Grösse der Datenmenge. Es entsteht dadurch das Problem von unterschiedlich langen Publikationszeiträumen. Sie spielen eine Rolle für die Identifizierung von relevanten Journals, die in der vorliegenden Analyse berücksichtigt werden (vgl. Kapitel 5.3.2). Aufgrund der unterschiedlichen Länge der Zeiträume könnte bei PH im Vergleich zu CH und BIO ein höherer prozentualer Anteil von Journals als irrelevant identifiziert werden, was einen Einfluss auf die Qualität der untersuchten Klassifikationssysteme haben kann. Auf dieser Basis der getroffenen Selektionsentscheidung sind für die Analyse in dieser Arbeit vergleichbar grosse Datensätze an 2'027'302 Publikationen für PH, 1'981'776 für CH und 2'160’044 Publikationen für BIO erhoben worden.

5.3.2 Datenaufbereitung

Die Datenaufbereitung bzw. die Vorbereitung der Daten beinhaltete folgende Schritte

1. Entfernen von nicht vollständigen Beiträgen: Bei den exportierten Datensätzen fehlten zum Teil das Abstract oder die Angabe des Journals zu manchen Publikationen. Diese unvollständigen Datensätze wurden vom Datensatz entfernt. Für PH wurden 434'000, für CH 156'865 und für BIO 264'247 Datensätze entfernt.
2. Duplikate löschen: Mithilfe der ID des Artikels konnte geprüft werden, ob Duplikate vorhanden sind. Mit der Klasse Counter aus der Bibliothek collections wurde überprüft, ob es IDs gibt, die mehr als einmal vorkommen. Wäre das der Fall, hätten mithilfe des Publikationsjahrs die älteren Einträge gelöscht werden können. Es wurden jedoch keine Duplikate gefunden.
3. Löschung irrelevanter Publikationen: In einem weiteren Schritt wurden diejenigen Publikationen, die in für diese Studie als irrelevant definierten Journals publiziert sind, identifiziert und entfernt. Die Relevanz der Journals wurde auf zwei Kriterien festgelegt:

a. Das Journal sollte eine minimale Publikationshäufigkeit nicht unterschreiten. Die Untergrenze wurde so festgelegt, dass das Journal im Durchschnitt für den im Datensatz abgedeckten Zeitraum nicht weniger als 1.3 Publikationen pro Monat aufweisen darf. Fairbairn et al. (2014) haben in ihrer Analyse gezeigt, dass Journals im Median 4 Ausgaben pro Jahr publizieren. Ergänzt sei ferner, dass i.d.R. nur Journals mit einer höheren Anzahl von Artikeln als relevant betrachtet werden können3. Infolgedessen wurde der Studienumfang experimentell auf Journals mit mindestens 16 Artikeln pro Jahr (4 Artikel pro Ausgabe und 4 Ausgaben pro Jahr), d.h. mindestens 1.3 Artikel pro Monat im Durchschnitt, beschränkt. Um die Journals zu identifizieren, die dieses Kriterium erfüllen, musste zuerst berechnet werden, wie viele Publikationen jedes Journals mindestens haben müsste. Um diese Zahl für jeden Datensatz zu berechnen, wurde die Formel 1.3*12(Monate)*abgedeckte Jahre angewandt. Zum Beispiel bedeutet dies für das Fachgebiet PH mit einem Zeitraum von 2010-2021: 1.3*12*12 = 187.2.
b. Ein weiteres wichtiges Kriterium war, sicherzustellen, dass sich nur Publikationen von vertrauenswürdigen Journals im Datensatz befinden. Zur Beurteilung der Vertrauenswürdigkeit – bzw. deren Fehlen –, wurde auf die Beall's Liste abgestützt, weil diese regelmässig aktualisiert wird und eine gewisse Vollständigkeit in Bezug auf Verlage mit betrügerischen Geschäftspraktiken aufweist (Beall, 2021). Die betroffenen Journals sind auf einer Webseite gelistet. Sie wurden mit einem eigenen Skript extrahiert und in einer Excel-Datei gespeichert, damit sie einfacher verwendet werden kann.

Tabelle 3 bietet eine Übersicht über die vorgenommenen Schritte und die jeweiligen Auswirkungen auf die Gesamtzahl der Anzahl Journals, die nach Durchführung der Bereinigungsmassnahmen noch für die Analyse im Datensatz verblieben.

[3] Diese Auffassung teilt der Autor der Studie mit seinen Kolleginnen und Kollegen der Gruppe Knowledge Management der ETH-Bibliothek, welche an der ETH Zürich für bibliometrische Services zuständig ist.

	PH	**CH**	**BIO**
Alle Journals	9511	12527	19903
Journals, die das Kriterium 3a (Mindestpublikationshäufigkeit) nicht erfüllen *(Mindestzahl Gesamtpublikationen pro Journal)*	8920 *(187.2)*	11488 *(109.2)*	17768 *(109.2)*
Journals, die das Kriterium 3b (Vertrauenswürdigkeit) nicht erfüllen	0	0	0
Journals, die beide Kriterien erfüllen	**591**	**1039**	**2135**

Tabelle 3: Identifikation der relevanten und irrelevanten Journals und Anzahl ihrer Einträge

4. Sprachselektion: In dieser Analyse wurden nur Abstracts in englischer Sprache verwendet, weil nicht genügend anderssprachige Abstracts in Dimensions vorhanden sind. Ausserdem konnte so eine Vergleichbarkeit mit anderen Analysen gewährleistet werden, die ebenfalls ausschliesslich auf englischsprachigen Abstracts basieren. Bei der Identifikation und Entfernung nicht-englischsprachige Texte galten folgende Regeln:

a. War das Abstract vollständig in einer anderen Sprache verfasst, wurde der Beitrag aus dem Datensatz entfernt.
b. War das Abstract teilweise in einer anderen Sprache verfasst, wurde der englischsprachige Textteil beibehalten, der Rest entfernt.

Für diesen Schritt wurden die Bibliotheken Spacy und Spacy_langdetect verwendet. Zunächst wurde der grosse Korpus «en_core_web_lg» geladen und als nlp benannt. Anschliessend wurde mit dem Decorator @Language.factory eine Pipeline für den Sprachdetektor erstellt. Diese Pipeline wurde dann zu nlp hinzugefügt. Mithilfe von nlp wurde für jedes Abstract überprüft, ob dieses die Bezeichnung für die englische Sprache «en» erhielt. War das nicht der Fall, wurde der Eintrag bzw. das Abstract entfernt. Andernfalls wurde weiter analysiert, ob jeder Satz im Abstract auch in Englisch geschrieben war. Dies war notwendig, weil Spacy_langedetect den Text als pauschal betrachtete. Beinhaltete ein Text also Sätze, die in mehreren Sprachen geschrieben sind, würde jede vorkommende Sprache eine Punktzahl (score) erhalten und die Sprache mit der höchsten Punktzahl identifiziert. In den vorhandenen Datensätzen war dies nicht selten der Fall, denn eine Publikation konnte z.B. auf Deutsch geschrieben worden sein, das Abstract aber auf Deutsch und Englisch. Deshalb war es notwendig, die Sätze zu identifizieren, welche nicht in Englisch verfasst waren, und diese zu löschen. Folgendes Beispiel dient zur Illustration der Funktionsweise:

```
Text = '''Optical studies of crystals have been carried out. A significant
concentration effect of the Nd3+ ion on the crystal structure of these solid
solutions was identified. It was found from the absorption spectra of neodym-
ium ions in the transition region that at a concentration two nonequivalent
Nd3+ centers appear, which is explained by the formation of two polytype mod-
ifications with space groups R32 and C2/c. At a concentration only one modi-
fication with the space group R32 is observed.
появляются два неэквивалентных центра Nd 3+, что объясняется образованием
двух политипных модификаций с пространственными группами R32 и C2/c. При
концентрации x<0.6 наблюдается преимущественно только одна модификация с
пространственной группой R32. Ключевые слова: редкоземельные хромовые бораты,
кристаллическая структура, политипия, фурье-спектроскопия.'''
```

```
import spacy
from spacy_langdetect import LanguageDetector
from spacy.language import Language

nlp = spacy.load("en_core_web_lg")

@Language.factory('detector')
def create_detection(nlp: Language, name):
    return LanguageDetector()
nlp.add_pipe('detector')

doc = nlp(text)
print(doc._.language)
```

```
{'language': 'en', 'score': 0.8571397581617546}
```

```
for sent in doc.sents:
    if sent._.language['language'] != 'en':
        doc=str(doc).replace(str(sent), '')
print(doc)
```

```
Optical studies of crystals have been carried out. A significant concentra-
tion effect of the Nd3+ ion on the crystal structure of these solid solutions
was identifie. It was found from the absorption spectra of neodymium ions in
the transition region that at a concentration two nonequivalent Nd3+ centers
appear,which is explained by the formation of two polytype modifications with
space groups R32 and C2/c. At a concentration only one modification with the
space group R32 is observed.
```

Durch diesen Schritt wurden für PH 16’331, für CH 13’658 und für BIO 17’768 Einträge entfernt. Am Ende des Vorgangs wurde kontrolliert, ob alle Abstracts überprüft und richtig verarbeitet wurden. Wäre dies nicht der Fall gewesen wäre, gäbe es einen Fehlerhinweis. Dies war in der Verarbeitung aller Abstracts in den drei Datensätzen nicht der Fall. Die vollständige Durchführung dieses Schrittes sieht in Python wie folgend aus:

```
import spacy
from spacy_langdetect import LanguageDetector
from spacy.language import Language

nlp = spacy.load("en_core_web_lg")
@Language.factory('detector')
def create_detection(nlp: Language, name):
    return LanguageDetector()
nlp.add_pipe('detector')

def get_english_abstracts(dataframe):
    eng_abstracts = []
    for abstr in dataframe.abstract__preferred:
```

```
        doc = nlp(abstr)
        try:
            if doc._.language['language']=='en':
                for sent in doc.sents:
                    if sent._.language['language']!='en':
                        abstr = abstr.replace(str(sent), '')
                eng_abstracts.append(abstr)
            else:
                eng_abstracts.append(None)
        except:
            eng_abstracts.append(None)
    if len(eng_abstracts) == len(dataframe):
        dataframe['abstract'] = eng_abstracts
        df_eng = dataframe.dropna()
        data = {'df_after_remove_none_english':len(df_eng)}

        return df_eng, data
    else:
        print('Something went wrong. Count of English abstracts is not as
much as count of the rows of the data frame')
```

5. LaTeX-Konvertierung: Die Verwendung der Software LaTeX ist bei wissenschaftlichen Arbeiten verbreitet. Für eine korrekte Verarbeitung von mit dieser Software bearbeiteten Dokumenten waren für die vorliegende Studie zusätzliche Massnahmen erforderlich. Die LaTeX-Tags wurden identifiziert und entfernt. Die Abstracts, welche in LaTeX geschrieben wurden, wurden mit den LaTeX-Tags bzw. mit dem LaTeX-source-Modus für diese Abstracts von Dimensions ausgegeben. Es handelte sich um 25’599 Abstracts für PH, 4’118 für CH und 1’357 für BIO. Abstracts mit LaTeX-Tags sehen beispielsweise so aus:

```
abstract[4] = '''We present a coupled-channel analysis of the two Σπ\\docu-
mentclass[12pt]{minimal}\n\t\t\\usepackage{amsmath}\n\t\t\\usepack-
age{wasysym}\n\t\t\\usepackage{amsfonts}\n\t\t\\usepack-
age{amssymb}\n\t\t\\usepackage{amsbsy}\n\t\t\\usepack-
age{mathrsfs}\n\t\t\\usepackage{upgreek}\n\t\t\\setlength{\\oddsidemargin}{-
69pt}\n\t\t\\begin{document}$$\\varSigma \\pi $$\\end{document} charge states
and of K- p\\documentclass[12pt]

{minimal}\n\t\t\\usepackage{amsmath}\n\t

[…] Σ(1915)5/2+\\documentclass[12pt]{minimal}\n\t\t\\usepackage{ams-
math}\n\t\t\\usepackage{wasysym}\n\t\t\\usepackage{amsfonts}\n\t\t\\usepack-
age{amssymb}\n\t\t\\usepackage{amsbsy}\n\t\t\\usepack-
age{mathrsfs}\n\t\t\\usepackage{upgreek}\n\t\t\\setlength{\\oddsidemargin}{-
69pt}\n\t\t\\begin{document}

$$\\varSigma (1915)5/2^+$$\\end{document} are produced abundantly in the in-
termediate state. Cross sections for their production are evaluated.'''
```

Diese Texte beinhalteten wie abgebildet Spezialbegriffe wie z.B *usepackage, amsmath, upgreek, beginn und end*, welche im Schritt 6 *Normalisierung* nicht entfernt worden wären. Dementsprechend wären sie von den Feature-Engineerings

[4] Gekürzter Textteil wird mit […] dargestellt.

behandelt worden, als seien es inhaltliche Begriffe im Text selbst, obwohl das nicht der Fall war. Es war deswegen notwendig, eine Methode zu finden, welche in der Lage war, die LaTeX-Tags zu identifizieren und zu entfernen. Hierfür wurde mit der Python-Bibliothek *pylatexenc* gearbeitet. Diese Bibliothek bzw. die Klasse *LatexNodes2Text* kann ein Text von LaTeX-source-Modus (Input) zum normalen Text (Output) konvertieren.

Wäre der Input allerdings ein normaler Text, würde dieser manchmal unvollständig ausgegeben. Deshalb war es notwendig, zuerst herauszufinden, ob der Text tatsächlich im LaTeX-source-Modus war oder nicht. Wenn ja, konnte dieser von der oben genannten Klasse konvertiert werden. Wenn nein, dann wurde er ohne Konvertierung beibehalten. Für diese Aufgabe wurde für jeden Text mit der Klasse *LatexWalker* aus derselben oben genannten Python-Bibliothek überprüft, ob der Text im LaTeX-source-Modus vorhanden ist. Technisch bedeutet das, es wurde analysiert, ob der Text mehr als 20 *LatexNodes*[5] beinhaltet. Die Grenze von 20 LatexNodes wurde durch Tests festgelegt. Bei der Konvertierung wurden alle \t und \n als Text (nicht LaTeX-Tag) behandelt und daher als Teil des ausgegebenen Textes ausgegeben. Sie mussten noch separat entfernt werden. Im Folgenden werden die vorgängig beschriebenen Schritte exemplarisch mithilfe des oben eingegebenen Abstracts illustriert:

```
from pylatexenc.latex2text import LatexNodes2Text
from pylatexenc.latexwalker import LatexWalker
lw=LatexWalker(abstract)
ln=lw.get_latex_nodes()
print(len(ln[0]))
```

```
127
```

```
print(ln[0][:3])
```

```
[LatexCharsNode(parsing_state=<parsing state 3137586632784>, pos=0, len=51,
chars='We present a coupled-channel analysis of the two Σπ'), LatexMac-
roNode(parsing_state=<parsing state 3137586632784>, pos=51, len=30, mac-
roname='documentclass')]
LatexCharsNode(parsing_state=<parsing state 3137586632784>, pos=81, len=5,
chars='\n\t\t\t\t')
```

```
import re
lnt=LatexNodes2Text()
n_text = lnt.latex_to_text(abstract)
new_text = re.sub('[\t\n]','',n_text)
final_text = new_text.replace('  ','')
print(final_text)
```

[5] LatexNodes: Die Funktion get_latex_nodes von der Klasse LatexWalker gibt eine Liste zurück. Der erste Eintrag in dieser Liste sagt, wieviele Latex-Nodes der eingegebenen Text enthält. Diese Nodes sind entweder LatexCharsNode, welche für die normale Sätze ausgegeben wird oder LatexMacroNode, welche für die Latex-Tags ausgegeben wird. Ein normaler Text hat nur LatexCharsNodes.

```
We present a coupled-channel analysis of the two Σππ charge states and of K-
pK^-p pair, produced in the reaction γp→K+(Σπ)γ p→ K^+(π ), and in the reac-
tion γp→K+(Kp)γ p→ K^+(K^- p). Hyperon resonances in the mass range from
Λ(1405)1/2-(1405)1/2^- to the Σ(1915)5/2+(1915)5/2^+ are produced abundantly
in the intermediate state. Cross sections for their production are evaluated.
```

Die alten Texte wurden mit den neuen Texten ersetzt und kontrolliert, ob dieser Schritt korrekt und vollständig durchgeführt wurde. Es hätte einen Hinweis gegeben, falls ein Text mit LaTeX-Tags nicht mit einem konvertierten Text ersetzt worden wäre, was aber nicht der Fall war. Die vollständige Durchführung dieses Schrittes sieht in Python so aus:

```
from pylatexenc.latex2text import LatexNodes2Text
from pylatexenc.latexwalker import LatexWalker
import re
def remove_latex_tags(dataframe):
    dataframe = dataframe.dropna()
    #get indexes of the latex articles
    #latex abstract has usually more than 20 nodes
    latex_indexes = []
    for index,abstract in zip(dataframe.index, dataframe.abstract):
        lw = LatexWalker(abstract)
        ln = lw.get_latex_nodes()
        if len(ln[0]) > 20:
            latex_indexes.append(index)
    data = {'abstracts_in_latex':len(latex_indexes)}
    lnt = LatexNodes2Text()
    def remove_latex(text):
        n_text = lnt.latex_to_text(text)
        new_text = re.sub('[\t\n]','',n_text)
        final_text = new_text.replace('  ','')
        return final_text

    datafrme_copy = dataframe.copy()
    latex_not_replaced = []
    for ind in latex_indexes:
        try:
            datafrme_copy.abstract.loc[ind] = remove_latex(dataframe.ab-
stract.loc[ind])
        except:
            latex_not_replaced.append(ind)
    data['Count_latex_not_replaced'] = len(latex_not_replaced)
    if len(latex_not_replaced) != 0:
        data['indexes_latex_not_replaced'] = latex_not_replaced

    datafrme_copy = datafrme_copy.dropna()
    data['df_final'] = len(datafrme_copy)

    return datafrme_copy, data
```

6. Normalisierung: Bei der Durchführung dieses Schrittes wurde primär das Skript text_normalizer.py[6] verwendet (Sarkar, 2019). Der Autor dieser Arbeit hatte daran einige Anpassungen vorgenommen, damit das Skript für die vorliegenden Analyse

[6] text_normalizer.py

optimal eingesetzt werden konnte. Dieses angepasste Skript beinhaltete eine Pipeline von Funktionen, die zur Normalisierung eines Textes führen. Diese wurden in dieser Rheinfolge durchgeführt:

a. Entfernung von html-Tags: Dieser Schritt war notwendig, obwohl die Texte von nicht html-Seiten gecrawlt wurden. Einige Texte beinhalten Tags wie z.B. <\i>. Das «i» wäre nach der Entfernung der Sonderzeichen «<\>» beibehalten und als ein eigenes Wort behandeltworden. Das hätte das Feature-Engineering beeinträchtigt. Daher wurden solche Kombinationen (Sonderzeichen wie «<» und «>» und Zeichen wie z.B. «i» und «li») als HTML-Tags identifiziert und entfernt.
b. Konvertierung von Wörtern mit Kontraktionen: So wurde z.B. das Wort «won't» zu «will not» und «didn't» zu «did not» konvertiert. Dies wurde mithilfe einer Contraction-map[7], welche von Sarkar (2019) erstellt wurde, durchgeführt.
c. Lemmatisierung mithilfe der *Spacy* Bibliothek: In diesem Schritt wurden die Wörter auf ihre Grundform konvertiert (Sarkar, 2019). Es wurde beispielsweise das Wort «observations» zu «observation» konvertiert, wobei Wörter wie die Lemmatisierung vom Wort «intensely» ohne Änderung zurückgegeben werden.
d. Entfernung von Nicht-Buchstaben: In diesem Schritt werden alle Zeichen, welche nicht als Buchstabe identifiziert werden konnten, inklusive Zahlen, entfernt. So wurde beispielsweise «^6Li(3.563)» zu «Li» konvertiert.
e. Entfernung von Stoppwörtern und Konvertierung zur Kleinschreibung: In diesem Schritt wurden alle Wörter bzw. alle Buchstaben in Kleinschreibung konvertiert. Danach wurden alle Wörter, welche in der Liste der Bibliothek *nltk* als englische Stoppwörter vorhanden sind, identifiziert und entfernt. Diese Wörter kommen in der Regel in einem Text am häufigsten vor, haben aber wenig oder keine inhaltliche Bedeutung, besonders, wenn es um ein sinnvolles Feature-Engineering geht (Sarkar, 2019). In diesem Schritt wurden zudem alle Wörter bzw. *tokens*, die aus nur einen Buchstaben bestanden, wie z.B: «I», entfernt. Hintergrund dafür war, dass sie in der Regel für das Kontext-Verständnis nicht aussagekräftig sind. Im nächsten Beispiel wird illustriert, wie diese Wörter bzw. tokens eliminiert werden:

[7] contractions.py

```
text = """In this paper, we carry out an assessment of cosmic distance dual-
ity relation (CDDR) based on the latest observations of HII galaxies acting
as standard candles and ultra-compact structure in radio quasars acting as
standard rulers"""

from nltk.tokenize.toktok import ToktokTokenizer
tokenizer = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')
def remove_stopwords(text, is_lower_case=False, stopwords=stopword_list):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stop-
words and len(token)>1]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in
stopwords and len(token)>1]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
print(remove_stopwords(text))
```

```
paper carry assessment cosmic distance duality relation CDDR based latest ob-
servations HII galaxies acting standard candles ultra-compact structure radio
quasars acting standard rulers
```

Abschliessend wurden die lemmatisierten Texte bzw. Wörter noch durch eine Stemming-Funktion mithilfe der Klasse *PorterStemmer* von der Bibliothek *nltk* zum Wurzelstamm konvertiert. Stemming unterscheidet sich von der Lemmatisierung insofern, als dass es sich bei der Grundform eines Wortes bei der Lemmatisierung um den Wortstamm und beim Stemming um die Wortwurzel handelt (Sarakar, 2019). So wurden hier beispielsweise die Wörter «observations» zu «observ» und «intensely» zu «intens» konvertiert. Am Ende dieses Schrittes wurden die Texte zweimal zurückgegeben, einmal in Lemmatisierungsform und einmal in Stemmingform. Dieser Schritt war wichtig, weil in dieser Analyse getestet werden sollte, mit welcher Grundform bessere Ergebnisse geliefert werden können.

7. Encodierung der Titel der Journals bzw. der Labels: Durch die Encodierung wurde jedes Journal mit einer Zahl zwischen 1 und der Anzahl einzigartigen Titeln von Journals encodiert. Dieser Schritt war notwendig, weil einerseits Deep Learning bzw. neuronale Netze nicht mit kategorischen bzw. nicht nummerischen Daten umgehen können (da Costa & Cardoso, 2005; Garg, 2021), andererseits weil damit die Genauigkeit der trainierten Modelle für die Top-n (für den Fall, dass *n* nicht gleich eins ist) berechnet werden konnte[8].

   Die Labels wurden mithilfe der Klasse *LabelEncoder* von der Bibliothek *Scikit-learn* encodiert.

[8] Berechnung von top-n wird unter dem Kapitel 5.6 *Evaluation* genauer erläutert.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#add the encoded labels as a new column to the data frame
df_normalized['journal_code'] = le.fit_transform(df_normalized.journal)
```

8. Datenaufteilung und Anpassung der Grösse der Datensätze: Durch die oben beschriebenen Schritte 1-6 entstanden drei aufbereitete Datensätze, welche 1'501'899 Publikationen aus 590 Journals für PH, 1'710’940 Publikationen aus 1’038 Journals für CH und 1'694'512 Publikationen aus 2’133 Journals für BIO enthielten. Bei der Durchführung der Studie bzw. beim Trainieren der Modelle wurde festgestellt, dass im Rahmen dieser Analyse nicht mit allen Daten gearbeitet werden kann. Die vorhandene Hardware hatte nicht die erforderlichen Kapazitäten, in der zur Verfügung stehenden Zeit Modelle mit so grossen Datenmengen zu trainieren. Daher wurde entschieden, nur mit der Hälfte der Daten pro Datensatz zu arbeiten. Die Aufteilung der Daten wurde per Zufallsverfahren mit der Funktion *train_test_split* und dem Parameter *test_size=0.5* von der Bibliothek *Scikit-learn* durchgeführt. Der Parameter *stratify* wurde verwendet, damit alle im Datensatz vorhandenen Journals ähnlich aufgeteilt und in den beiden neu entstehenden Datensätzen möglichst in gleicher Anzahl vorkommen würden. Mit einem der beiden Datensätze – aufgrund der grossen Ähnlichkeit der beiden Datensätze war es nicht relevant, welcher verwendet wurde – wurde weitergearbeitet und die Trainings- und Testdaten erstellt. Das geschah mit derselben Logik, die auch zur Aufteilung auf zwei Datensätze durchgeführt angewendet wurde, ausser, dass der Parameter test_size=0.2 gesetzt wurde. So wurden die Daten auf 80% zum Trainieren und 20% zum Testen aufgeteilt. Die in dieser Form aufgeteilten Daten (x_train, x_test, y_train, y_test) wurden jeweils in einer Datei im Format *pickle* gespeichert. Der nachfolgende Code zeigt, wie dieser Schritt in Python für einen Datensatz (in diesem Beispiel für PH) implementiert wurde.

```
import pandas as pd
import numpy as np
import pickle
from sklearn.model_selection import train_test_split

df = pd.read_csv('../data/2_physics.csv').dropna()
df1_x, df2_x, df1_y, df2_y = train_test_split(np.array(df['cleaned_lemma']),
np.array(df['journal_code']), test_size=0.5, random_state=42, stratify=np.ar-
ray(df['journal_code']))

x_train, x_test, y_train, y_test =train_test_split(df1_x, df1_y,
test_size=0.2,random_state=42, stratify=df2_y)

with open('../data_1/x_train.pickle', 'wb') as xtr:
    pickle.dump(x_train, xtr)
# x_test, y_train und y_test sind analog wie x_train gespeichert
```

Als Ergebnis der Datenaufbereitung entstanden drei für die Verwendung vorbereitete Datensätze, die das folgende beinhalten:

		PH	CH	BIO
Abstracts	**Train**	600’749	684’259	677’687
	Test	150’188	171’065	169’422
	Summe	750’937	855’324	847’109
Journals	**Train**	590	1’038	2’133
	Test	590	1’038	2’132

Tabelle 4: Anzahl berücksichtigter Abstracts und Journals pro Fachgebiet

5.4 Modellaufbau und Fine-Tuning

Nach der Aufbereitung der Datensätze wurden die Features jedes Datensatzes mit den drei ausgewählten Feature-Engineerings extrahiert. Mit den jeweiligen extrahierten Features wurden die 4 ausgewählten Klassifikationssysteme trainiert und evaluiert. Das ergab für jeden Datensatz bzw. für jedes Fachgebiet 12 unterschiedliche Kombinationen von Feature-Engineerings und Klassifikationssystemen. Die folgende Abbildung zeigt diesen Prozess.

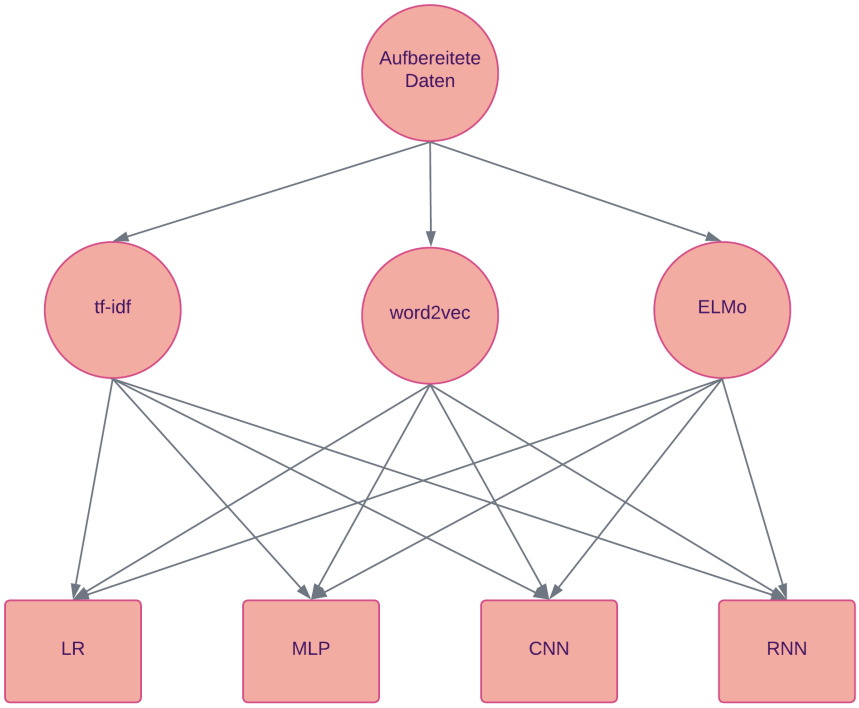

Abbildung 1: Durchführungsprozess für einen Datensatz

Bevor mit dem Training von Modellen angefangen werden konnte, musste zuerst herausgefunden werden, wie die Daten mit jedem Feature-Engineering für jedes Klassifikationssystem vorbereitet werden und mit welchen Parametern bzw. mit welcher Struktur jedes Modells/Klassifizierungssystems aufgebaut und trainiert werden sollte. Für diesen Zweck wurde in den meisten Fällen die Klasse *GridSearchCV* von *Scikit-learn* verwendet. Für die Modelle, die auf neuronalen Netzten basieren, wurde zuerst daran gearbeitet, die beste Struktur des neuronalen Netzes und dann in einem zweiten Schritt die beste Anzahl von Epochen von 5, 10, 15, 20 oder 30 herauszufinden. Aus den folgenden drei Gründen wurde die Anzahl Epochen auf maximal 30 Epochen beschränkt:

- Um prüfen zu können, wie gut die Ergebnisse sind, die ein neuronales Netz mit einer reduzierten Anzahl von Epochen liefern kann.
- Da das Training mit vielen Epochen mit so grossen Datenmengen mehr Zeit beansprucht, als zur Durchführung dieser Studie verfügbar war.
- Zur Vermeidung von Overfitting: Einer Gefahr, die bei der Verwendung einer hohen Zahl von Epochen bestehen kann.

Alle neuronalen Netze wurden mit der Bibliothek *tensorflow* aufgebaut und mit dem Optimizer *Adam* sowie mit der Loss-Funktion *categorical_crossentropy* kompiliert. Adam ist einer der meist verwendeten Optimizer für Natural Language Processing Analysen (Ruder, 2017). Eine andere Loss-Funktion wie bespielsweise die *Binary Cross entropy* ist nicht geeignet, weil sie sich besser für binäre Klassifikationen eignet.

Für das Fine-Tuning, also das Herausfinden der optimalen Parameter und der besten Struktur, wurde ein Sample von den Daten von PH entnommen und verarbeitet. Der Datensatz von PH war als Erstes verfügbar und daher früher als die anderen bereit zur Verwendung. Die Entnahme des Samples geschah mithilfe der Bibliothek *pandas* durch eine zufällige Auswahl von den Daten mit einem definierten Zufälligkeitsstatus. Der Sample-Datensatz beinhaltete 15'019 Dateneinträge, welche 575 unterschiedliche Journals abdecken. Diese Dateneinträge bilden 1% der gesamthaft vorhandenen Daten und 2% der am Ende der Datenaufbereitung im Datensatz verbleibenden Daten. 67% der Daten wurde für das Training und 33% für das Testing eingesetzt.

```
Import pandas as pd
df=pd.read_csv('../physik/data/2_physics.csv')
df_sample = df.sample(frac=0.01, random_state=42)
df_sample.to_csv('sample.csv')
```

Für das Fine-Tuning wurde ein Notebook Core i9-11950H, 2.60GHz, 96 GB RAM mit Grafikkarte Nvidia RTX A5000 16 GB verwendet.

Auf den nächsten Seiten wird beschrieben, wie das Fine-Tuning bzw. die Festlegung optimaler Parameter und der besten Struktur durchgeführt wurde, mit denen die Modelle später trainiert wurden. Ebenfalls wird erläutert, wie die verschiedenen Feature-Engineerings in Kombination mit Klassifikationssystemen eingesetzt wurden.

5.4.1 tf-idf

Im Folgenden wird erläutert, wie tf-idf in Kombination mit jedem der vier Klassifikationssysteme eingesetzt wird.

5.4.1.1 tf-idf - LR

In einem ersten Schritt wurde analysiert, ob für die besten Ergebnisse besser mit lemmatisierten oder Stemming-Daten gearbeitet werden soll. Für diesen Zweck wurden zwei Modelle mit LR trainiert, einmal mit lemmatisierten und einmal mit Stemming-Daten. Die Modelle wurden mit den Default-Parametern trainiert. Zur Veranschaulichung, wie dies genau geschah, dient beispielhaft der folgende Code für das Trainieren eines Modells mit lemmatisierten Daten:

```
import pandas as pd
df=pd.read_csv('../sample.csv')
from sklearn.model_selection import train_test_split
train_data, test_data, train_label, test_label = train_test_split(np.ar-
ray(df['cleaned_lemma']), np.array(df['j_code']), test_size=0.33, ran-
dom_state=42)

from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer() #Default Parameters
tv_train_features = tv.fit_transform(train_data)
tv_test_features = tv.transform(test_data)

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression() #Default Parameters
lr.fit(tv_train_features, train_label)
print('Score lemma lr:',lr.score(tv_test_features, test_label))
```

```
Score lemma lr: 0.18569065343258892
```

Das Modell, welches mit Stemming-Daten trainiert wurde, lieferte leicht bessere Ergebnisse (19%) als das Modell, welches mit lemmatisierten Daten trainiert wurde (18%). Für die Analyse wurden daher die Stemming-Daten bevorzugt.

In einem zweiten Schritt wurden die besten Werte für ausgewählte Parameter durch ein Hyperparameter-Tuning gesucht. Zu diesem Zweck wurden die Klassen *Pipeline* und *GridsearchCV* von *Scikit-learn* verwendet, um die beste Kombination der Parameter *ngram_range* von *TfidfVectorizer*, *C* und *max_iter* von LR zu ermitteln. *Crossvalidation* wurde auf 5 gesetzt. Das heisst, dass das Modell mit jeder Kombination dieser Parameter fünfmal trainiert und jedes Mal ein anderer Teil der Daten (20%) für die Validierung

verwendet wurde. Für eine erfolgreiche Durchführung sollte jede Klasse (Journal) mindestens 5 Einträge (Abstracts) beinhalten, damit jede Klasse in den Validierungsdaten mindestens einmal repräsentiert würde. Dieser Schritt musste vor der Aufteilung der Daten in Trainings- und Testdaten durchgeführt werden. Der folgende Code zeigt den gesamten Prozess.

```
import pandas as pd
import numpy as np
from collections import Counter
df = pd.read_csv('../sample.csv')
c = Counter(df.journal_code)
les_5 = [k for k,v in c.items() if v <5]
dff = df[~df.journal_code.isin(les_5)]

from sklearn.model_selection import train_test_split
train_data, test_data, train_label, test_label =
train_test_split(np.array(dff['cleaned_stem']), np.array(dff['jour-
nal_code']),test_size=0.33, random_state=42)

from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
lr_pipeline = Pipeline([('tfidf', TfidfVectorizer()),
                        ('lr', LogisticRegression(random_state=42))])
param_grid = {'tfidf__ngram_range': [(1, 1), (1, 2),(2,2),(2,3),(3,3)],
              'lr__C': [1, 5, 10],
              'lr__max_iter':[100,200,300]}
gs_lr = GridSearchCV(lr_pipeline, param_grid, cv=5, verbose=2)
gs_lr = gs_lr.fit(train_data, train_label)
```

ngram_range = (1,1), $C = 5$ und *max_iter* = 200 erwiesen sich als die besten estimators.

5.4.1.2 tf-idf - MLP

Analog wie für LR wurde hier im ersten Schritt untersucht, ob das Arbeiten mit lemmatisierten oder Stemming-Daten sinnvoller ist. Für diesen Schritt wurde die gleiche Herangehensweise wie für tf-idf - LR gewählt. Folgendes Beispiel zeigt mit lemmatisierten Daten die Durchführung mit MLP:

```
from sklearn.neural_network import MLPClassifier
mlp_lemma = MLPClassifier()
mlp_lemma.fit(tv_train_features, train_label)
print('score lemma mlp:',mlp_lemma.score(tv_test_features, test_label))
Score lemma mlp: 0.217328370554177
```

Auch hier haben die Ergebnisse gezeigt, dass mit Stemming-Daten leicht bessere Ergebnisse (21.9%) erzielt werden können als mit lemmatisierten Daten (21.7%).

Das Hyperparameter-Tuning wurde im Gegensatz zu LR nicht durchgeführt. Begründet ist das dadurch, dass MLP wegen seiner aufwändigeren und komplexeren Struktur lange Zeit zum Trainieren benötigt. Abbildung 2 zeigt dies mit einem konkreten Beispiel.

```
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
mlp_pipeline = Pipeline([('tfidf', TfidfVectorizer()),
                         ('mlp', MLPClassifier())
                        ])

param_grid = {'tfidf__ngram_range': [(1, 1), (1, 2), (2,2), (2,3)]}

gs_mlp = GridSearchCV(mlp_pipeline, param_grid, cv=5, verbose=2)
gs_mlp = gs_mlp.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] END ..........................tfidf__ngram_range=(1, 1); total time= 8.6min
[CV] END ..........................tfidf__ngram_range=(1, 1); total time= 8.6min
[CV] END ..........................tfidf__ngram_range=(1, 1); total time=76.5min
[CV] END ..........................tfidf__ngram_range=(1, 1); total time=85.2min
[CV] END ..........................tfidf__ngram_range=(1, 1); total time=641.7min
```

Abbildung 2: Laufzeiten von GridSearchCV für tf-idf - MLP

Abbildung 2 zeigt, dass das Berechnen des besten Werts für das Parameter *ngram_range* in Kombination mit den Standard-Einstellungen von MLP mehrere Tage in Anspruch nehmen könnte[9]. Zum Vergleich: Das Hyperparameter-Tuning für LR hat hingegen nur 10 Stunden und eine Minute gedauert. Aufgrund des hohen Zeitaufwands für das Hyperparameter-Tuning für MLP wurde entschieden, das im Rahmen der Studie nicht durchzuführen und stattdessen mit den Standard-Einstellungen bzw. den vordefinierten Werten der Parameter von MLP zu arbeiten. Das betraf aus demselben Grund nicht nur die Kombination mit tf-idf, sondern auch die Kombination mit anderen Feature-Engineerings (i.e. word2vec und ELMo).

5.4.1.3 tf-idf - CNN

tf-idf generiert, wie vorgängig beschrieben, lange Vektoren für jedes Dokument. Das kann ein Nachteil bei der Verwendung von neuronalen Netzen sein, denn es führt dazu, dass auf der Eingabeschicht so viele Neuronen vorhanden sein sollen wie die Anzahl Werte in jedem Vektor. Jeder dieser Neuronen wird mit allen Neuronen auf der nächsten Schicht eine Verbindung haben. Das erhöht schnell die Komplexität des neuronalen Netzes. Diese Art von Komplexität führt zu einem enormen Bedarf an Zwischenspeicher. Darüber hinaus können Daten in Form einer Sparse-Matrix nicht als Eingabe für neuronale Netze verwendet werden. Die Daten müssen stattdessen in Form von Arrays vorhanden sein und sie müssen daher zuerst konvertiert werden. In der Regel kann dies mit den Funktionen *toarray* oder *todense* erreicht werden. Dies führt, wie im 5.2.1.1 erwähnt, zu einer Matrize, welche so viele Vektoren wie die Anzahl Dokumente hat, und jeder Vektor hat Werte, die der Anzahl der einzigartig vorkommenden Wörter entspricht.

[9] Hinweis: In Abbildung 2 werden nur 5 Fits gezeigt. Beim 6. Fit hat der Autor den Prozess unterbrochen, weil dieses mehr als ein Tag gedauert hat.

Vor diesem Hintergrund wurde entschieden, dass für CNN in Kombination mit tf-idf die Stemming-Daten genutzt werden sollen. Die Anzahl der Werte pro Vektor basierend auf Stemming-Daten (27'700) ist deutlich geringer als die von lemmatisierten Daten (33'874). Damit sollte die Problematik der höhen Dimensionalität und die dadurch entstehende Problematik des Bedarfs an Zwischenspeicher reduziert werden.

Für die Ermittlung der bestmöglichen Struktur konnte hier aus Hardware-Kapazitätsgründen nicht mit *GridSearchCV* gearbeitet werden. Bei neuronalen Netzen wurde generell mit GPU gearbeitet. Dies weil das Trainieren von neuronalen Netzen mit CPU im Vergleich zur Verwendung von GPU viel mehr Zeit in Anspruch nimmt. Die GPUs haben aber weniger Zwischenspeicherkapazität. Deswegen war es notwendig, dass der GPU-Cache nach dem Testen jeder Struktur geleert wurde, bevor mit einer neuen Struktur getestet wurde. Aus diesem Grund wurde das Fine-Tuning hier manuell gemacht. Es wurden 10 CNN mit unterschiedlichen Strukturen aufgebaut und nur mit einer Epoche trainiert. Dann wurden diese getestet bzw. evaluiert und die Genauigkeit sowie die Loss-Werte zwischengespeichert. Die CNNs hatten eine unterschiedliche Anzahl von Faltungsschichten (2,3), Filters (32,64,128), Kernel_size (3,5,10), Maxpoolings (2,3), Pooling_size (3,4,5), Dropout (0,2), versteckte Schichten (1,2) sowie eine unterschiedliche Anzahl Neuronen auf den versteckten Schichten (256, 512). Die bestmögliche Struktur für ein CNN war jene, welche die höchste Genauigkeit lieferte.

Nach der Festlegung der bestmöglichen Struktur wurde die bestmögliche Anzahl von Epochen analysiert. Das geschah, indem die vorgängig beschrieben fünf Möglichkeiten (5-,10-,15- ,20- ,30-Epochen) mit der festgelegten Best-Struktur getestet wurden. Es wurde die Epochenzahl ausgewählt, die die höchste Genauigkeit aufwies.

Tabelle 5 bietet eine Zusammenfassung über die die bestmögliche Struktur inklusive der Anzahl Epochen.

Faltungssichten	Kernels	Kernel size	Maxpooling-Schichten	Pool size	Versteckte Schichten	Neuronen	Epochen
2	128, 64	10, 10	2	3, 3	1	256	20

Tabelle 5: Bestmögliche Struktur tf-idf - CNN

Die finale Struktur ist im folgenden Code abgebildet:

```
model=Sequential()
model.add(layers.Conv1D(128, kernel_size=10, in-
put_shape=(x_train.shape[1],1), activation='relu'))
model.add(layers.MaxPooling1D(pool_size=3))
model.add(layers.Conv1D(64, kernel_size=10, activation='relu'))
model.add(layers.MaxPooling1D(pool_size=3))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
model.fit(x_train.toarray(), y_train, epochs = 20, batch_size=32, valida-
tion_split=0.1)
val_loss, val_acc = model.evaluate(x_test.toarray(), y_test)
```

5.4.1.4 tf-idf - RNN

Für RNN wurde aus demselben Grund wie bei CNN mit den Stemming-Daten gearbeitet. Ebenfalls wurde auch hier die bestmögliche Struktur manuell bzw. nicht mit *GridSearchCV* herausgefunden. Es wurden 22 unterschiedliche RNN mit unterschiedlichen Strukturen aufgebaut. Das bedeutete eine unterschiedliche Anzahl von LSTM-Schichten (1,2,3), units (5,10,32,64,100,128,512,1028,2048,2056,3000), Dropout (0,.3), versteckten Schichten (0,1) und Anzahl Neuronen auf den versteckten Schichten (10,16,64,100,256,300,512). Auch hier galt, dass die bestmögliche Struktur diejenige war, welche die höchste Genauigkeit erzielte. Die beste Struktur wurde wiederum für das Herausfinden der bestmöglichen Anzahl von Epochen verwendet. In diesem Schritt sollte die Anzahl *units* wegen der beschränkten Hardware-Kapazität optimiert werden. Sobald die Anzahl Epochen erhöht wurde, stiess der Cache bzw. der Zwischenspeicher schnell an seine Grenzen und in der Folge wurde das Training unterbrochen. Die Anzahl *units* wurde daher auf 1024 festgelegt. Während der Durchführung wurde festgestellt, dass sich die Genauigkeit bei den unterschiedlichen Epochen nicht wirklich unterschied. Durch Definition des Parameters *learning_rate* beim Optimizer *Adam* auf den Wert 0.1 haben sich die Genauigkeiten bei der Erhöhung von Epochen dann erwartungsgemäss verhalten. Die aus dieser Analyse hervorgehende Aufbaustruktur für RNN ist in Tabelle 6 abgebildet.

LSTM	Units	Dropout	Versteckte Schichten	Neuronen	Epochen
1	1024	0.3	0	0	20

Tabelle 6: Bestmögliche Struktur tf-idf - RNN

Mit *tensorflow* sieht der Code für RNN so aus:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout,
model = Sequential()
model.add(LSTM(1024, input_shape=(1,x_train.shape[1])))
model.add(Dropout(0.3))
model.add(Dense(y_train.shape[1], activation='softmax'))
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accu-
racy'])
model.fit(x_train.toarray().reshape(-1,1,x_train.shape[1]), y_train,
epochs=epo, batch_size=64, validation_split=0.1)
val_loss, val_acc = model.evaluate(x_test.toarray().re-
shape(1,1,x_train.shape[1]), y_test, batch_size=256)
```

5.4.2 Word2vec

Bei der Verwendung von Word2vec als Feature-Engineering ist die Problematik der hohen Dimensionalität nicht so schwerwiegend wie bei tf-idf. Aus diesem Grund ist es nicht notwendig, die Anzahl einzigartiger Wörter durch die Verwendung von Stemming-Daten zu reduzieren. Hier wurde daher mit den lemmatisierten Daten gearbeitet, da Lemmatisierung mehr Sinn macht, wenn der Kontext des Worts berücksichtigt werden soll (Data Basecamp, 2022; Elia, 2020), was bei der Methode word2vec der Fall ist.

5.4.2.1 Datenvorbereitung für LR und MLP

Bevor auf das Fine-Tuning von LR bei der Nutzung mit word2vec eingegangen wird, soll zuerst gezeigt werden, wie die Trainings- und Testdaten dafür vorbereitet wurden. Wie bereits unter 5.2.1.2 beschrieben, generiert word2vec einen Vektor für jedes Wort in einem Dokument. Das bedeutet, dass ein Dokument mit 100 Wörtern mit einer Matrize von 100 Vektoren mit jeweils z.B. 100 Werten repräsentiert wird. LR sowie auch MLP erwarten aber, dass für ein Dokument nur ein Vektor eingegeben wird und nicht eine Matrize. Es mussten daher alle Vektoren eines Dokuments in einem einzigen Vektor repräsentiert bzw. zusammengeführt werden. Im Folgenden wird dieser Prozess mithilfe von dem in der Analyse verwendeten Code genauer erläutert.

Voraussetzung war, dass die Daten bereits in Trainings- und Testdaten aufgeteilt waren. Für das Trainieren vom word2vec-Modell, das Vektoren für jedes Wort ausgeben soll, wurden die Trainingsdaten verwendet. Sie sollten dem Modell in Form von einzelnen Wörtern pro Dokument hinzugefügt werden. Das heisst, dass pro Dokument eine Liste mit den einzelnen Wörtern erstellt werden musste. Das kann wie folgt erreicht werden:

```
X_train = [text.split() for text in train_data]
```

Die Liste wurde anschliessend dem Modell zum Trainieren übergeben. In diesem Schritt wurden die Standardeinstellung des Modells bzw. der Klasse *Word2vec* von der

Bibliothek *gensim* übernommen, ausser des Parameters *min_count.* Per Default war dieser auf 5 eingestellt. Das heisst, dass die Wörter berücksichtigt werden (und für sie ein Vektor generiert wird), die mindestens 5-mal in den Trainingsdaten vorkommen. Würde für diesen Parameter der Wert 5 übernommen, würden damit 67% der Wörter von den Trainingsdaten nicht berücksichtigt. Dieser Wert wurde daher auf 2 reduziert.

```
w2v_model = gensim.models.Word2Vec(X_train, vector_size=100, window=5,
min_count=2)
```

Das Modell *w2vec_model* generierte pro Wort einen Vektor mit 100 Werten. Damit der Kontext des Wortes erfasst werden konnte, wurde das Wort im Zusammenhang mit jedem einzelnen der 5 vorgelagerten und der 5 nachfolgenden Wörtern berücksichtigt (*window=5*). In einem weiteren Schritt wurden die Parameter vom Modell word2vec noch optimiert bzw. das Fine-Tuning durchgeführt.

Nach dem Trainieren des Word2vec-Modells mit den Trainingsdaten wurde das Modell für das Generieren der Vektoren für die Wörter verwendet. Der Code am Ende dieses Abschnittes zeigt die Umsetzung. Wie vorgängig beschrieben, wurde *min_count* auf 2 gesetzt. Das heisst, dass alle Wörter, die in den Trainingsdaten nur einmal vorkommen, nicht berücksichtigt und dementsprechend auch keine Vektoren mit diesem Modell generiert wurden. Aus diesem Grund wurde ein *set* von Wörtern erstellt, mit denen das Modell trainiert wurde. Dann wurde ein Dokument nach dem anderen iteriert und für jedes Wort, mit dem das Modell trainiert ist, ein Vektor generiert. Wörter, die das Modell nicht kannte, wurden ignoriert. Aus diesem Grund kann es sein, dass ein Dokument mit beispielsweise 100 Wörter nur durch 90 Vektoren repräsentiert wurde, weil die fehlenden 10 Wörter dem Modell nicht bekannt waren. Die für jedes Dokument generierte Matrize und der für jedes Wort generierter Vektor wurden als *numpy.array* gespeichert.

```
words = set(w2v_model.wv.index_to_key)
x_train_vect = np.array([np.array([w2v_model.wv.get_vector(word) for word in
doc if word in words]) for doc in X_train])
```

Ergebnis des obigen Codes war ein Tensor mit so vielen Matrizen wie die Anzahl Dokumente der Trainingsdaten. Jede Matrize beinhaltete so viele Vektoren, wie die Anzahl der vom Modell bekannten Wörtern aus diesem Dokument. Jeder Vektor beinhaltete 100 Werte.

In einem nächsten Schritt sollten die Vektoren jedes Dokuments in einen einzigen Vektor konvertiert werden, welcher das Dokument repräsentieren sollte. Dies wurde wie folgt umgesetzt:

```
x_train_vect_avg = []
for vectors in x_train_vect:
    x_train_vect_avg.append(vectors.mean(axis=1))
```

Mit einer Iteration wurde jedes Dokument bzw. jede Matrize abgerufen und die Durchschnittswerte der Vektoren berechnet. Die Berechnung erfolgte auf der Ebene *axis=1*. Diese Ebene kann man sich wie die Spalten einer Tabelle vorstellen. Der Ablauf ist wie folgt: Angenommen, ein Dokument (doc) besteht aus 3 Wörtern und jedes Wort wird mit einem Vektor (v) von 5 Werten repräsentiert. Ein Vektor mit Durchschnittswerten dieser 3 Vektoren wird wie folgt berechnet: doc = [(v1[0]+ v2[0]+ v3[0])/3, (v1[1]+ v2[1]+ v3[1])/3, …, (v1[4]+ v2[4]+ v3[4])/3].

```
doc = [[1,2,3,4,5],
       [2,3,4,5,6],
       [3,4,5,6,7]]
doc = [(1+2+3)/3,(2+3+4)/3,(3+4+5)/3,(4+5+6)/3,(5+6+7)/3]
doc = [2.0, 3.0, 4.0, 5.0, 6.0]
```

Durch diesen Schritt entstand ein Vektor, welcher zu einer Liste hinzugefügt wurde. Diese Liste beinhaltete am Ende so viele Vektoren wie die Anzahl Dokumente. Alle Vektoren beinhalteten dann die gleiche Anzahl Werte (100) und waren somit gleich lang. Diese Liste beinhaltete dann die Trainingsdaten, mit welchen die Klassifikationssysteme LR und MLP trainiert wurden. Die Testdaten wurden analog wie die Trainingsdaten vorbereitet.

Anhand der oben beschriebenen Schritte wurden die optimalen Parameter für das word2vec-Modell determiniert. Es wurden Kombinationen von verschiedenen Werten mancher Parameter getestet. Von jeder Kombination entstand ein Modell, mit welchem Trainings- und Testdaten vorbereitet und mit LR und MLP trainiert und getestet wurden. Am Ende wurde geprüft, mit welchen Werten der Parameter LR und MLP jeweils die höchste Genauigkeit erreicht hat. Diese Werte wurden dann bei der Durchführung der Analyse mit den kompletten Datensätzen verwendet. Der folgende Code zeigt, wie dieser Schritt realisiert wurde:

```
import gensim
import numpy as np
import pandas as pd
def word2vec(vec_size, window, min_count, sg, epoche):
    w2v_model = gensim.models.Word2Vec(X_train, vector_size=vec_size,   win-
dow=window, min_count=min_count, epochs=epo, sg=sg)

    words = set(w2v_model.wv.index_to_key)
    X_train_vect = np.array([np.array([w2v_model.wv.get_vector(word) for word
in doc if word in words]) for doc in X_train])
    X_text_vect = np.array([np.array([w2v_model.wv.get_vector(word) for word
in doc if word in words]) for doc in X_test])

    X_train_vect_avg = []
    for vectors in X_train_vect:
        if vectors.size:
            X_train_vect_avg.append(vectors.mean(axis=0))
    X_test_vect_avg = []
    for vectors in X_test_vect:
        if vectors.size:
            X_test_vect_avg.append(vectors.mean(axis=0))
        else:
            X_test_vect_avg.append(np.zeros(vec_size, dtype=float))
    lr = LogisticRegression(penalty='l2', max_iter=20, C=1, random_state=42)
    lr.fit(X_train_vect_avg, train_label)
    score_lr = lr.score(X_test_vect_avg, test_label)

    mlp = MLPClassifier(max_iter=10, random_state=42) #max_iter only 10 be-
cause it's slower
    mlp.fit(X_train_vect_avg, train_label)
    score_mlp = mlp.score(X_test_vect_avg, test_label)

    data={'Score_lr':round(score_lr,3), 'Score_mlp':round(score_mlp,3), 'Vec-
tor_size':vec_size, 'window':window, 'min_count':min_count,
'sg':sg ,'Epochs':epo}

    return data
vec_sizes = [100, 200, 300]
windows = [i for i in range(3,9)]
min_counts = [i for i in range(1,5)]
sgs = [0,1]
epochs = [5,10,20,30]
matrix_list = []
for vec in vec_sizes:
    for window in windows:
        for min_count in min_ counts:
            for sg in sgs:
                for epoch in epochs:
                    result = word2vec(vec, window, min_count, sg, epoch)
                    matrix_list.append(result)
matrix = pd.DataFrame(matrix_list)
#get index of the highest value of the first two columns (score_lr,
score_mlp)
index_best_score = matrix.idxmax(axis=0)[:2].tolist()
#print best parameters
Print(matrix.loc[index_best_score])
```

	Score_lr	Score_mlp	Vector_size	window	min_count	sg	Epochs
335	0.219	0.190	200	7	2	1	30
547	0.211	0.224	300	8	1	0	30

Der Parameter *sg*[10] bezieht sich darauf, ob das word2vec-Modell mit Continuous Bag of Words (CBOW) oder mit Skip-gram trainiert werden soll. Der Wert 0 bedeutet Trainieren mit CBOW und 1 für Skip-gram.

Mithilfe dieser Parameter wurden zwei word2vec-Modelle trainiert, mit den jeweiligen Trainings- und Testdaten vorbereitet und damit je einmal LR und MLP trainiert und getestet. Dieses Mal wurde der Parameter *max_iter* von LR und MLP auf 200 (Standardwert) erhöht - ausgehend von 20 für LR und 10 für MLP wie im obigen Code. Die Ergebnisse der Tests zeigten interessanterweise, dass die Parameter, mit denen LR die höchste Genauigkeit erreicht hat, die besten Ergebnisse für die beiden Klassifikationssysteme lieferten. Sie erreichten somit auch für MLP bessere Ergebnisse als diejenigen, die wie im oberen Output gezeigt wurde, für MLP die höchste Genauigkeit erzielten. Aus diesem Grund wurde entschieden, dass mit diesen Parametern (*vector_size=200, window=7, min_count=2, sg=1, epochs=30*) ein word2vec-Modell trainiert und gespeichert wird. Dieses wurde in dieser Phase der Analyse für die Vorbereitung von Trainings- und Testdaten verwendet, mit denen die Klassifikationssysteme trainiert und evaluiert wurden.

5.4.2.2 word2vec - LR

Nachdem festgelegt wurde, wie das Word2vec-Modell aufgebaut bzw. trainiert und wie dadurch Trainings- und Testdaten für LR und MLP vorbereitet würden, konnte daran gearbeitet werden, die besten Parameter für LR zu finden. Das wurde mithilfe von *GridSearchCV* analysiert. Wie bei tf-idf wurden auch hier Kombinationen zwischen Werten für den Parameter *C* (0.1,1, 5, 10) und 3 für *max_iter* (100,200,300) benutzt. Dies sieht in Python wie folgt aus:

[10] «In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. While in the Skip-gram model, the distributed representation of the input word is used to predict the context.» (Kulshrestha, 2019)

```
w2v_model = gensim.models.Word2Vec(X_train, vector_size=200, window=7,
min_count=2, epochs=30, sg=1)
'''
Preparation x_train and x_test
'''
lr_pipeline = Pipeline([('lr', LogisticRegression(random_state=42))])
param_grid = {'lr__C': [0.1,1, 5, 10],
              'lr__max_iter':[100,200,300],}
gs_lr = GridSearchCV(lr_pipeline, param_grid, cv=5, verbose=2)
gs_lr = gs_lr.fit(X_train_vect_1_avg, train_label)
```

Dieser Schritt hat ergeben, dass die besten estimators für LR mit word2vec *C* = 5 und *max_iter* = 100 sind.

5.4.2.3 word2vec - MLP

Für MLP wurde aufgrund der bereits unter Kapitel 5.4.1.2 *tf-idf - MLP* erläuterten Gründe das Hyperparameter-Tuning nicht durchgeführt, sondern stattdessen mit den Default-Einstellungen gearbeitet.

5.4.2.4 Datenvorbereitung für CNN und RNN

Bei der Nutzung von word2vec in neuronalen Netzen mussten die Daten anders vorbereitet werden als für die Nutzung mit LR und MLP. In den nächsten Zeilen wird dieser Prozess Schritt für Schritt mithilfe von Codes aus der Analyse erläutert.

Für die Datenvorbereitung wurde das bereits trainierte und gespeicherte word2vec-Modell verwendet, um Vektoren pro Wort zu generieren.

```
w2v_model=pd.read_pickle('w2vec_model')
X_train = [text.split() for text in train_data]
```

Dann wurde ein *dictionary* erstellt, in der die Wörter-Vektoren als *Key-Value Paare* gespeichert werden. Hier werden nicht alle Wörter von den Trainingsdaten übernommen, sondern nur die Wörter, die dem Modell bekannt sind.

```
Vocab = w2v_model.wv.key_to_index
Vocab = list(vocab.keys())
word_vec_dict = {}
for word in vocab:
    word_vec_dict[word] = w2v_model.wv.get_vector(word)
```

In einem nächsten Schritt wird die Klasse *Tokenizer* von *tensorflow* verwendet. Diese gibt jedem Wort von den Trainingsdaten einen Index bzw. eine nummerische Bezeichnung. Dieser Index fängt mit 1 an und nicht wie üblich mit 0. Mit der Funktion *text_to_sequences* wird ein Array generiert, der alle Dokumente repräsentiert. In diesem Array wird jedes Dokument mit einer Liste repräsentiert, welche die Indexe aller Wörter beinhaltet, die in diesem Dokument vorhanden sind. Im unteren Beispiel werden die Indexe der ersten 5 Wörter vom ersten Dokument ausgegeben.

```
from tensorflow.keras.preprocessing.text import Tokenizer
tok = Tokenizer()
tok.fit_on_texts(X_train)
train_lists = tok.texts_to_sequences(X_train)
print(X_train[0][:5])
print(train_lists[0][:5])
```

```
['short', 'focus', 'glass', 'lens', 'strong']
[395, 321, 450, 1222, 177]
```

Diese Listen bzw. Repräsentationen von Dokumenten werden als Eingabedaten für die erste Schicht der neuronalen Netze verwendet. Hier tritt das Problem auf, dass diese Listen nicht gleich lang sind, weil die Dokumente unterschiedlich viele Wörter beinhalten. Diese Problematik kann mit mithilfe der Funktion *pad_sequences* von *tensorflow* gelöst werden. Im *pad_sequences* soll definiert werden, wie lang die Listen sein sollen. Überschreitet die Länge einer Liste diese vordefinierte Länge in *pad_sequences*, werden alle Einträge der Liste nach dieser Zahl ignoriert und nicht übernommen. Erreicht die Länge einer Liste diese vordefinierte Länge nicht, wird die Liste mit Nullen ergänzt. In dieser Phase der Analyse wurde diese Zahl analog der Länge des längsten Textes in den Trainingsdaten auf 778 festgelegt.

```
maxi = max([len(i) for i in X_train]) #=>778
from tensorflow.keras.preprocessing.sequence import pad_sequences
x_train= pad_sequences(train_lists, maxlen=maxi, padding='post', truncat-
ing="post")
#padding (post)=> add zeros at the end of sequence,
#truncating(post)=>remove values larger than 778 at the end of the sequence
```

Die Testdaten werden analog wie folgt vorbereitet:

```
test_2se = tok.texts_to_sequences(X_test)
x_test= pad_sequences(test_2se, maxlen=maxi, padding='post', truncat-
ing="post")
```

Nachdem die Daten für die Eingabeschicht für die neuronalen Netzen vorbereitet sind, soll noch eine Embedding-matrix vorbereitet werden, welche für die Embedding-Layer notwendig ist. Es handelt sich hier um einen Array bzw. eine Matrize, die einen Eintrag bzw. einen Vektor mehr hat als die Anzahl einzigartige Indexe, welche vom *Tokenizer* generiert worden sind. Jeder Vektor in dieser Matrize hat so viele Werte wie die Anzahl Vektoren pro Wort (200). Am Anfang beinhalten diese Vektoren nur Nullen. Jeder Vektor wird dann mit dem Vektor ersetzt, welcher ein Wort repräsentiert und vom word2vec-Modell generiert worden ist. Die Ersetzung erfolgt auf Basis vom Index des Wortes, welcher vom *Tokenizer* generiert wurde. Repräsentiert ein Index ein Wort, das mit dem word2vec-Modell nicht trainiert wurde, wird der Array von diesem Index in der Liste/Matrize mit Nullen beibehalten. Vor diesem Hintergrund war es notwendig, dass die Länge dieser Matrize +1 ist, also ein Wert mehr als die Anzahl Indexes. Auf Position 0 dieser Matrize ist ein Vektor, welcher nur aus Nullen besteht. Das Vorgehen ist gewählt worden,

damit die Zuweisung auf Indexe der Einträge in dieser Matrize nicht falsch geschieht und mit Blick auf die Tatsache, dass Indexe vom *Tokenizer* mit 1 und nicht mit 0 beginnen.

```
embedding_matrix = np.zeros(shape=(len(tok.word_index) + 1, 200))
for word, i in tok.word_index.items():
    embed_vector = word_vec_dict.get(word)
    if embed_vector is not None:
        embedding_matrix[i] = embed_vector
```

Damit alle Daten, die für die CNN und RNN benötigt werden, vorbereitet sind, mussten die Labels (y_daten) vorbereitet werden. Damit die Multiklasse Klassifikation in *tensorflow* funktioniert, müssen die Label-Daten beispielsweise mit der Funktion *to_categorical* von *keras* zu *one hot encoding* konvertiert werden.

```
from keras.utils.np_utils import to_categorical
y_train=to_categorical(train_label)
y_test=to_categorical(test_label)
```

Erst jetzt kann das Hyperparameter-Tuning für CNN und RNN mit word2vec gemacht werden. Für diesen Zweck wurden die Klassen *KerasClassifier* und *GridSearchCV* benutzt.

5.4.2.5 word2vec - CNN

Hierzu wurde eine Funktion *create_model* geschrieben, durch die ein CNN-Modell aufgebaut wird. Die Parameter dieser Funktion sind die Elemente eines CNNs. Man kann pro Parameter eine Liste von Werten definieren, mit denen das Modell für dieses Parameter trainiert wird. Diese Funktion bzw. das dadurch entstehende Modell wird in *KerasClassifier* als Wert des Parameters *build_fn* definiert. In *KeraClassifier* sollte auch definiert werden, mit wie vielen Epochen das Modell trainiert wird, sowie auch die Grösse der Batches. Dieser wurde mit den unterschiedlichen Werten der Parameter von *create_model* durch *GridSearCV* trainiert. Der Wert der Crossvalidation wurde hier auf 5 gesetzt. *GridSearchCV* gibt dann die beste Kombination bzw. die beste Struktur aus, mit welcher CNN aufgebaut werden kann. Die bestmögliche Struktur ist diejenige, welche die höchste Genauigkeit erzielt. Dieser Schritt sieht so aus:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
def create_model(conv_layers, filters, kernel_size, pool_size, drop, hiddens,
neurons, optimizer):
    model=Sequential()
    model.add(layers.Embedding(input_dim=embedding_matrix.shape[0],
                      output_dim= embedding _matrix.shape[1],
                     weights=[embedding _matrix],
                      input_length=x_train.shape[1]))
    for i in range(conv_layers):
        model.add(layers.Conv1D(filters=filters, kernel_size=kernel_size, ac-
tivation='relu'))
        model.add(layers.MaxPooling1D(pool_size=pool_size))
        model.add(layers.Dropout(drop))
        filters=filters/2
    model.add(layers.Flatten())
    if hiddens !=0:
           for n in range(hiddens):
                model.add(layers.Dense(neurons, activation='relu'))
    model.add(layers.Dense(y_train.shape[1], activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, met-
rics=['accuracy'])
    return model

model=KerasClassifier(build_fn=create_model,epochs=1 ,batch_size=64)

conv_layers = [1,2,3,4]
filters = [64,128,256,512]
kernel_size = [3,5,10,15]
pool_size = [2,3,5]
drop = [0.0,0.2,0.3,0.5]
hiddens = [0,1,2,3,4,5]
neurons = [64,128,256,512]
optimizer = ['sgd','adam','rmsprop','adadelta','adamax']
param_grid = dict(conv_layers = conv_layers, filters=filters,
                  kernel_size=kernel_size,pool_size=pool_size,
                  drop=drop,hiddens=hiddens, neurons=neurons,
                  optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid ,n_jobs=1, cv=5)
grid_result = grid.fit(x_train, y_train)
```

Wegen beschränkter Hardware-Kapazität war es nicht möglich, alle Kombinationen von Parametern auf einmal zu testen. Der obige Code musste daher in mehreren Durchläufen ausgeführt werden, jedes Mal mit anderen Parametern. Zum Beispiel wurde zuerst eine bestimmte Anzahl *filters, kernel_size, pool_size und Dropout* festgelegt, dann die Anzahl Faltungsschichten, versteckter Schichten und Neuronen. In einem weiteren Schritt wurde die Einstellung des Optimizers festgelegt. Zuletzt wurde wegen der Auswahl von *learning_rate* bei dem Optimizer ein Lauf ausgeführt. Nach jedem Lauf musste der Cache geleert und eine neue Session gestartet werden.

Die durch diesen Schritt festgelegte Struktur als bestmögliche Struktur eines CNNs wurde als Basis zur Suche nach der besten Anzahl Epochen verwendet. Die Auswahl

der Anzahl Epochen erfolgte wie üblich durch eine Iteration mit einer *for* Schlaufe, mit den Möglichkeiten 5, 10, 15, 20, 30. Die Ergebnisse sind in der untenstehenden Tabelle 7 abgebildet.

Einbettungsschicht	1	Pool_size	2	Optimizer	Adam
Faltungsschicht	1	Dropout	0.2	Learning_rate	0
Filters	256	Flatten layers	1	Epochen	15
Kernel-size	3	Versteckte Schicht	1	Anteil Validation-Data von Trainingsdaten	10%
MaxPooling1D layer	1	Neuronen	256		

Tabelle 7: Bestmögliche Struktur word2vec - CNN

Die Einbettungsschicht wurde von diesem Schritt nicht beeinflusst, diese gibt es davon unabhängig nur einmal. Diese Schicht hat die Aufgabe, die eingegeben Daten, die Listen mit Indexen von Wörtern, entgegenzunehmen und für jedes Wort den Vektor auszugeben, welcher in der Embedding-Matrix die Indexposition des Vektors enthält, welcher dieses Wort repräsentiert. Für die vorbereiteten Daten in diesem Kapitel bedeutet dies beispielsweise, dass die Einbettungsschicht für jeden Eintrag einen Array mit 778 Werten bzw. Indexe entgegennimmt und eine Matrize mit dem Shape (778, 200) ausgibt.

5.4.2.6 word2vec - RNN

Das Herausfinden der bestmöglichen Struktur von RNN erfolgte ähnlich wie beim CNN. Auch hier wurde mit der Funktion *create_model* gearbeitet. Diese hatte hier jedoch andere Parameter und zwar solche zum Aufbau eines RNN Modells. Ebenfalls wurden für diesen Zweck *KerasClassifier* und *GridSearchCV* verwendet. Anders als bei CNN wurde dieser Schritt zweimal durchgeführt. Einmal mit *Reshape-Schicht* zwischen Embedding-Layer und LSTM-Layer und einmal ohne *Reshape*. Zuerst wurde ohne Reshape getestet. Dabei stellte sich jedoch heraus, dass die Modelle nicht lernten und die Genauigkeit über die Epochen sowie über die unterschiedlichen Kombinationen von Parametern hinweg bzw. trotz der unterschiedlichen Strukturen immer die gleichen Ergebnisse lieferten. Deswegen wurde dasselbe Vorgehen wiederholt, aber mit einer Reshape-Schicht. Diese Schicht konvertiert in dem hier verwendeten Beispiel eine Matrize mit der (778, 200) zu einem Vektor mit (1,778*2000). Durch diesen Schritt konnten Veränderungen der Genauigkeiten beobachtet werden, was als Hinweis darauf gewertet werden kann, dass die Modelle auf die unterschiedlichen Strukturen reagieren und die Genauigkeiten sowie die Loss-Werte dementsprechend unterschiedlich ausfallen.

```
def create_model(lstms, units, hiddens, neurons, drops):
    model=Sequential()
    model.add(layers.Embedding(input_dim=embedding_matrix.shape[0],
                      output_dim=embedding_matrix.shape[1],
                     weights=[embed_matrix],
                      input_length=x_train.shape[1]))   model.add(layers.Re-
shape((1,x_train.shape[1]*embedding_matrix.shape[1])))
    for i in range(lstms):
        if i != lstms-1:
            model.add(layers.LSTM(units, return_sequences=True))
            model.add(layers.Dropout(drops))
        else:
            model.add(layers.LSTM(units))
            model.add(layers.Dropout(drops))
    if hiddens != 0:
        for n in range(hiddens):
            model.add(layers.Dense(neurons, activation='relu'))
            model.add(layers.Dropout(drops))
    model.add(layers.Dense(y_train.shape[1], activation = 'softmax'))
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt, metrics=['accuracy'])
    return model
model=KerasClassifier(build_fn=create_model,epochs=1 ,batch_size=64)
lstms = [1,2,3,4]
units = [128,256,512,1028]
drops=[0.2,0.3,0.5]
hiddens=[0,1]
neurons=[64,128,256]
param_grid=dict(lstms=lstms, units=units, drops=drops, hiddens=hiddens, neu-
rons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid ,n_jobs=1, cv=5)
grid_result = grid.fit(x_train, y_train)
```

Wie bei CNN konnte aufgrund der beschränkten Hardware-Kapazität nicht alle Kombinationen von Parametern auf einmal getestet werden. Deswegen wurden aus der Auswahl an Parametern zuerst die ersten zwei Optionen getestet, z.B: *lstms* [1,2], *units* [128,256], etc. Die Option, welche die besseren Ergebnisse lieferte, wurde mit den weiteren Optionen in der Liste verglichen. War beispielsweise lstms=2 besser als lstms=1, dann würde in der nächsten Runde lstms [2,3] zum Testen verglichen.

Anders als bei CNN war die Genauigkeit der bestmöglichen Struktur, welche durch diese Herangehensweise herausgefunden wurde, nicht zufriedenstellend. Da dem Autor die Genauigkeiten unzureichend erschien, hat er basierend auf den gewonnenen Ergebnissen mithilfe von *KerasClassifier* und *GridSearchCV* manuelle Anpassungen vorgenommen. Das heisst, er hat die Struktur eines RNN, die von *GridSearchCV* als beste ausgewiesen wurde, übernommen und nochmals separat getestet und analysiert, aufgrund welcher Parameter Verbesserungen erzielt werden könnten. Tests haben gezeigt, dass das Problem (unzureichende Genauigkeit) nicht nur an der Struktur der RNN liegen könnte, sondern auch am word2vec-Modell selbst. Dieses Modell ist ursprünglich für LR und MLP trainiert worden und hat trotzdem zufriedenstellende Ergebnisse bei CNN

geliefert. Für RNN ist es aber notwendig, dass solche Modelle mit hoher Anzahl von Epochen trainiert werden. Es wurde daher ein neues word2vec-Modell trainiert. Dieses Mal wurde die Anzahl Epochen aber von ursprünglich 30 auf 100 erhöht. Die anderen Parameter blieben unverändert. Mithilfe dieses Modells und durch manuelle Anpassungen in der Struktur konnte RNN bessere Ergebnisse liefern.

Nun konnte dazu übergegangen werden, die bestmögliche Anzahl Epochen herauszufinden. Dies wurde genauso wie unter CNN beschrieben angegangen.

Die bestmögliche Struktur der RNN ist in Tabelle 8 ersichtlich.

Einbettungsschicht	1	Dropout	0.2	Epochen	5
Reshape-Schicht	1	Versteckte Schicht	0	Batch-Grosse	256
LSTM-Schicht	1	Optimizer	Adam	Anteil Validation-Data von Trainingsdaten	10%
Units	256	Learning_rate	0.01		

Tabelle 8: Bestmögliche Struktur word2vec - RNN

5.4.3 ELMo

Bei der Verwendung des Feature-Engineerings ELMo wurde aufgrund der unter Kapitel 5.4.2 für word2vec erwähnten Gründe entschieden, mit den lemmatisierten Daten zu arbeiten. Hintergrund war, dass besagte Gründe auch für ELMo zutreffen.

5.4.3.1 Datenvorbereitung für LR und MLP

ELMo nimmt üblicherweise eine Liste von Texten entgegen und generiert für jeden Text eine Matrize mit Vektoren entsprechend der Anzahl der vorhandenen Wörter im längsten Text in dieser Liste. Jeder Vektor hat 1024 Werte. LR und MLP können nicht mit Matrizen bzw. Listen von Vektoren für jedes Dokument umgehen. Sie erwarten nur einen Vektor für jedes Dokument. Es musste daher ähnlich wie bei word2vec von jeder Matrize mit beliebig vielen Vektoren ein einziger Vektor generiert werden. Dieses Ziel wurde mit derselben Logik wie bei word2vec erreicht. Es wurden also die Durchschnittswerte aus den Vektoren berechnet. Das geschah mit der vorgestellten Spaltenebene (axis=0 in *numpy* und axis=1 in *tensorflow*), damit die dadurch entstehenden Vektoren von allen Dokumenten gleich lang wurden. In den nächsten Zeilen wird dieser Prozess Schritt für Schritt und anhand konkreter Codes beschrieben.

Damit ELMo von *Tensorflow-hub* heruntergeladen werden konnte, musste die sogenannte *eager_execution* von *tensorflow* deaktiviert werden. Deswegen wurde tensorflow version 1.0 aktiviert.

```
import tensorflow.compat.v1 as tf  #To make tf 2.0 compatible with tf1.0 code
tf.disable_eager_execution()
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)
```

Es wurde dann eine Funktion[11] geschrieben, welche eine Liste von Texten übernahm und von ELMo verarbeitet wurde. ELMo generierte dann Vektoren für jeden Text und gab eine Liste aus, die die Repräsentationen aller Texte beinhaltete (so eine Liste sieht z.B. so aus: [doc_1[[v1],[v2],…,[vn]], doc_2[[v1],[v2],…,[vn]], … ,doc_n[[v1],[v2],…,[vn]]). Es wurde dann eine *tensorflow Session* erstellt, mit welcher ein Vektor pro Text generiert wurde und welche die Durchschnittswerte aller Vektoren dieses Textes repräsentierte. Die neu generierten Vektoren wurden dann als Array mit Listen ausgegeben (z.B: array([[doc_1],[doc_2],...,[doc_n]])). Dieser Array hatte das Shape (Anzahl Dokumente, 1024).

```
def elmo_vectors(text_list):
    embeddings = elmo(text_list, signature="default", as_dict=True)["elmo"]
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(tf.tables_initializer())
        return sess.run(tf.reduce_mean(embeddings, axis=1))
```

Aufgrund beschränkter Hardwarekapazität konnten nicht alle Texte auf einmal von dieser Funktion verarbeitet werden. Die Texte mussten daher Batch um Batch einzeln eingegeben werden. Die obige Funktion generierte für jedes Batch eine Liste. Die Liste musste in einer anderen Liste zwischengespeichert und am Ende mithilfe der Funktion *concatenate* von der Bibliothek *numpy* zusammengeführt werden.

```
#batch_size=40
list_train = [train_data[i:i+40] for i in range(0, train_data.shape[0], 40)]
elmo_train = [elmo_vectors(batch) for batch in list_train]
elmo_train_new = np.concatenate(elmo_train, axis=0)
print(len(train_data) == len(elmo_train_new)) #check that it went well
```

```
True
```

Die Testdaten konnten analog den Trainingsdaten vorbereitet werden.

5.4.3.2 ELMo - LR

Nachdem die Daten vorbereitet waren, konnte das Hyperparameter-Tuning für LR mithilfe von *GridSearchCV* gemacht werden. Es wurde analog wie für word2vec durchgeführt. Es zeigte sich, dass *C=5* und *max_iter =300* die besten estimators für das Trainieren von LR mit den von ELMo generierten Features sind.

[11] Der Code ist von Joshi (2019) übernommen bzw. inspiriert.

5.4.3.3 ELMo - MLP

Für MLP wurde, wie bereits erwähnt, auf das Hyperparameter-Tuning verzichtet und es wurde mit den Standardeinstellungen trainiert.

5.4.3.4 Datenvorbereitung für CNN und RNN

ELMo konnte mit nur wenig Vorbereitungsaufwand mit CNN und RNN kombiniert werden, weil ELMo als eine Einbettungsschicht direkt nach der Eingabeschicht zu einem CNN und RNN hinzugefügt werden kann. Die Eingabeschicht nimmt Texte entgegen, die mit ELMo durch eine Einbettungsschicht zu nummerischen Werten konvertiert und an die nächste Schicht weitergegeben werden. Die ELMo- bzw. Einbettungsschicht wurde hier aber nicht von *tensorflow.keras* erstellt, sondern von *tensorflow-hub*. In den nächsten Zeilen wird mithilfe von Code gezeigt, wie das realisiert wurde:

```
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.python.framework.ops import disable_eager_execution
disable_eager_execution()
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)
```

Hier musste eine Funktion geschrieben werden, welche einen Text entgegennimmt und diesen mit ELMo verarbeitet. Diese Funktion wurde dafür optimiert, sowohl einzelne Texte als auch Listen von Texten entgegenzunehmen und zu verarbeiten.

```
def ELMo_embedding(input_text):
    return elmo(tf.reshape(tf.cast(input_text, tf.string), [-1]), signa-
ture="default", as_dict=True)["elmo"]
```

Diese Funktion wurde dann durch die Funktion *Lambda* als Einbettungsschicht bei jedem Batch bzw. bei jedem eingegeben Text ausgeführt.

```
hub_layer = hub.KerasLayer(Lambda(ELMo_embedding, output_shape=(1024,)))
```

Dies sieht in einem CNN/RNN so aus:

```
model=Sequential()
model.add(Input(shape=(1,), dtype="string"))
model.add(hub_layer)
model.add(Conv1D())/model.add(LSTM())
 …
model.compile()
```

5.4.3.5 ELMo - CNN

Dieser Prozess erfolgt ähnlich wie im 5.4.2.5 *word2vec-CNN*. Im Unterschied dazu war hier aber kein Keras Embedding-Layer notwendig. Stattdessen wurde der *hub_layer* zur Einbettung verwendet. Zudem war hier eine Eingabeschicht notwendig. Abschliessend wurde mit dem Layer *GlobalAveragePooling1D* gearbeitet, um die Ausgabe der

Faltungsschicht(en), die wegen der *filters* zerlegt waren, zusammenzuführen, damit sie an die weitere Schicht übergeben werden konnten. Hier wurde ebenfalls mit *create_model, KerasClassifier* und *GridSearchCV* gearbeitet. Die Parameter und deren Werte, die hier getestet wurden, sind:

```
conv_layers = [1,2,3]
filters = [64,128,256,512]
kernel_size = [3,5,10]
pool_size = [2,3,5]
hiddens = [1,2]
neurons = [256]
```

Auch hier konnten aus Kapazitätsgründen nicht alle Kombinationen auf einmal getestet werden. Und wie in Kapitel 5.4.2.5 *word2vec-CNN* beschrieben, mussten die Kombinationen nacheinander getestet werden.

Mithilfe der besten Kombination wurde ein CNN aufgebaut und innerhalb einer Schlaufe verwendet, um die bestmögliche Anzahl Epochen herauszufinden. Die durch diesen Prozess gefundene bestmögliche Struktur und Anzahl Epochen ist in Tabelle 9 dargestellt.

Eingabeschicht	1	Pool_size	5	Learning_rate	0
KerasLayer für ELMo	1	Dropout	0.2	Epochen	10
Faltungsschicht	1	Flatten Layers	1	Anteil Validation-Data von Trainingsdaten	10%
Filters	128	Versteckte Schicht	2		
Kernel-size	3	Neuronen	256		
MaxPooling1D layer	1	Optimizer	Adam		

Tabelle 9: Bestmögliche Struktur ELMo - CNN

5.4.3.6 ELMo - RNN

Analog wie bei CNN kamen hier ebenfalls die *GridSearchCV, KerasClassifier* und die Funktion *create_model* zum Ansatz. Die Parameter, mit denen getestet wurde, sind:

```
lstms = [1,2,3]
units = [256,512,1028]
hiddens=[0,1,2]
neurons=[256]
```

Das Herausfinden der besten Anzahl Epochen erfolgte identisch wie bei CNN. Durch diesen Prozess konnte die bestmögliche Struktur für RNN festgelegt werden. Diese Struktur und die Anzahl Epochen sind in Tabelle 10 dargestellt:

Eingabeschicht	1	Versteckte Schicht	0	Batch-Grosse	32
KerasLayer für ELMo	1	Optimizer	Adam	Anteil Validation-Data von Trainingsdaten	10%
LSTM-Schicht	1	Learning_rate	0		
Units	256	Epochen	20		

Tabelle 10: Bestmögliche Struktur ELMo - RNN

5.5 Durchführung

In der letzten Phase der Analyse wurde festgelegt, wie jedes Klassifizierungssystem in Kombination mit jedem Feature-Engineering aufgebaut und trainiert werden kann. In der Phase der Durchführung wird nicht mehr nur ein Sample der Daten, sondern die Gesamtheit der Daten genutzt. Jede Kombination (Feature-Engineering - Klassifikationssystem) wurde drei Mal bzw. je einmal pro Fachbereich mit dem dazugehörigen Datensatz trainiert und evaluiert.

In dieser Phase wurden somit insgesamt 36 Modelle aufgebaut, trainiert und evaluiert. Für diesen Zweck wurden drei Maschinen verwendet:

- Workstation mit 250 GB RAM, welche hauptsächlich für das Training von LR und MLP verwendet wurde.
- Workstation mit drei Tesla (T4) GPUs (Graphics Processing Unit), jeweils mit 16 GB, welche hauptsächlich für das Training von tf-idf - CNN verwendet wurde.
- Ein Laptop mit 128 GB RAM und RTX (A5000) GPU mit 16 GB zum Training von allen anderen Modellen.

Während bei der Datenvorbereitung durch die drei Feature-Engineerings genauso vorgegangen werden konnte wie unter Kapitel 5.4 beschrieben, konnten beim Trainieren der Klassifikationssysteme in manchen Fällen die dort festgelegten besten Parameter, bestmöglichen Strukturen sowie Anzahl Epochen nicht vollständig übernommen werden. Im Folgenden wird bei jeder Kombination (Feature-Engineering - Klassifizierungssystem) beschrieben, welche Änderungen gegenüber der Vorbereitungsphase bei Bedarf vorgenommen wurden. Weiter wird auf die Gründe für diese Änderungen sowie die konkrete Umsetzung eingegangen.

5.5.1 Einschränkung der Textlänge

Vor der Durchführung wurde entschieden, dass die maximale Grösse der Texte für sämtliche Modellkombinationen auf 500 Wörter eingeschränkt wird. Diese Entscheidung wurde ausfolgenden Gründen getroffen:

- Die Länge der Texte sah in den drei Datensätzen wie folgt aus:

	1-100	**101-200**	**201-300**	**301-400**	**401-500**	**>500**
PH	362276	224757	12417	1046	163	90
PH%	0.603	0.374	0.021	0.002	0	0
CH	298193	368029	14719	2466	794	58
CH%	0.436	0.538	0.022	0.004	0.001	0
BIO	161899	473429	40670	1461	147	81
BIO%	0.239	0.699	0.06	0.002	0	0

Tabelle 11: Aufteilung der Textlänge getrennt nach Fachgebieten

In Tabelle 11 beziehen sich die Spaltennamen auf die Anzahl Wörter in einem Text. In den Zeilen PH, CH, BIO wird die Anzahl Dokumente angegeben, die so viele Wörter haben, wie im Spaltennamen angegeben ist. Die Zeilen PH%, CH%, BIO% zeigen den prozentualen Anteil der Dokumente mit bestimmten Textlängen am Gesamtkorpus der Trainingsdaten. Die Tabelle zeigt, dass die Anzahl Dokumente, welche aus mehr als 500 Wörtern bestehen, in jedem der Datensätze weniger als 0.000% betrug. Aus diesem Grund und um eine einheitliche Länge der Texte gewährleisten zu können, wurde die Zahl 500 als Einschränkung der Maximallänge der Texte festgelegt. Bei den Texten, die dieses Limit überschritten, wurden nur die ersten 500 Wörter berücksichtigt. Mit der Beschränkung auf die maximale Länge von 500 wurden mehr als 99.999% der Dokumente ohne Einschränkungen repräsentiert und weniger als 0.001% auf 500 Wörter reduziert.

- Die längsten Texte der Trainingsdaten beinhalteten 1’591 (PH), 1’281 (CH), 1’241 (BIO) Wörter. Mit Blick auf die Funktionsweise von word2vec und ELMo und unter Berücksichtigung der Länge der Texte, wie in Tabelle 11 gezeigt, musste bei deren Zulassung mit folgenden Problemen gerechnet werden:
  - Word2vec: Jedes Dokument wird mit einer Liste von Indexen von Wörtern repräsentiert. Jede Liste wäre genau so lang geworden wie der längste Text in den Daten, z.B. 1’591 Wörter im Fall von PH. Wie der Tabelle 11 entnommen

werden kann, wären mindestens 68% der Texte dieser Liste mit Nullen ausgefüllt worden. Diese Nullen repräsentieren immer den ersten Eintrag in der Embedding-Matrix, welche aus einem Vektor von 200 Nullen besteht und keine Wörter repräsentiert. Das bedeutet, dass die neuronalen Netze mehr als 68% der eingegebenen Werte nicht hätten verwenden können. Das hätte einerseits zu einer schlechten Qualität des Learnings führen können, und andererseits würden dadurch die neuronalen Netze komplexere Strukturen erhalten. Die Ausgabe der Embedding-Layer in diesem Beispiel hätte einen Tensor mit dem Shape (1591, 200) statt (500, 200) ergeben. Durch diese Komplexität würde sich der Bedarf an Zwischenspeicher sowie Zeit, die zum Trainieren in Anspruch genommen werden müsste, erhöhen.

  - ELMo: Für ELMo würde eine höhere Anzahl Wörter bedeuten, dass, wenn der ELMo-Einbettungsschicht ein Batch mit z.B. 32 Texten übergeben wird, in dem ein Text mit der Länge von 1591 Wörtern vorhanden ist, eine Ausgabe bzw. einen Tensor mit dem Shape (32, 1591, 1024) generiert wird. Dieser Tensor würde an die weitere Schicht (z.B. Faltungsschicht mit 128 Filters Kernel-Grösse 3) weitergeleitet. Diese Schicht generiert wiederum einen Tensor mit dem Shape (32, 512, 1591, 45). Solche grossen Tensors hätten nicht mit der zur Verfügung stehenden Hardware verarbeitet werden können, weshalb mit Unterbrüchen beim Trainieren hätte gerechnet werden müssen. Um das zu vermeiden, konnte entweder die Struktur des neuronalen Netzes angepasst oder die Länge der Texte eingeschränkt werden. Da es wenig Sinn machte, die bestmögliche Struktur der neuronalen Netze zu ändern, wurde die zweite Option, die Limitierung der maximalen Textlänge, vorgezogen.

- Aus Gründen der Einheitlichkeit wurde diese Einschränkung nicht nur im Fall von word2vec und ELMo umgesetzt, sondern auch in tf-idf. Diese Einschränkung wurde wie folgt implementiert:

```
x_train = np.array([' '.join(text.split()[:500]) for text in x_train])
x_test = np.array([' '.join(text.split()[:500]) for text in x_test])
```

5.5.2 tf-idf

Im Folgenden wird erläutert, wie die Klassifikationssysteme mit den mittels tf-idf extrahierten Features trainiert und evaluiert wurden.

5.5.2.1 tf-idf - LR

Bei diesem Klassifikationssystem wurden der Wert vom *solver* von «lbfgs» auf «saga» und der Wert von *n_jobs* von «None» auf «2» angepasst. Der *solver=saga* hat unter anderem den Vorteil, schneller bei grossen Datenmengen zu sein (scikit-learn, 2022a). *n_jobs* hat per Default den Wert None, womit das Training sequenziell (auf einem CPU-Kernel) und nicht parallel (auf mehreren CPU-Kernels) läuft. Mit *n_jobs=2* wird das Modell auf zwei CPU-Kernels gleichzeitig trainiert, was die benötigte Zeit zum Trainieren reduziert.

```
lr = LogisticRegression(C=5, max_iter=200, solver='saga', n_jobs=2)
lr.fit(x_train, y_train)
```

5.5.2.2 tf-idf - MLP

MLP wurde mit den Standard-Einstellungen trainiert. Dementsprechend war die Durchführung identisch mit jener, die in den vorangegangenen Kapiteln beschrieben wurde.

```
mlp = MLPClassifier()
mlp.fit(x_train, y_train)
```

5.5.2.3 tf-idf - CNN

Aufgrund der höheren Dimensionalität der von tf-idf generierten Daten waren Anpassungen notwendig. Das Feature-Engineering durch tf-idf hat für die Datensätze von PH, CH und BIO Vektoren pro Dokument mit einer Länge von 296’673, 486’561 und 606’601 erzeugt. Wegen dieser Längen war das Training von CNN mit Einsatz der durch das Hyperparameter-Tuning festgelegten bestmöglichen Struktur auf den vorhanden GPUs mit Ausnahme von PH nicht möglich. Es gab ständig Unterbrüche, trotz Verkleinerungen der Batches. Das Trainieren von neuronalen Netzen auf der CPU war im Rahmen dieser Studie aufgrund des begrenzten Zeitrahmens nicht möglich, da das Training auf CPUs wegen der grossen Datenmengen sehr viel Zeit in Anspruch nimmt. Daher wurde entschieden, die Anzahl *filters* von den Faltungsschichten zur Hälfte und die Anzahl Neuronen auf der versteckten Schicht auf einen Viertel zu reduzieren. So wurden die *filters* von 128, 64 auf 64, 32 und die Anzahl Neuronen von 256 auf 64 angepasst. So sieht die verwendete Struktur aus:

```
model=Sequential()
model.add(Conv1D(64, kernel_size=10, input_shape=(x_train.shape[1],1), acti-
vation='relu'))
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.2))
model.add(Conv1D(32, kernel_size=10, activation='relu'))
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
```

Wie bereits erläutert, wurden die von tf-idf generierten Daten in einer *Sparse matrix* von *Scipy* gespeichert. Daten in diesem Format werden von *tensorflow* als Eingabe für neuronale Netze nicht akzeptiert. Die Daten, die Sparse Matrix, mussten daher zu Arrays konvertiert werden. Dies kann mit der Funktion *toarray* oder *todense* gemacht werden. Das konnte aber nicht auf einmal für alle Trainingsdaten (x_daten) durchgeführt werden, denn dadurch würde z.B. für PH ein Array mit dem Shape (600749, 296673) entstehen. Um ein Array dieser Grösse zwischenzuspeichern, wird 1.3 TB RAM benötigt. Der folgende Code würde also nicht funktionieren.

```
model.fit(x_train.toarray(), y_train)
```

Um diese Problematik zu lösen, wurde mit einem Generator gearbeitet. Dem Generator wurden die Trainingsdaten, -Label und Batchgrösse als Parameter definiert. Er generiert anhand der definierten Grösse Batches, Trainingsdaten und -Label, mit welchen das Modell trainieren kann. Die generierten Daten haben das Shape (Batchgrosse, Länge der Vektoren (z.B.: 296673 für PH),1). Dieser Schritt wird in einer Iteration durchgeführt und wiederholt sich, bis das Modell mit allen Daten trainiert wurde. Wird das Modell in mehreren Epochen trainiert, wird im Generator der Prozess bei jeder neuen Epoche neu gestartet. Zum Trainieren mit dem Generator kann dieser dem Modell als Parameter die Trainingsdaten- und -Label übergeben. Der Parameter *steps_per_epoch* definiert dann, wie viele Runden bzw. Iterationen der Generator pro Epoche machen soll. Das Trainieren mit dem Generator[12] sieht so aus:

[12] Der Code ist von https://stackoverflow.com/questions/49834532/keras-generator inspiriert.

```
def batch_generator(X_data, y_data, batch_size):
    number_of_batches = X_data.shape[0]/batch_size
    counter=0
    shuffle_index = np.arange(np.shape(y_data)[0])
    np.random.shuffle(shuffle_index)
    X = X_data[shuffle_index, :]
    y = y_data[shuffle_index]
    while counter<number_of_batches:
        index_batch = shuffle_index[batch_size*counter:batch_size*(coun-
ter+1)]
        X_batch = X[index_batch,:].todense()
        y_batch = y[index_batch]
        counter += 1
        yield(np.array(X_batch, dtype='float32'),y_batch)
        if (counter >= number_of_batches):
            np.random.shuffle(shuffle_index)
            counter=0
train_generator = batch_generator(x_train, y_train, 16)
model.fit(train_generator, epochs = 10,
steps_per_epoch=(x_train.shape[0]/16))
```

Durch diesen Weg wurde der Zwischenspeicher nicht überfüllt und das Trainieren des Modells konnte problemlos durchgeführt werden. Anders als beim Trainieren ohne Generator konnte während des Trainierens mit Generator nicht validiert werden. Zu diesem Zweck wurden deshalb die drei Parameter *validation_split*, *validation_steps* und *validation_batch_size* von der Funktion *fit* getestet. Da jedoch auch mit keinem von ihnen erfolgreich validiert werden konnte, wurde entschieden, ohne Validierung zu trainieren. Für die Evaluation der Genauigkeit wurden dem Modell die Testdaten ebenso wie bereits beim Trainieren durch einen Generator übergeben.

5.5.2.4 tf-idf RNN

Der Verlauf bei RNN entsprach grösstenteils jenem bei CNN. Hier musste die Anzahl *units* wegen der höheren Dimensionalität reduziert werden. Die Anzahl *units* wurde von 1024 auf 256 für PH und CH und auf 200 für BIO reduziert.

```
Model=Sequential()
model.add(LSTM(256, input_shape=(1, x_train.shape[1])))
model.add(Dropout(0.3))
model.add(Dense(y_train.shape[1], activation='softmax'))
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['ac-
curacy'])
```

Auch hier wurde aus demselben Grund wie bei CNN mit einem Generator gearbeitet. Dieser hatte dieselbe Funktionsweise wie für CNN, mit einer Ausnahme: die generierten Daten hatten das Shape (Batchgrosse, 1, Länge der Vektoren). Dieses Shape war notwendig, damit die eingegebenen Daten von der Schicht LSTM akzeptiert und verarbeitet werden konnten. Dies wurde im Generator wie folgt realisiert.

```
Def batch_generator(x_data, y_data, batch_size):
    '''some code'''
    while counter<number_of_batches:
        '''some code'''
        yield(np.array(x_batch, dtype='float32').reshape(-
1,1,x_data.shape[1]), y_batch)
        '''some code'''
train_generator = batch_generator(x_train, y_train, 16)
model.fit(train_generator, epochs = 10,
steps_per_epoch=(x_train.shape[0]/16))
```

Die Anzahl Epochen wurde hier anderes definiert, als für die bestmögliche Anzahl Epochen festgelegt: Die Anzahl Epochen wurde im Vergleich zur für die beste Genauigkeit eruierten Zahl von 20 auf 10 reduziert. Zum einen aus Zeitgründen, weil die Verarbeitungszeit mit einer Epochenzahl von 20 den für diese Arbeit verfügbaren Zeitrahmen mit der verfügbaren Hardware gesprengt hätte. Zum anderen, weil durch die Reduktion der Epochen, getestet mit dem Sample-Datensatz, die Genauigkeit für top-1 nur minimal abnahm: von 21.9% (Epochen=20) auf 21.1% Genauigkeit (Epochen=10).

5.5.3 Word2vec

Im Folgenden wird erläutert, wie die Klassifikationssysteme mit den mittels word2vec extrahierten Features trainiert und evaluiert wurden.

5.5.3.1 word2vec - LR

Die Durchführung konnte genauso wie unter tf-idf - LR beschrieben umgesetzt werden, ausser, dass die Anzahl maximaler Iterationen, gemäss Erkenntnissen aus dem Fine-Tuning, auf 100 definiert wurde.

```
lr = LogisticRegression(C=5, max_iter=100, solver='saga', n_jobs=2)
lr.fit(x_train, y_train)
```

5.5.3.2 word2vec - MLP

MLP wurde wie bereits erwähnt mit den Default-Einstellungen trainiert.

```
mlp = MLPClassifier()
mlp.fit(x_train, y_train)
```

5.5.3.3 word2vec - CNN

In Kombination mit word2vec wurde CNN wie geplant aufgebaut und mit der festgelegten bestmöglichen Struktur und Anzahl Epochen trainiert.

```
model=Sequential()
model.add(Embedding(input_dim=embedding_matrix.shape[0],
                    output_dim=embedding_matrix.shape[1],
                   weights=[embedding_matrix],
                    input_length=x_train.shape[1]))
model.add(Conv1D(filters=256, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten()) #put all maxpoolings in one array/layer
model.add(Dense(256, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
model.fit(x_train, y_train, epochs = 15, batch_size=32, validation_split=0.1)
```

5.5.3.4 word2vec - RNN

Wie bei CNN wurden auch hier keine Änderungen in der Struktur vorgenommen. Die Anzahl Epochen wurde hier jedoch angepasst. Als die bestmögliche Anzahl Epochen wurde 5 ermittelt. Dies schien wenig im Vergleich mit der Anzahl Epochen, mit denen alle anderen neuronalen Netze trainiert wurden. Deswegen wurde RNN mit dem ersten Datensatz (PH) testweise einmal mit 5 und einmal mit 10 Epochen trainiert. Die Ergebnisse mit 10 Epochen waren besser (Top-1: 20.3% - Top-20: 67.1%) als mit 5 Epochen (Top-1:17.6% - Top-20: 61.4%). Daher wurde entschieden, RNN mit den von word2vec vorbereiteten Daten mit 10 Epochen zu trainieren. Die Batchgrösse spielte hier eine Rolle, denn es konnte beobachtet werden, dass die Genauigkeit mit sinkender Batchgrösse abnahm. Die Batchgrösse wurde aus diesem Grund von 265 auf 2048 erhöht.

```
model=Sequential()
model.add(Embedding(input_dim=embedding_matrix.shape[0],
                    output_dim=embedding_matrix.shape[1],
                   weights=[embedding_matrix],
                    input_length=x_train.shape[1]))
model.add(Reshape((1,x_train.shape[1]*embedding_matrix.shape[1])))
model.add(LSTM(512))
model.add(Dropout(0.2))
model.add(Dense(y_train.shape[1], activation = 'softmax'))
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=2048, epochs=10, validation_split=0.1)
```

5.5.4 ELMo

Im Folgenden wird erläutert, wie die Klassifikationssysteme mit den mittels ELMo extrahierten Features trainiert und evaluiert wurden.

5.5.4.1 ELMo - LR

Es wurde genauso vorgegangen, wie unter tf-idf - LR (vgl. Kapitel 5.5.2.1) beschrieben, ausser dass die Anzahl maximaler Iterationen, gemäss Erkenntnissen aus dem Fine-Tuning, auf 300 gesetzt wurde.

```
Lr = LogisticRegression(C=5,max_iter=300, solver='saga', n_jobs=2)
lr.fit(x_train, y_train)
```

5.5.4.2 ELMo - MLP

Hier wurde mit den Default-Einstellungen trainiert.

```
mlp = MLPClassifier()
mlp.fit(x_train, y_train)
```

5.5.4.3 ELMo - CNN

CNN konnte in Kombination mit ELMo genau so aufgebaut und trainiert werden, wie die festgelegte bestmögliche Struktur und Anzahl Epochen es vorsahen. Es wurde also wie folgt ausgeführt:

```
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)
def ELMoEmbedding(input_text):
    return elmo(tf.reshape(tf.cast(input_text, tf.string), [-1]), signa-
ture="default", as_dict=True)["elmo"]
hub_layer = hub.KerasLayer(Lambda(ELMoEmbedding, output_shape=(1024,)))
```

```
model = Sequential()
model.add(Input(shape=(1,), dtype="string"))
model.add(hub_layer)
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=5))
model.add(Dropout(0.2))
#put global average of all maxpoolings in one array/layer
model.add(GlobalAveragePooling1D())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
model.fit(x_train,y_train, epochs=10, batch_size=32, validation_split=0.1)
```

5.5.4.4 ELMo - RNN

Ähnlich wie CNN konnte RNN genau so aufgebaut und trainiert werden wie die festgelegte bestmögliche Struktur. Die Anzahl Epochen wurde hier aufgrund der zum Trainieren benötigten Zeit von 20 auf 10 Epochen reduziert. Die Batchgrösse konnte dank der verfügbaren Kapazität im Zwischenspeicher von 32 auf 36 erhöht werden. Damit konnte das Training schneller durchgeführt werden.

```
model=Sequential()
model.add(Input(shape=(1,), dtype="string"))
model.add(hub_layer)
model.add(LSTM(256))
model.add(Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=36, validation_split=0.1)
```

5.6 Evaluation

Für die Evaluation der Qualität jedes Modells wurde die Metrik «Genauigkeit» (Anzahl richtige Vorhersagen / Anzahl aller Vorhersagen) verwendet. Diese wurde in Anlehnung an bestehende Forschung für Top-1, Top-3, Top-5, Top-10, Top-15 und Top-20 berechnet. Wenn das Target-Journal in den Top-n vorhergesagt wird, bedeutet dies eine korrekte Vorhersage. In dieser Studie wurde dies mithilfe der Funktion *top_k_accuracy_score* von der Bibliothek *Scikit-learn* berechnet. Diese Funktion nimmt die Test-Labels, die vorhergesagten Labels und die Anzahl *top_k/top_n* als Eingabe. Die Test-Labels müssen als nummerische Werte in einem einzigen Vektor (1D-Array) vorhanden sein. Dafür war die Label-Encodierung der Journals notwendig. Die vorhergesagten Labels müssen als Array vorhanden sein, in welchem für jeden Eintrag ein Vektor existiert. Jeder dieser Vektoren besteht in der Regel aus so vielen Werten, wie es einzigartige Labels in den Trainings-Labels gibt. Diese Werte repräsentieren jeweils die Wahrscheinlichkeit einer Klasse, dass sie die richtige Klasse ist. Die Positionen der höchsten *n* Wahrscheinlichkeitswerte in einem Vektor werden mit dem Test-Label verglichen. Wenn das Test-Label identisch mit einer dieser Positionen ist, gilt dies als korrekte Vorhersage. Hier ein Beispiel zur Veranschaulichung, wie diese Funktion eine Berechnung für Top-3 für nur einen Eintrag macht: Angenommen, es sind Journals A,B,..,K als Train-Labels vorhanden. Diese wurden in Folge mit 0,2,…,9 encodiert. Ein Modell, welches mit den codierten Labels trainiert ist, macht für einen Eintrag mit Test-Label=3 folgende Vorhersage [0.15,0.06,0.07,0.12,0.05,0.09,0.27,0.09,0.07,0.03]. Die Positionen der höchsten drei Wahrscheinlichkeitswerte sind [6,0,3]. Das Test-Label (3) liegt also in den höchsten drei Werten. Das bedeutet, dass das Modell für Top-3 für diesen Eintrag richtig vorhergesagt hat.

Wenn die Anzahl einzigartiger Labels in Test-Labels nicht identisch mit der Anzahl in Train-Labels ist, muss für diese Funktion der Parameter *labels* definiert werden. Dieser sollte die höchste Zahl der kodierten Labels enthalten. In der Regel entspricht dieser der Länge der Vektoren, die als Vorhersage vom Modell ausgegeben werden.

Die mithilfe dieser Funktion berechneten Genauigkeiten für jede Kombination (Feature-Engineering - Klassifikationssystem) wurden in einer Json-Datei als *dictionary* gespeichert. Für jeden Datensatz bzw. für jedes Fachgebiet wurde eine eigene Json-Datei erstellt. In dieser Analyse wurde diese Funktion wie folgt verwendet:

```
prediction = model.predict(x_test) or
prediction = model.predict_proba(x_test)
scores = {}
from sklearn.metrics import top_k_accuracy_score
for top in [1,3,5,10,15,20]:
   s = top_k_accuracy_score(y_test, prediction, k=top, labels=(range(predic-
tion.shape[1])))
   scores[f'Top-{top}'] = s
import json
with open('results.json','w') as w_file:
    json.dump(score, w_file, indent=4)
```

6 Ergebnisse

Im Rahmen dieser Studie wurden 12 Kombinationen zwischen drei Feature-Engineerings und 4 Klassifikationssystemen für jedes der ausgewählten Fachgebiete durchgeführt. Für jede Kombination gab es folglich drei Modelle. Es wurden dementsprechend für die drei Fachgebiete insgesamt 36 Modelle trainiert. Diese Modelle sind nach ihren Genauigkeiten für die Top-n (1,3,5,12,15,20) evaluiert worden. Zur Beantwortung der Forschungsfragen 1, 2 und 3 wurden die Genauigkeiten der Modelle jeder Kombination von den 3 Fachgebieten auf Basis des Mittelwerts aggregiert. So wurde z.B. die Kombination ELMo - LR wie folgt aggregiert: ELMo - LR(aggregiert) = (ELMo - LR(PH)+ ELMo - LR(CH)+ ELMo - LR(BIO))/3. Dies wurde für alle Top-n gemacht.

Zur Beantwortung der Forschungsfrage 4 wurden die Ergebnisse der unterschiedlichen Fachgebiete miteinander verglichen, um herauszufinden, welche Ähnlichkeiten oder Unterschiede es zwischen den Fachgebieten gibt und ob diese einen Einfluss auf die Qualität bzw. Genauigkeiten der Modelle haben.

- F1: Welches Feature-Engineering erzielt die beste Qualität für ein Empfehlungssystem für Journals

Mit dieser Frage sollte herausgefunden werden, ob eine allgemeingültige Aussage darüber gemacht werden kann, ob und welches der hier in dieser Analyse verwendeten Feature-Engineerings immer die bessere Qualität erreicht. Eine solche Aussage kann nicht gemacht werden, denn für die Qualität bzw. die Genauigkeit spielt neben dem Feature-Engineering auch das Klassifikationssystem eine Rolle. Zur Abbildung der vorhandenen Tendenzen wurde jeweils der Mittelwert für jedes Feature-Engineering für alle top-n berechnet. Für ELMo Top-1 bedeutete dies z.B. den Durchschnittswert aller aggregierten Genauigkeiten für ELMo in Kombination mit den vier Klassifikationssystemen, unter Betrachtung von Top-1. Die Ergebnisse dieser Berechnung sind in der Abbildung 3 aufgeführt.

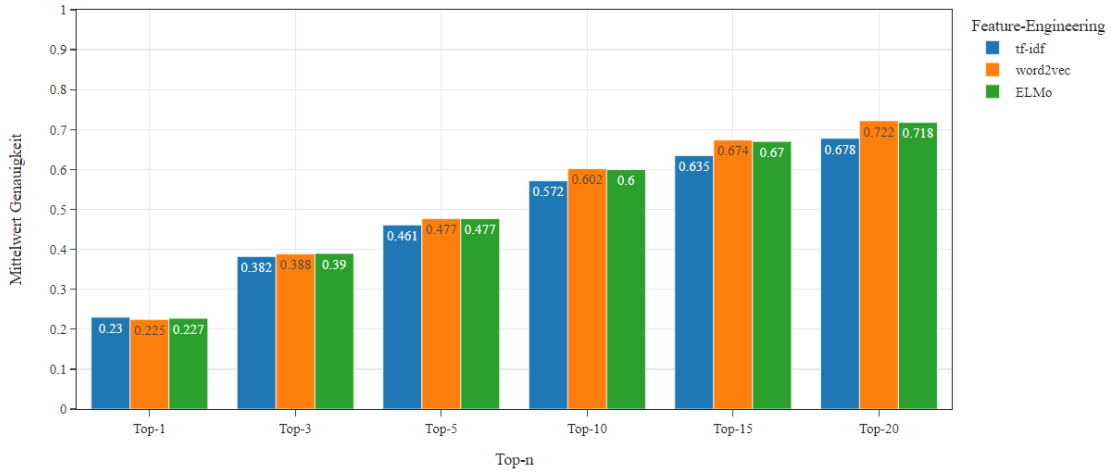

Abbildung 3: Mittelwerte der Genauigkeiten der Feature-Engineerings

Abbildung 3 zeigt, dass für Top-1 und Top-3 alle drei Feature-Engineerings ähnliche Leistungen mit minimalem Unterschied und vergleichbarer Qualität liefern. Je grösser aber die Anzahl Top-n wird, desto mehr Qualität liefern ELMo und word2vec. Man sieht in Abbildung 3 auch, dass die beiden Modelle ab Top-5 eine ähnlich gute Qualität liefern, während die Genauigkeit von tf-idf abnimmt.

- F2: Auf Basis welches Klassifikationssystems kann ein Empfehlungssystems die beste Qualität erzielen?

Analog zur ersten Forschungsfrage war hier das Ziel herauszufinden, ob es ein Klassifikationssystem gibt, welches unabhängig vom Feature-Engineering generell eine bessere Qualität liefert als die anderen. Abbildung 4 zeigt, dass LR in Kombination mit den unterschiedlichen Feature-Engineerings bei der Berücksichtigung von allen Top-n im Durchschnitt die höchste Qualität liefern kann, gefolgt von MLP mit Ausnahme von Top-1, wo MLP und RNN die gleiche Genauigkeit erzielt haben.

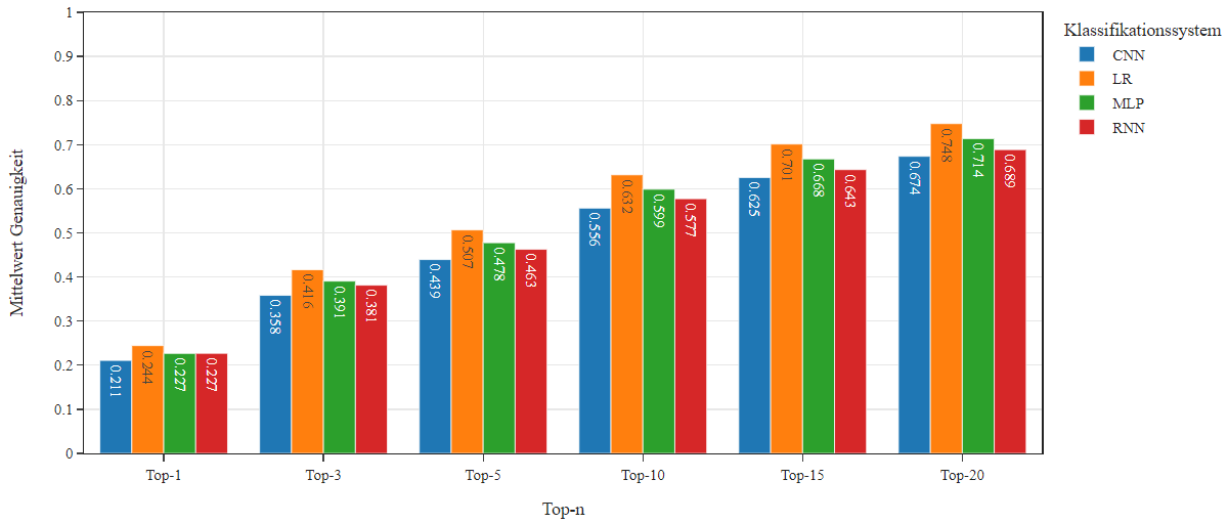

Abbildung 4: Mittelwerte der Genauigkeiten der Klassifikationssysteme

- F3: Mit welcher Kombination von Feature-Engineering und Klassifikationssystem kann ein Empfehlungssystem die beste Qualität erzielen?

Tabelle 12 zeigt für jede Kombination die Genauigkeit und das Ranking, wobei sich der Ranking-Wert 1 auf die höchste und 12 auf die niedrigste Genauigkeit bezieht, welche eine Kombination für ein Top-n erzielt hat. Der Tabelle kann entnommen werden, dass sich die Kombinationen tf-idf - LR für alle Top-n auf Platz 1 und word2vec - MLP auf Platz 2 durchsetzen. Auch die meisten anderen Modellkombinationen bleiben sich ihrem Ranking in der Tendenz über die meisten Top-n hinweg treu, mit nur leichten Abweichungen. Davon ausgenommen sind tf-idf - CNN und tf-idf - RNN.

	Top-1		**Top-3**		**Top-5**		**Top-10**		**Top-15**		**Top-20**	
	Acc	Ran	Acc	Ran	Acc	Ran	Acc	Ran	Acc	Ran	Acc	Ran
tf-idf_LR	0.285	1	0.468	1	0.56	1	0.682	1	0.747	1	0.79	1
tf-idf_MLP	0.207	10	0.351	10	0.428	10	0.535	10	0.597	10	0.64	11
tf-idf_CNN	0.205	11	0.348	11	0.425	11	0.538	9	0.607	9	0.654	9
tf-idf_RNN	0.223	7	0.361	9	0.431	9	0.53	11	0.588	12	0.628	12
word2vec_LR	0.235	4	0.413	3	0.509	3	0.642	3	0.716	3	0.765	3
word2vec_MLP	0.247	2	0.43	2	0.525	2	0.657	2	0.729	2	0.776	2
word2vec_CNN	0.198	12	0.334	12	0.411	12	0.524	12	0.593	11	0.641	10
word2vec_RNN	0.217	8	0.376	7	0.463	7	0.585	7	0.656	7	0.705	7
ELMo_LR	0.213	9	0.368	8	0.451	8	0.571	8	0.641	8	0.688	8
ELMo_MLP	0.226	6	0.391	6	0.48	6	0.605	6	0.676	6	0.725	6
ELMo_CNN	0.23	5	0.393	5	0.482	5	0.606	5	0.677	5	0.725	5
ELMo_RNN	0.241	3	0.407	4	0.495	4	0.617	4	0.686	4	0.733	4

Tabelle 12: Genauigkeit (Acc) und Ranking (Ran) aller Kombinationen für alle Top-n

- F4: Inwiefern ändert sich die Qualität eines Empfehlungssystems für Journals durch die Verwendung von Datensätzen aus unterschiedlichen Fachgebieten?

Abbildung 5 zeigt, dass die Qualität eines Modells sich von Fachgebiet zu Fachgebiet ändern kann. Die durchschnittliche Genauigkeit für jedes Modell für PH ist höher als bei CH und BIO. Die meisten Modelle weisen bei CH und BIO eine vergleichbare Qualität auf, wobei tf-idf - CNN, tf-idf - RNN und ELMo - RNN von dieser Beobachtung auszunehmen sind. Die Qualität der Modelle für ein Fachgebiet bleibt jeweils konstant höher oder niedriger im Vergleich zur Qualität bei den anderen Fachgebieten, es gibt keine Interaktionseffekte.

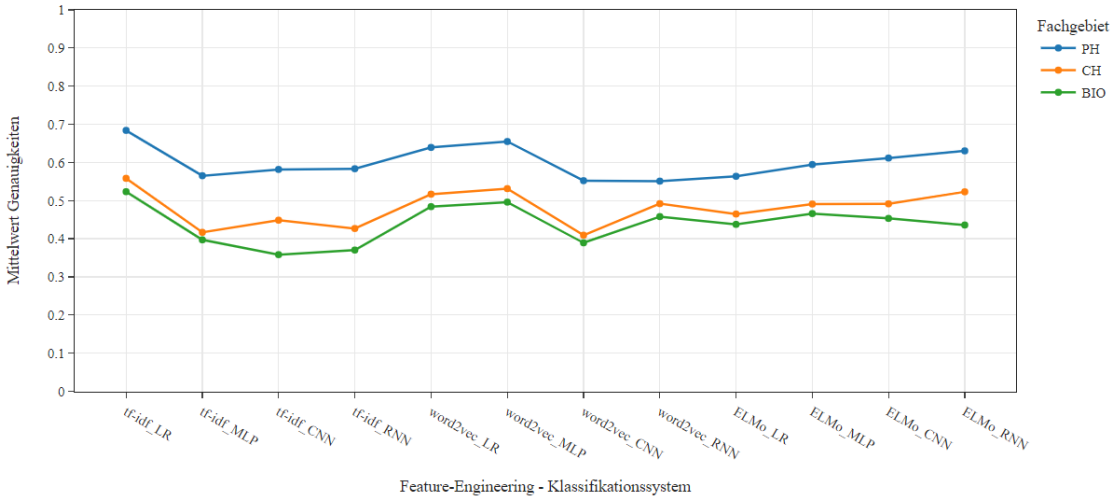

Abbildung 5: Mittelwerte der Genauigkeiten für alle Top-n pro Kombination und Fachgebiet

Tabelle 13 zeigt die Qualität der Modelle für die unterschiedlichen Top-n im Zusammenhang mit den Fachgebieten.

		Top-1		Top-3		Top-5		Top-10		Top-15		Top-20	
Kombination	**FG**	**Acc**	**Ran**	**Acc**	**Ran**	**Acc**	**Ran**	**Acc**	**Ran**	**Acc**	**Ran**	**Acc**	**Ran**
tf-idf_LR	**PH**	0.367	1	0.58	1	0.672	1	0.782	1	0.835	1	0.868	1
	CH	0.249	1	0.426	1	0.522	1	0.654	1	0.726	1	0.773	1
	BIO	0.239	1	0.399	1	0.485	1	0.609	1	0.68	1	0.728	1
tf-idf_MLP	**PH**	0.275	9	0.453	9	0.54	9	0.654	9	0.715	10	0.754	11
	CH	0.171	11	0.304	11	0.38	11	0.49	11	0.555	11	0.601	12
	BIO	0.175	10	0.297	9	0.363	9	0.462	9	0.522	9	0.564	10
tf-idf_CNN	**PH**	0.292	7	0.468	8	0.554	8	0.668	7	0.733	7	0.775	7
	CH	0.181	10	0.323	9	0.405	9	0.527	9	0.602	9	0.654	9
	BIO	0.142	12	0.252	12	0.317	12	0.42	12	0.485	11	0.534	11
tf-idf_RNN	**PH**	0.307	4	0.484	6	0.563	7	0.666	8	0.722	8	0.759	9
	CH	0.186	9	0.318	10	0.391	10	0.499	10	0.561	10	0.606	10
	BIO	0.175	9	0.282	10	0.339	11	0.426	11	0.48	12	0.52	12
word2vec_LR	**PH**	0.304	5	0.519	3	0.619	3	0.742	3	0.806	3	0.846	3
	CH	0.203	4	0.374	4	0.472	4	0.613	4	0.693	3	0.746	3
	BIO	0.198	4	0.347	3	0.436	3	0.571	3	0.65	3	0.703	3
word2vec_MLP	**PH**	0.322	2	0.538	2	0.637	2	0.758	2	0.818	2	0.856	2
	CH	0.214	3	0.389	2	0.488	2	0.629	2	0.708	2	0.76	2
	BIO	0.206	2	0.362	2	0.451	2	0.583	2	0.661	2	0.713	2
word2vec_CNN	**PH**	0.269	10	0.437	11	0.522	11	0.637	12	0.702	12	0.746	12
	CH	0.162	12	0.288	12	0.365	12	0.482	12	0.554	12	0.605	11
	BIO	0.163	11	0.278	11	0.347	10	0.454	10	0.522	10	0.572	9

word2vec_RNN	**PH**	0.254	12	0.43	12	0.519	12	0.639	11	0.709	11	0.755	10
	CH	0.2	5	0.36	5	0.451	5	0.581	6	0.655	7	0.705	7
	BIO	0.198	5	0.338	5	0.418	5	0.535	5	0.605	5	0.654	5
ELMo_LR	**PH**	0.264	11	0.449	10	0.537	10	0.654	10	0.718	9	0.76	8
	CH	0.188	8	0.334	8	0.421	8	0.548	8	0.623	8	0.675	8
	BIO	0.188	8	0.32	8	0.396	7	0.511	7	0.581	7	0.63	7
ELMo_MLP	**PH**	0.281	8	0.473	7	0.568	6	0.69	6	0.756	6	0.799	6
	CH	0.198	6	0.356	6	0.447	6	0.579	7	0.657	6	0.709	6
	BIO	0.199	3	0.344	4	0.425	4	0.545	4	0.616	4	0.667	4
ELMo_CNN	**PH**	0.296	6	0.493	5	0.586	5	0.708	5	0.772	5	0.814	5
	CH	0.196	7	0.353	7	0.446	7	0.581	5	0.66	5	0.713	5
	BIO	0.197	6	0.334	6	0.413	6	0.528	6	0.599	6	0.649	6
ELMo_RNN	**PH**	0.313	3	0.513	4	0.607	4	0.727	4	0.791	4	0.831	4
	CH	0.221	2	0.388	3	0.482	3	0.616	3	0.69	4	0.741	4
	BIO	0.189	7	0.32	7	0.395	8	0.508	8	0.577	8	0.627	8

Tabelle 13: Genauigkeit (Acc) und Ranking (Ran) aller Kombinationen für alle Top-n in allen Fachgebieten (FG)

Die Ergebnisse zeigen, dass sich die Qualität über die Fachgebiete hinweg unterscheidet und das Ranking der Qualität eines Modells je nach Fachgebiet unterschiedlich sein kann. Abgesehen von den am besten abschneidenden Modellen tf-idf - LR und word2vec - MLP (Ausnahme Top-3 für CH) gibt es keine weiteren Modelle bzw. Kombinationen, die bei allen Varianten von Top-n und Fachgebieten immer das gleiche Ranking haben. Abbildung 6 illustriert diese Erkenntnisse.

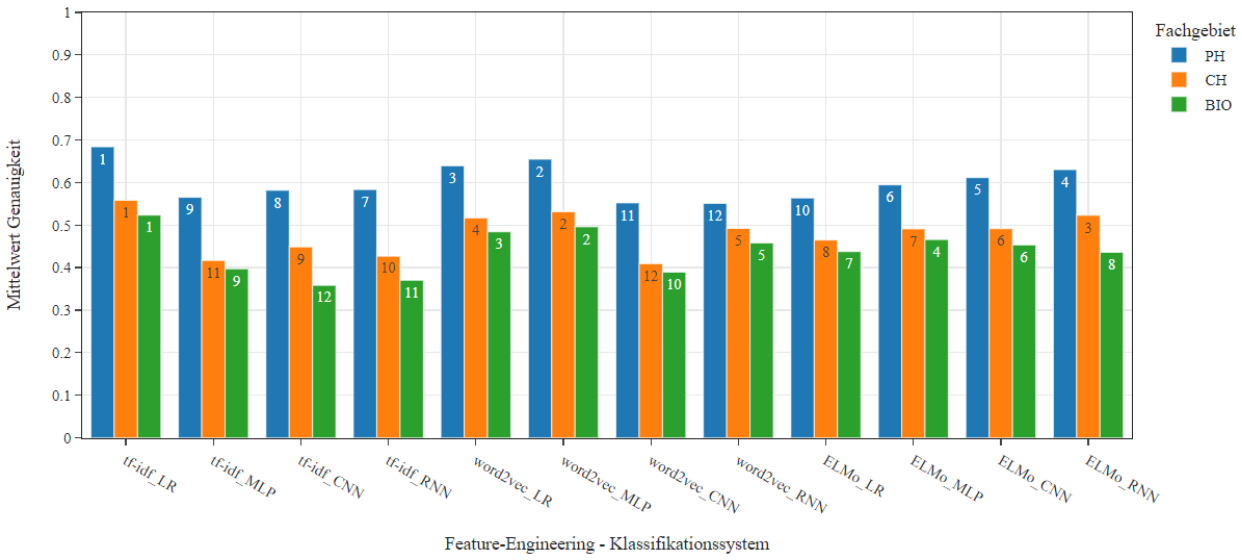

Abbildung 6: Qualitätsranking aller Modelle pro Fachgebiet basierend auf den Mittelwerten der Genauigkeiten aller Top-n

Bestimmte Modelle wie z.B. word2vec - RNN erreicht bei PH Rang 12, aber für CH und BIO Rang 5. Das Modell liefert somit für CH und BIO bessere Ergebnisse als für PH im Vergleich zu den anderen Modellen. Das Gegenteil ist der Fall für tf-idf - RNN. Dieses liefert mit PH-Daten die höhere Qualität als bei Verwendung von CH- oder BIO-Daten. Zusammenfassend gilt, dass sich ausser tf-idf - LR und word2vec - MLP keine Kombination fachgebietsübergreifend bewährt.

Die Erhöhung der Qualität aufgrund der Erhöhung der Top-n weist über alle Fachgebiete hinweg ein ähnliches Muster auf. Abbildung 7 illustriert diesen Zusammenhang.

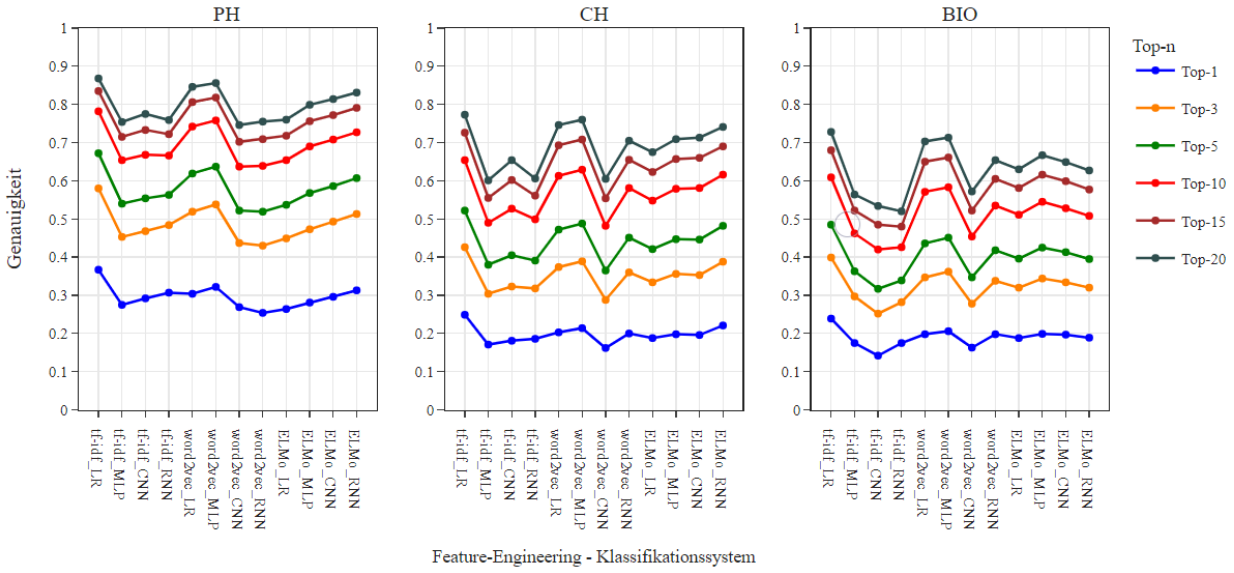

Abbildung 7: Änderung der Genauigkeit durch Erhöhung von Top-n für alle Modelle in allen Fachgebieten

Der Genauigkeitsunterschied zwischen Top-1 und Top-3 ist bei allen Fachgebieten und Modellen der Grösste. Die Differenz zwischen den erzielten Genauigkeiten zwischen Top-1 und Top-10 ist nennenswert grösser als jene zwischen Top-10 und Top-20. Unabhängig vom Fachgebiet scheint sich die Genauigkeit bis Top-10 also schnell und danach nur noch langsam zu erhöhen.

7 Diskussion

Im Rahmen dieser Studie konnten durch Entwicklung von inhaltsbasierten Empfehlungssystemen Erkenntnisse über das bestgeeignete Feature-Engineering, das bestgeeignete Klassifikationssystem, sowie die beste Kombination zwischen Feature-Engineering und Klassifikationssystem gewonnen werden. Ebenfalls gab sie Aufschluss darüber, ob die Nutzung von Daten von verschiedenen Fachgebieten einen Einfluss auf die Qualität eines Modelles bzw. einer Kombination haben kann. Damit werden Fragen beantwortet, welche bisher in keiner Analyse weder in diesem Umfang von Kombinationen noch unter Verwendung von ähnlich grossen Datenmengen untersucht worden sind. Nichtsdestotrotz bestehen aber einige Kritikpunkte, die sich auf die Evaluation der Studie sowie die Qualitätsbeurteilung der trainierten Modelle auswirken und im Folgenden vorgestellt werden sollen.

7.1 Datenvorbereitung

Ein Aspekt, der alle drei verwendeten Datensätze betrifft, ist die Datenverteilung der Publikationen. In PH, CH und BIO stammen 55%, 56% und 50% der hier verwendeten Abstracts von nur 10% der Journals im jeweiligen Datensatz. Das heisst, dass die Systeme für manche Journals mit viel mehr Samples trainiert werden als für die anderen. Diese unbalancierte Datenverteilung kann negative Einflüsse auf die Qualität der Klassifizierungssysteme haben. Die in dieser Studie vorgenommene Limitation der minimalen Anzahl Publikationen pro Journal konnte diese Problematik reduzieren, löst sie aber nicht komplett. Diese Problematik kann im Rahmen eines Klassifikationsverfahren kaum vermieden werden. Andere Verfahren wie beispielsweise *Ähnlichkeit* wären für den Umgang mit dieser Problematik womöglich besser geeignet.

Zudem könnte die Festlegung einer Mindestanzahl Publikationen pro Journal zwar grundsätzlich einen positiven Anstoss für Verlage darstellen, dass sie aktiv mehr publizieren, um von einem Empfehlungssystem empfohlen zu werden. Gleichzeitig hat diese Minimalgrenze jedoch möglicherweise auch einen negativen Einfluss auf die Qualität des Empfehlungssystems. Es gibt Journals, die jährlich nicht viele Artikel publizieren und trotzdem eine hohe Qualität bzw. einen hohen Impact haben. Das Journal *Living Reviews in Relativity* nimmt zum Beispiel im Jahr 2021 den 3. Platz ein im *Scimago Journal Ranking*, obwohl es in den drei Jahren davor nur 17 Publikationen veröffentlicht hat (Scimago Journal & Country Rank, 2022). Wenn solche Journals ausgeschlossen werden, können sie nicht empfohlen werden, obwohl sie wegen ihrer guten Qualität und ihrer fachlichen Ausrichtung für eine bestimmte Publikation die beste Wahl sein könnten. Darüber hinaus

bezog sich die Berechnung der erforderlichen Minimalzahl der Publikation pro Journal auf den gesamten definierten Publikationszeitraum. Für allfällige erst im Lauf dieser Periode eingeführte Journals wurde daher der gesamte Zeitraum als Berechnungsmassstab angelegt. So wurden Journals als relevant beurteilt, die beispielsweise in den letzten Jahren keine Publikationen veröffentlichten, wenn sie in früheren Jahren viel publiziert hatten. Hingegen fehlen Journals, die erst im Laufe des berücksichtigen Zeitraums eingeführt wurden und deswegen die notwendige durchschnittliche Publikationszahl nicht erreichten. Die Begrenzung auf bestimmte Zeiträume bringt in der Praxis zudem eine weitere Herausforderung mit sich. Dadurch werden Journals, die sich mit Themen, die in einem bestimmten Zeitraum häufig behandelt (z.B. Trends) oder erst ab einem gewissen Entdeckungszeitpunkt überhaupt eingeführt (z.B. bahnbrechende Forschungserkenntnisse) wurden, möglicherweise falsch bzw. nicht anhand ihres aktuellen oder gesamten Portfolios repräsentiert und vorgeschlagen. Die Gefahr einer solchen Verzerrung besteht, wenn im Rahmen des Zeitraums, auf dem das Empfehlungssystem basiert, diese Inhalte entweder unter- oder überrepräsentiert wurden.

Des Weiteren erforderte der Ablauf der verschiedenen Schritte bei der Datenaufbereitung Entscheidungen, die auch gewisse Nachteile mit sich brachten. So wurde die Limitation auf die minimale Anzahl Publikationen pro Journal gesetzt und durchgeführt, bevor die nicht-englischsprachigen Abstracts und die LaTeX-Tags entfernt wurden. Die Identifizierung von nicht-englischsprachigen Texten und die Behandlung von Texten mit LaTeX-Tags sind rechenintensiv und nehmen relativ viel Zeit in Anspruch. Aus diesem Grund erschien es sinnvoll, die Einschränkung der minimalen Zahl der Publikationen zuerst vorzunehmen, um die Anzahl Abstracts, die von der Sprach- und LaTeX-Identifikation betroffen sind, vorgängig zu reduzieren und dadurch den Prozess der Identifikation zu beschleunigen. Dadurch wurden Journals als relevant identifiziert, weil sie die Mindestpublikationsanzahl überschritten. Danach wurden jedoch einige dazugehörende Einträge noch entfernt, weil sie entweder nicht auf Englisch verfasst wurden oder weil bei ihnen nach Entfernung von LaTeX-Tags kein Text mehr vorhanden war. Dies führte dazu, dass am Ende Journals im Datensatz verblieben, deren verwendete Anzahl Publikationen die festgelegte Limitation unterschritten. Obwohl diese weniger als 1% der berücksichtigten Journals des jeweiligen Fachgebiets ausmachten, hatte dies Auswirkungen auf die Datenverteilung und möglicherweise auch auf die Qualität der trainierten Klassifikationssysteme.

Eine weitere Optimierungsmöglichkeit bei der Datenvorbereitung liegt in der Entfernung von Zahlen. Im Rahmen der Normalisierung wurden nur digitale Zahlen entfernt. Wären zusätzlich auch Zahlen in Textform identifiziert und entfernt worden, hätte dies einerseits

zu besseren Extraktionen der Features führen können, andererseits zur Reduzierung der Dimensionalität der Daten beigetragen, welche im Rahmen von Feature-Engineerings entstanden sind. Beide Aspekte könnten einen Einfluss auf die Qualität der Modelle haben.

7.2 Modellaufbau

Damit mit den Klassifikationssystemen die bestmögliche Qualität erzielt werden konnte, wurden die optimalen Parameter-Werte durch Hyperparameter-Tuning anhand eines Sample-Datensatzes durchgeführt. Auf diese Weise konnten alle Modelle für die drei Fachbereiche trainiert und generalisierbare Aussagen über den Einfluss dieser Fachgebiete auf die Qualität gemacht werden. Der für diesen Zweck verwendete Sample-Datensatz wurde jedoch den Daten von PH entnommen. Der Grund besteht darin, dass der Autor dieser Arbeit den Datensatz für PH früher nutzen konnte, als dies für die anderen Disziplinen der Fall war. Dies hat aber möglicherweise dazu geführt, dass die Modelle für die Daten von PH optimiert wurden und entsprechend weniger geeignet sind für die Nutzung mit anderen Daten. Dies könnte eine mögliche Interpretation dafür sein, warum die Modelle, welche mit den Daten von PH trainiert wurden, klar höhere Genauigkeiten erreichten, als dies mit CH und BIO der Fall war.

Zudem wurden die Daten für das Hyperparameter-Tuning auf 67% Trainings- und 33% Testdaten aufgeteilt. Erst später, beim Trainieren mit den vollständigen Daten, wurde entschieden, dass die Daten auf 80% Trainings- und 20% Testdaten aufgeteilt werden sollen, da 20% dieser Datenmenge zur Evaluation der Modelle ausreichen sollten. Diese Abweichung könnte einen Einfluss auf die Qualität der Modelle haben, da das Hyperparameter-Tuning auf einer anderen Logik oder Aufteilungsrate erfolgt ist als diejenige, mit der die Modelle am Ende trainiert wurden. Es könnte dementsprechend sein, dass die Parameter andere optimale Werte gehabt hätten, wenn die Daten im Sample-Datensatz eine andere Aufteilung (80%-20%) gehabt hätten. Oder die Modelle hätten möglicherweise andere Genauigkeiten aufgewiesen, wenn für die kompletten Daten dieselbe Aufteilung (67%-33%) wie für das Hyperparameter-Tuning beibehalten worden wäre. Für zukünftige Arbeiten sollte eine solche Abweichung nicht vorgenommen, sondern stattdessen im Vorfeld eine der beiden Varianten (eher 80%-20%) für alle Schritte konsistent angewendet werden.

Das Hyperparameter-Tuning für MLP hat in dieser Analyse aus Zeitgründen nicht in Kombination mit allen Feature-Engineerings stattgefunden. Dies wurde aufgrund des Zeitaufwandes bei tf-idf bzw. daraus hervorgehend der Entscheidung zur einheitlichen

Behandlung aller Feature-Engineerings so festgelegt. Doch die durch Kombinationen mit word2vec und ELMo extrahierten Features würden keine so hohe Dimensionalitäten haben wie im Fall von tf-idf. Das Fine-Tuning wäre dafür also nicht so zeitaufwändig und somit möglich gewesen. Dies hätte ein besseres Abbild der Qualität der beiden darauf basierenden Modelle ermöglicht – allerdings wegen der Andersbehandlung auf Kosten der Vergleichbarkeit der drei Modelle. Des Weiteren wurde ebenfalls wegen des Zeitaufwandes das Hyperparameter-Tuning von CNN und RNN mit ELMo im Vergleich zu denselben Kombinationen mit word2vec auf weniger Varianten von Kombinationen, Parametern und Strukturen beschränkt. Wenn die Zeit dafür zur Verfügung steht, sollte man dieses aber aus Gründen der Einheitlichkeit umfassender durchführen. Dies, weil dadurch noch optimalere Werte der Parameter oder eine optimalere Struktur gefunden werden könnte, was vermutlich letztendlich die Qualität dieser Modelle verbessern würde. Obwohl das word2vec-Modell nur für LR optimiert wurde, wurde es auch für alle anderen Klassifikationssystemen verwendet. Im Fall von MLP war dies inhaltlich sinnvoll. Bei RNN wurden lediglich die Epochen von 30 auf 100 erhöht, darüber hinaus aber nichts anderes angepasst. Obwohl in der vorliegenden Studie keine ungewöhnlichen Auswirkungen festzustellen waren, sollten word2vec-Modelle jeweils individuell für CNN und RNN optimiert werden, da CNN und RNN im Vergleich zu LR und MLP anders funktionieren. Solche Optimierungen könnten, wie bei RNN beobachtet, die Genauigkeit der CNN und RNN erhöhen.

Beim Trainieren des Modells tf-idf - RNN wurde die Anzahl *units* in den drei Fachgebieten uneinheitlich gesetzt. Das Modell tf-idf - RNN für BIO wurde zuletzt, nach PH und CH, trainiert. Da die Dimensionen der Daten bzw. der extrahierten Features von tf-idf in Fall von BIO viel höher waren als bei PH und CH, war *units=256* für PH und CH zwar problemlos möglich, doch für BIO reichte die Hardware-Kapazität nicht aus und erforderte eine Reduktion auf *units=200.* Für ein einheitliches Training sollte zuerst mit den komplexesten Daten, in diesem Fall BIO, begonnen werden und mit der für sie passenden Struktur die neuronalen Netze aufgebaut und trainiert werden. Diese Struktur kann dann für die einfacheren Daten, hier PH und CH, übernommen werden. Dies kann wichtig sein, denn wenn die Einheitlichkeit der Struktur von neuronalen Netzen über die Fachgebiete hinweg nicht gewährleistet ist, könnte die Vergleichbarkeit zwischen den Fachgebieten in Frage gestellt werden.

7.3 Relevanz der Hardware

In dieser Arbeit wurden die Eigenschaften der Hardware erwähnt, welche zur Durchführung dieser Analyse verwendet wurde. Dies hielt der Autor dieser Arbeit für wichtig, denn in anderen vergleichbaren verfügbaren Studien dieses Forschungsgebiets wird die verwendete Hardware kaum thematisiert. Diese Information wäre jedoch hilfreich gewesen, denn erst bei der Durchführung wurden das Ausmass der Hardware-Anforderungen bzw. notwendige Einschränkungen des Forschungsvorhabens infolge ungenügender Kapazitäten deutlich. Informationen über Hardware-Anforderungen, insbesondere beim Aufbau von neuronalen Netzen, sollten daher standardmässig in Publikationen thematisiert werden.

Ein Beispiel für eine mögliche Einschränkung ist die lange Trainingszeit von neuronalen Netzen, wenn kein GPU vorhanden ist. Es hat sich gezeigt, dass das Trainieren von neuronalen Netzen auf GPUs die Trainingszeit erheblich reduziert (auf etwa 60%-80%). Wenn die Dauer der Trainingszeit keine Rolle spielt, könnten die neuronalen Netze mit möglicherweise komplexeren Strukturen auf CPU (Central Processing Unit) trainiert werden. Da CPUs in der Regel mehr Zwischenspeicherkapazität haben als GPUs, könnten damit neuronale Netze mit komplexeren Strukturen aufgebaut und auch mit grösseren Datenmengen problemlos trainiert werden. Wie bereits erläutert, wurde das Hyperparameter-Tuning für CNN und RNN wegen der beschränkten Hardwarekapazität bzw. dem limitierenden Zwischenspeicher schrittweise und manuell auf der GPU durchgeführt. Die CPU wäre aber besser geeignet, da sie alle möglichen Kombinationen von Parametern und Strukturen in nur einem Schritt testen könnte.

7.4 Operationalisierung und Interpretation der Ergebnisse

Ähnlich wie in vielen früheren Studien wurde auch hier die Metrik Genauigkeit verwendet, um die Qualität der Systeme bzw. der Modelle zu evaluieren – also eine der einfachsten und am häufigsten verwendeten Metriken, um die Qualität von Klassifikationssystemen zu beurteilen (Grandini et al., 2020). Diese Metrik funktioniert gut, wenn das Interesse darin liegt, zu evaluieren, ob die einzelnen Einträge, in diesem Fall sind es Abstracts, richtig klassifiziert werden bzw. die richtige Klasse (Journal) vorhergesagt wird (Grandini et al., 2020). Wenn aber das Interesse darin liegt, zu beurteilen, ob die unterschiedlichen Klassen ähnlich gut vorhergesagt werden können, kann diese Metrik im Fall von unbalancierter Datenverteilung die Beurteilung der Qualität trüben. Dies, weil bei der Berechnung der Genauigkeit häufiger vorhandene Klassen mehr gewichtet werden als weniger häufig vorhandene Klassen (Grandini et al., 2020). Um dies zu vermeiden und die

Qualität der Modelle genauer evaluieren zu können, sollten weitere Metriken wie beispielsweise Precision, Recall, Balanced Accuracy sowie F1-Score verwendet werden. Im Rahmen dieser Studie konnten diese Metriken nicht verwendet werden, denn gemäss Wissensstand des Autors gibt es keine bereits vorhandenen Funktionen in den Bibliotheken z.B. von *Scikit-learn*, die diese Metriken berechnen können, wenn mehr als ein Treffer (Top->1) berücksichtigt werden soll. Solche Funktionen könnten implementiert werden, im Rahmen dieser Studie war dies aber aus Kapazitätsgründen nicht möglich.

Bei der Betrachtung der Genauigkeit der Modelle fällt auf, dass sie ziemlich gering ausfällt. Insbesondere für Top-1 und Top-3. Dies könnte auf mehrere Gründe zurückzuführen sein:

- Manche Journals konkurrieren miteinander. Es gibt also zuweilen eine Überlappung von abgedeckten Themen in den Publikationen unterschiedlicher Journals (vgl. (Wang et al., 2018)). Das Resultat davon kann zum Beispiel eine Vorhersage für Klasse A sein, obwohl die richtige Klasse B wäre. So eine Vorhersage gilt bei der Evaluation als falsche, was aber in Wirklichkeit nicht zutrifft, denn Klasse A ist ggf. genauso gut geeignet für ein bestimmtes Abstract wie Klasse B. Diese Konkurrenzbeziehung besteht oft nicht nur zwischen zwei, sondern gleich mehreren Journals.
- Einen anderen Grund kann die hier vorliegende unbalancierte Datenverteilungen darstellen. Modelle, die mit Einträgen bestimmter Klassen viel mehr trainiert werden, merken sich diese Klassen besser als die anderen, was einen Einfluss auf die Qualität des Modells haben kann.
- Ein weiterer Grund könnte sein, dass die Datenmenge erst nach der Datenvorbereitung halbiert wurde. Dies führte dazu, dass jede Klasse nur mit der Hälfte der zur Verfügung stehenden Daten trainiert wurde, während gleichzeitig die Anzahl der Klassen nicht reduziert wurde. Wäre die Entscheidung, mit der Hälfte der Daten zu arbeiten, früher bzw. bei der Konzipierung der Studie getroffen worden, wäre die Datenmenge bei der Erhebung berücksichtigt worden. Dadurch hätte sich eventuell eine geringere Anzahl an Klassen mit derselben Menge der Daten ergeben. Dies bedeutet, dass obwohl mit derselben Menge der Daten gearbeitet worden wäre, jede Klasse mit mehr Einträgen hätte trainiert werden können. Dies sollte theoretisch zu einer besseren Qualität der Modelle führen. Diese Annahme scheint durch die Ergebnisse der vorhandenen Analyse gestützt zu werden.
- Betrachtet man die Qualität der Modelle für PH, fällt auf, dass sie eine höhere Genauigkeit erzielen als die Modelle für CH und BIO, letztere erzielen die geringste Genauigkeit. Ein Grund dafür könnte sein, dass obwohl die Daten ähnlich

viele Publikationen aus den unterschiedlichen Fachgebieten enthielten, diese von unterschiedlich vielen Journals publiziert worden sind. In PH ist die Anzahl der berücksichtigen Journals bzw. Klassen 590, für CH sind es 1038 und für BIO 2133. Rein mathematisch bedeutet dies, dass jede Klasse in PH fast doppelt so viele Einträge enthält wie die Klassen in CH und vierfach so viele wie in BIO. Dementsprechend gibt es in BIO nur fast halb so viele Einträge pro Klasse wie in CH. Die Differenz in der Genauigkeit zwischen PH und den anderen Fachgebieten kann auch daran liegen, dass die Modelle bzw. das Hyperparameter-Tuning für die Modelle mit den Daten von PH durchgeführt wurde und die Modelle somit für das Trainieren mit den Daten von PH optimiert worden sind. Die Betrachtung der Differenz zwischen der Genauigkeit zwischen PH, CH und BIO stützt diese Annahme. Die Differenz in der Genauigkeit zwischen CH und BIO ist nicht so gross wie die Differenz zwischen PH und CH (vgl. Abbildung 5), obwohl das Verhältnis der Verteilung Anzahl Einträge pro Klasse zwischen PH und CH vergleichbar ist mit jenem zwischen CH und BIO.

Die Antwort auf die erste Forschungsfrage lautet, dass word2vec und ELMo im Vergleich zu tf-idf im Durchschnitt in Kombination mit allen Klassifikationssystemen die bessere Qualität aufweisen. Die Antwort auf die dritte Forschungsfrage nach der besten Kombination lautet hingegen tf-idf - LR. Diese zwei Erkenntnisse erscheinen auf den ersten Blick paradox, denn wie kann es sein, dass tf-idf schlechtere Ergebnisse liefert und trotzdem Teil der besten Kombination ist? Es handelt sich bei den Ergebnissen von tf-idf in Kombination mit LR um einen Ausreisser, denn ausser in Kombination mit LR hat tf-idf im Durchschnitt immer niedrigere Werte als word2vec und ELMo. Bei den beiden anderen Feature-Engineering liegt jedoch kein solcher Ausreisser vor, daher haben sie im Durchschnitt höhere Werte als tf-idf.

Die Ergebnisse dieser Studie deuten darauf hin, dass tf-idf - LR die beste Kombination ist. Allerdings weist der Autor dieser Arbeit darauf hin, dass diese Kombination viele Zwischenspeicher-Ressourcen benötigt. Für das Trainieren dieses Modells mit der in dieser Analyse verwendetet Datenmenge waren 128 GB RAM nicht genug. Beim Aufbau eines Empfehlungssystems für Journals könnte die verwendete Datenmenge ein Mehrfaches grösser sein. Falls nicht genug RAM zur Verfügung steht, könnte word2vec - MLP als Alternative in Frage kommen. Dieses Modell benötigt zum Trainieren viel weniger Zwischenspeicher. Dieser könnte bei der Datenvorbereitung jedoch in grosser Kapazität benötigt werden (vgl. Kapitel 5.4.2.1). Man kann dieses Problem umgehen, indem man die Daten schrittweise jeweils nur mit einem Teil der Daten vorbereitet.

Die Ergebnisse zeigen, dass, selbst wenn ein Modell wie beispielsweise tf-idf - LR für alle Fachgebiete die beste Option sein könnte, nicht ein Modell für alle Fachgebiete erstellt bzw. alle Fachgebiete mit einem Modell abgedeckt werden sollten. Wie bereits gesehen, kann die Verwendung von Daten von verschiedenen Fachgebieten zu Unterschieden in der Qualität eines Modelles führen. Deswegen scheint es dem Autor sinnvoller, wenn für jedes Fachgebiet ein separates Modell erstellt wird, welches für die Daten dieses Fachgebietes optimiert ist. In der Praxis bedeutet dies, dass die NutzerInnen des Empfehlungssystems ein Fachgebiet eingeben bzw. auswählen müssten. Im Hintergrund würde dann das jeweils passende Modell abgerufen und für die Empfehlung verwendet. Vor diesem Hintergrund stellt sich dann natürlich die Frage, wie gut so ein System für die Empfehlung von Journals für interdisziplinäre bzw. fachübergreifende Publikationen ist.

8 Implikationen

In diesem Kapitel werden auf Basis der Erkenntnisse dieser Arbeit einige Hinweise und Vorschläge gemacht, die in der Forschung bzw. in der Praxis bei einer Implementierung eines Empfehlungssystems für wissenschaftliche Journals berücksichtigt werden sollten.

8.1 Implikationen für die Forschung

Während diese Studie erste Antworten für mehrere offene Fragen des Forschungsgebiets bereithält, wirft sie gleichzeitig zahlreiche weitere auf, welche durch zukünftige Forschungsarbeiten weiterbearbeitet bzw. beantwortet werden sollten.

So beschränkt sich die vorliegende Studie auf die Extraktion der Features mit nur drei Feature-Engineerings. Dadurch sind andere Feature-Engineerings ausgeschlossen. Es wäre interessant zu prüfen, ob die in dieser Analyse ausgeschlossenen Feature-Engineerings wie z.B. GloVe, Fasttext, doc2vec und sentence-Bert zu einer besseren Extraktion der Features und dementsprechend zu einer besseren Qualität der trainierten Modelle führen könnten.

Auch in der Verwendungsart der hier genutzten Feature-Engineerings bzw. Klassifikationssysteme gibt es noch viel Spielraum, um die besten Einstellungen für eine optimale Handhabung zu finden. So wurde z.B. das Klassifikationssystem MLP im Rahmen dieser Analyse ohne Hyperparameter-Tuning verwendet. Offen bleibt die Frage, ob die Qualität für dieses Klassifikationssystem besser ausgefallen wäre, wenn dieses in Kombination mit den hier verwendeten Feature-Engineerings durch Fine-Tuning optimiert worden wäre. Dies gilt ebenfalls für die neuronalen Netze wie z.B. tf-idf - CNN, die wegen der beschränkt zur Verfügung stehenden Hardware-Kapazität nicht genau so aufgebaut werden konnten, wie es durch das Fine-Tuning ermittelt wurde. Es wäre interessant zu wissen, ob sie durch den optimalen Aufbau eine höhere Genauigkeit hätten erreichen können, bzw. um wie viel höher diese ausgefallen wäre.

In dieser Analyse wurde für Klassifikationssysteme in Kombination mit tf-idf mit Stemming-Daten, in Kombination mit word2vec und ELMo aber mit lemmatisierten Daten gearbeitet, um für jedes Feature-Engineering die besten Ergebnisse zu erreichen. Die weitere Forschung sollte jedoch klären, ob diese Entscheidung sich als sinnvoll erweist, indem in einer ähnlichen Analyse überall die gleiche Grundform der Daten bzw. der Wörter verwendet würde. Eine weitere offene Frage in Bezug auf die für solche Analysen am besten geeignete Wortform liegt in der Haltung von (Kutuzov & Kuzmenko, 2019), dass beim Arbeiten mit ELMo die Lemmatisierung für englischsprachige Texte nicht notwendig

sei. Dies dürfte nach Verständnis des Autors der vorliegenden Studie auch für das Stemming gelten. Es wäre deshalb interessant, dies mit Blick auf Abstracts und die Klassifizierung von Journals zu überprüfen.

Auch die Auswahl der Journals, die in einem Empfehlungssystem abgebildet werden sollen, ist höchst relevant und bislang nicht ausreichend geklärt. Im Rahmen dieser Analyse wurde aus experimentellen Gründen entschieden, eine Einschränkung in Form einer minimalen durchschnittlichen Publikationsmenge vorzunehmen. Weiterführend könnte eine ähnliche Analyse wie die vorliegende, aber ohne Einschränkungen für die Journals, durchgeführt werden. Dadurch könnte evaluiert werden, ob die Verwendung aller vorhandenen Journals zu besserer Qualität eines Klassifizierungs- bzw. Empfehlungssystems führen kann.

Die forschungsleitenden Fragen der vorliegenden Studie könnten ausserdem mit anderen Verfahren als der Klassifikation oder mit einer Kombination von Verfahren untersucht werden. So könnten die Publikationen beispielsweise in unterschiedliche Themengebiete unterteilt werden. Dementsprechend würden auch die Journals thematisch unterteilt. Für eine Empfehlung müsste das Paper, für das ein passendes Journal gesucht wird, einem der vordefinierten Themengebiete zugewiesen werden. Dann könnte die Ähnlichkeit zwischen dem Paper und den Artikeln jedes Journals berechnet werden. Das Journal, dessen Artikel die grösste Ähnlichkeit aufweisen, wird empfohlen. Die Ergebnisse einer solchen Analyse könnten mit den Ergebnissen der vorliegenden Arbeit verglichen werden und es könnte beurteilt werden, ob dadurch bessere Resultate erzielt werden könnten.

In der Diskussion wurde erwähnt, dass die Klassifikationssysteme eine nicht besonders hohe Genauigkeit für Top-1 und Top-3, u.a. möglicherweise wegen der unbalancierter Datenverteilung, aufweisen. Es wäre zu untersuchen, ob mit anderem Verfahren wie z.B. *Ähnlichkeit* mit der Verwendung derselben Daten bessere Ergebnisse erreicht werden könnten. (Beyan & Fisher, 2015) haben für eine Klassifizierungsaufgabe mit unbalancierten Daten eine Methode namens *hierarchical decomposition*, basierend auf Clustering und Ausreisser-Erkennung, verwendet. Die Hierarchie konnte in ihrer Analyse anhand der Ähnlichkeiten gelabelter Daten aufgebaut werden. Mit dieser Methode konnten sie für ihre Aufgabe im Vergleich zu Klassifikationsverfahren bessere Ergebnisse erzielen. Vergleichbare Analysen, welche Verfahren und Methoden nutzen, die auf der Ähnlichkeit beruhen, könnten auch im Zusammenhang mit der Empfehlung von Journals untersucht werden.

Auch die Frage, wie gut die Qualität eines Empfehlungssystems basierend auf einer Kombination von Klassifikation und Ähnlichkeit ausfallen würde, verdient Beachtung.

Dabei könnte ein Klassifikationssystem erstellt werden, wobei als Vorhersage zum Beispiel die ersten 30 Treffer berücksichtigt würden. In einem zweiten Schritt würde dann die Ähnlichkeit zwischen der eigenen Arbeit und den Dokumenten jedes dieser 30 Journals berechnet. Das Journal, dessen Dokumente am ähnlichsten zur eigenen Arbeit sind, wird dann empfohlen. Dadurch liesse sich herausfinden, ob ein Empfehlungssystem, welches auf so einer Kombination aufgebaut ist, bessere Empfehlungen machen kann als ein System, das nur auf Klassifikations- oder Ähnlichkeitsverfahren basiert.

8.2 Implikationen für die Praxis

Die Erkenntnisse, welche durch diese Studie gewonnen werden konnten, führen zu verschiedenen Implikationen für die Praxis. So schaffen sie eine Wissensbasis, mit welchen Verfahren bzw. mit welchen der ausgewählten Feature-Engineerings bzw. Klassifizierungssysteme beim Aufbau eines Empfehlungssystems für Journals was erreicht kann. Sie helfen somit, schnell eine bewusste Entscheidung beim Festlegen des Aufbaus eines Empfehlungssystems zu treffen. Da mit tf-idf - LR und word2vec - MLP fachbereichsübergreifend die höchste Genauigkeit erreicht wurden, können diese Modelle beim Aufbau eines Empfehlungssystems bevorzugt werden von Institutionen, die grosse Datenmengen von gelabelten Daten besitzen und ein inhaltsbasiertes Empfehlungssystem basiert auf dem Klassifikationsverfahren aufbauen wollen.

Neben den Ergebnissen der Evaluation der unterschiedlichen Modelle kann dieser Studie auch entnommen werden, wie die Modelle trainiert und die Daten vorbereitet wurden. Mit den Erkenntnissen und Hinweisen, was für die jeweiligen Entscheidungen bzw. bei den Prozessschritten beachtet oder im Vergleich zum hier angewendeten Vorgehen optimiert werden sollte, kann in der Praxis bei der Modellbildung viel Zeit gespart werden. Auch Rückschläge können vermieden werden. Ein besonders bedeutsamer Aspekt dabei ist die für den Aufbau eines inhaltsbasierten Empfehlungssystems benötigte Hardware, eine Anforderung, die bereits bei der Konzipierung von solchen Projekten analysiert und berücksichtigt werden sollte. Diese Studie enthält modellspezifische Hinweise zu den Anforderungen an passende Hardware bezüglich Prozessor, Arbeitsspeicher oder Graphikkarte.

Zu guter Letzt sind die Erkenntnisse dieser Studie nicht nur für den Aufbau von Empfehlungssystemen für Journals von Bedeutung, sondern können überall dort angewendet werden, wo eine Klassifizierung von Multi-Klassen für textuelle Einträge umzusetzen ist. Insbesondere dann, wenn die Klassenaufteilung nicht balanciert ist.

9 Fazit

Die vorliegende Studie hat analysiert, wie geeignet Empfehlungssysteme für die Auswahl von passenden Journals für wissenschaftliche Manuskripte sind und auf Basis welcher Techniken und Verfahren sie qualitativ gute Empfehlungen machen können. Ergebnis der Recherche bestehender Literatur war die Erkenntnis, dass Empfehlungssysteme auf unterschiedlichen Techniken wie z.B. Collaborative-based Filtering (CF), Content-based Filtering (CB), Hybriden Techniken, d.h. einer Kombination von CF und CB, oder auch auf Netzwerken basieren können. Im Rahmen einiger Studien wurden auch Empfehlungssysteme aufgebaut und analysiert. Hierfür wurden unterschiedliche Techniken und Verfahren verwendet und mit verschiedenen Datensätzen, sowohl hinsichtlich thematischer Ausrichtung als auch Umfang, gearbeitet. Publikationen aus dem Bereich Computer Science wurden in diesen Studien am häufigsten untersucht. CB in Form von Klassifikationsverfahren zeigte sich in der bisherigen Forschung als eine vielversprechende Methode, wobei noch zahlreiche offene Fragen betr. der erfolgsbestimmenden Faktoren bestehen. Einige davon sollten im Rahmen der vorliegenden Analyse beantwortet werden.

Anhand von vier Forschungsfragen wurde analysiert, mit welchem Feature-Engineering, welchem Klassifikationssystem und welcher Kombinationen von Feature-Engineering und Klassifikationssystem ein inhaltsbasiertes Empfehlungssystem für Journals qualitativ gute Empfehlungen machen kann. Ebenfalls wurde untersucht, ob die Verwendung von Daten aus verschiedenen Fachgebieten einen Einfluss auf die Qualität eines Empfehlungssystems hat. Zur Beantwortung dieser Fragen wurde das Verfahren Klassifikation gewählt, mit den Ansätzen Logistic Regression (LR), Multi Layer Perceptron (MLP) Classifier, Convolutional Neural Network (CNN) und Recurrent Neural Network (RNN) bzw. Long Short Term Memory (LSTM). Von den unterschiedlichen Feature-Engineerings wurden die Term frequency - Inverse document frequency (tf-idf), word2vec und Embedding from Language Model (ELMo) zur Extraktion von Features ausgewählt und verwendet.

Die verwendeten Daten beinhalten Abstracts von Publikationen und die Titel der Journals, in denen die Publikationen veröffentlicht wurden. Sämtliche Daten stammen von der Plattform «Dimensions». Für die Analyse wurden drei Datensätze von unterschiedlichen Fachgebieten – Physik (PH), Chemie (CH) und Biologie (BIO) – verwendet. Nach der Erhebung der Daten sind diese durch eine Vorbereitungspipeline für die Analyse vorbereitet worden. Diese Pipeline beinhaltete die Entfernung von leeren Einträgen, Abstracts von Journals, welche als irrelevant identifiziert worden sind sowie von nicht in Englisch verfassten Abstracts und LaTeX-Tags. Ebenfalls beinhaltete die Pipeline die

Normalisierung der Texte, beispielsweise also die Entfernung von Stoppwörtern, Zahlen, nicht alphabetischer Zeichen sowie die Konvertierung der Wörter zu ihrer jeweiligen Grundform. Nach der Vorbereitung wurden von den zur Verfügung stehenden Daten 750'937 Abstracts von 590 Journals für PH, 855'324 Abstracts von 1’038 Journals für CH und 847'109 Abstracts von 2’133 Journals für BIO für die Analyse verwendet.

Im Rahmen der Analyse wurde jedes der Klassifikationssysteme mit jedem der Feature-Engineerings kombiniert bzw. mit den extrahierten Features von jedem Feature-Engineering trainiert und evaluiert. Dies bedeutet, dass mit der Verwendung der Daten von jedem Fachgebiet 12 Modelle trainiert und evaluiert wurden. Insgesamt entstanden demensprechend 36 Modelle für die drei Fachgebiete. Bevor die Klassifikationssysteme trainiert worden sind, wurde in einem ersten Schritt das Hyperparameter-Tuning durchgeführt. Für dieses wurde ein Sample-Datensatz des Fachgebiets PH von 15'019 Abstracts und 573 unterschiedlichen Journals verwendet. Im Rahmen dieses Schrittes wurde einerseits herausgefunden, wie die Features mit den erwähnten Feature-Engineerings extrahiert werden können und andererseits, welches die bestmöglichen Werte für die Parameter der Klassifikationssysteme bzw. die bestmögliche Struktur für die neuronalen Netze sind. Mit den gewonnenen Erkenntnissen aus dem Hyperparameter-Tuning konnten die Modelle trainiert und evaluiert werden. Zur Evaluation der Qualität der Modelle wurde hier die Metrik Genauigkeit verwendet.

Die Ergebnisse haben gezeigt, dass word2vec und ELMo in Kombination mit allen Klassifikationssystemen im Durchschnitt eine bessere Qualität aufwiesen als tf-idf. LR erreichte in Kombination mit allen Feature-Engineerings immer die höchste Genauigkeit. Interessanterweise liegt die beste Kombination zwischen Feature-Engineering und Klassifikationssystem für alle Fachgebiete und für alle Top-n im durchschnittlich am wenigsten erfolgreichen Feature-Engineering tf-idf und LR, darauf folgt auf dem zweiten Rang die Kombination word2vec - MLP. Die Ergebnisse haben auch gezeigt, dass die Qualität eines Klassifikationssystems vom Fachgebiet abhängig sein kann. Die zwei höchsten Genauigkeiten konnten jedoch für alle drei Fachgebiete mit den gleichen Modellen bzw. Kombinationen erzielt werden, wenn auch auf unterschiedlichem Niveau. Mit der Kombination tf-idf - LR konnten Genauigkeiten für Top-1, Top-10 und Top-20 von 37%, 78% und 87% für PH, 25%, 65% und 77% für CH sowie 24%, 61% und 73% für BIO erzielt werden.

10 Literaturverzeichnis

Adey, B. (2021). *Investigating ML Model Accuracy as Training Size Increases*. https://purple.telstra.com/blog/investigating-ml-model-accuracy-as-training-size-increases

Amidi, A. & Amidi, S. (2022, 26. März). *CS 230 - Recurrent Neural Networks Cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Ayyadevara, V. K. (2018). Word2vec. In V. K. Ayyadevara (Hrsg.), *Pro Machine Learning Algorithms* (S. 167–178). Apress. https://doi.org/10.1007/978-1-4842-3564-5_8

Bai, X., Wang, M., Lee, I., Yang, Z [Zhuo], Kong, X. & Xia, F. (2020, 10. August). *Scientific Paper Recommendation: A Survey*. http://arxiv.org/pdf/2008.13538v1

Beall, J. (2021). *Standalone Journals – Beall's List*. https://beallslist.net/standalone-journals/#update

Beyan, C. & Fisher, R. (2015). Classifying imbalanced data sets using similarity based hierarchical decomposition. *Pattern Recognition*, *48*(5), 1653–1672. https://doi.org/10.1016/j.patcog.2014.10.032

Blei, D. M., NG, A. Y. & Jordan, M. I. (2003). Latent dirichlet allocation. In *The Journal of Machine Learning Research* (S. 993–1022). https://dl.acm.org/doi/10.5555/944919.944937

Bobadilla, J., Ortega, F., Hernando, A. & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, *46*, 109–132. https://doi.org/10.1016/j.knosys.2013.03.012

Breese, J. S., Heckerman, D. & Kadie, C. (2013, 30. Januar). *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*. http://arxiv.org/pdf/1301.7363v1

Brownlee, J. (2020). Softmax Activation Function with Python. *Machine Learning Mastery*. https://machinelearningmastery.com/softmax-activation-function-with-python/

Cabanac, G. (2011). Accuracy of inter-researcher similarity measures based on topical and social clues. *Scientometrics*, *87*(3), 597–620. https://doi.org/10.1007/s11192-011-0358-1

Canvar, W. B. & Trenkle, J. M. (1994). N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.9367

colah's blog. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. & Kuksa, P. (2011). *Natural Language Processing (almost) from Scratch*. http://arxiv.org/pdf/1103.0398v1

Conry, D., Koren, Y. & Ramakrishnan, N. (2009). Recommender Systems for the Conference Paper Assignment Problem. In *ACM Conference on Recommender Systems 2009.* https://arxiv.org/pdf/0906.4044

Croce, D., Castellucci, G. & Basili, R. (2020). GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples. In D. Jurafsky, J. Chai, N. Schluter & J. Tetreault (Hrsg.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (S. 2114–2119). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.191

da Costa, J. P. & Cardoso, J. S. (2005). Classification of Ordinal Data Using Neural Networks. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge & L. Torgo (Hrsg.), *Lecture Notes in Computer Science. Machine learning--ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005 : proceedings* (Bd. 3720, S. 690–697). Springer Berlin Heidelberg. https://doi.org/10.1007/11564096_70

Data Basecamp. (2022). *Stemming vs. Lemmatization | Data Basecamp*. https://databasecamp.de/en/data/stemming-lemmatization#Is_Lemmatization_better_than_Stemming

dblp. (2022). *What is dblp?* dblp computer science bibliography. https://dblp.org/faq/1474565.html%20zitieren

Di Pietro, M. (2020). *Text Classification with NLP: Tf-Idf vs Word2Vec vs BERT*. https://towardsdatascience.com/text-classification-with-nlp-tf-idf-vs-word2vec-vs-bert-41ff868d1794

Elia, F. (24. Juni 2020). Stemming vs Lemmatization. *Baeldung on Computer Science*. https://www.baeldung.com/cs/stemming-vs-lemmatization

Elsevier. (2022). *Elsevier® JournalFinder*. https://journalfinder.elsevier.com/

En, S., Yin Feng & Can, T. (2019). A Quick Dive into Deep Learning: From Neural Cells to BERT. *Alibaba Cloud*. https://alibaba-cloud.medium.com/a-quick-dive-into-deep-learning-from-neural-cells-to-bert-ba8412e55553

Fairbairn, H., Holbrook, A., Bourke, S., Preston, G., Cantwell, R. & Scevak, J. (2014). *A profile of education journals*.

https://www.researchgate.net/publication/228365620_A_profile_of_education_journals

Feng, X., Zhang, H., Ren, Y., Shang, P., Zhu, Y., Liang, Y., Guan, R. & Xu, D. (2019). The Deep Learning-Based Recommender System "Pubmender" for Choosing a Biomedical Publication Venue: Development and Validation Study. *Journal of medical Internet research*, *21*(5), e12957. https://doi.org/10.2196/12957

Garg, S. (2021). *How to Deal with Categorical Data for Machine Learning - KDnuggets*. https://www.kdnuggets.com/2021/05/deal-with-categorical-data-machine-learning.html

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems (Second edition). O'Reilly. https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5892320

Grandini, M., Bagli, E. & Visani, G. (2020, 13. August). *Metrics for Multi-Class Classification: an Overview*. http://arxiv.org/pdf/2008.05756v1

Hornick, M. & Tamayo, P. (2012). Extending Recommender Systems for Disjoint User/Item Sets: The Conference Recommendation Problem. *IEEE Transactions on Knowledge and Data Engineering*, *24*(8), 1478–1490. https://doi.org/10.1109/TKDE.2011.90

Huang, W., Wu, Z., Mitra, P. & Giles, C. L. (2014). RefSeer: A citation recommendation system. In *IEEE/ACM Joint Conference on Digital Libraries* (S. 371–374). IEEE. https://doi.org/10.1109/JCDL.2014.6970192

Huynh, S. T., Huynh, P. T., Nguyen, D. H., Cuong, D. V. & Nguyen, B. T. (2020). S2RSCS: An Efficient Scientific Submission Recommendation System for Computer Science. In H. Fujita, P. Fournier-Viger, M. Ali & J. Sasaki (Hrsg.), *Lecture Notes in Computer Science. Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices* (Bd. 12144, S. 186–198). Springer International Publishing. https://doi.org/10.1007/978-3-030-55789-8_17

Janakiev, N. (2018). Practical Text Classification With Python and Keras. *Real Python*. https://realpython.com/python-keras-text-classification/#convolutional-neural-networks-cnn

Joshi, P. (2019). *What is ELMo | ELMo For text Classification in Python.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Kang, N., Doornenbal, M. A. & Schijvenaars, R. J. (2015). Elsevier Journal Finder: Recommending Journals for your Paper. In H. Werthner, M. Zanker, J. Golbeck & G.

Semeraro (Hrsg.), *Proceedings of the 9th ACM Conference on Recommender Systems* (S. 261–264). ACM. https://doi.org/10.1145/2792838.2799663

Kathrani, K. (2020). *All about Embeddings*. https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b

Khusro, S., Ali, Z. & Ullah, I. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. In K. J. Kim & N. Joukov (Hrsg.), *Lecture Notes in Electrical Engineering. Information Science and Applications (ICISA) 2016* (Bd. 376, S. 1179–1189). Springer Singapore. https://doi.org/10.1007/978-981-10-0557-2_112

Kulshrestha, R. (24. November 2019). NLP 101: Word2Vec — Skip-gram and CBOW - Towards Data Science. *Towards Data Science*. https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

Kutuzov, A. & Kuzmenko, E. (2019, 6. September). To lemmatize or not to lemmatize: how word normalisation affects ELMo performance in word sense disambiguation. http://arxiv.org/pdf/1909.03135v1

Linden, G., Smith, B. & York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, *7*(1), 76–80. https://doi.org/10.1109/MIC.2003.1167344

Lu, J., Wu, D., Mao, M., Wang, W. & Zhang, G. (2015). Recommender system application developments: A survey. *Decision Support Systems*, *74*, 12–32. https://doi.org/10.1016/j.dss.2015.03.008

Luong, H., Huynh, T., Gauch, S., Do, L. & Hoang, K. (2012). Publication Venue Recommendation Using Author Network's Publication History. In J.-S. Pan, S.-M. Chen & N. T. Nguyen (Hrsg.), *Lecture Notes in Computer Science. Intelligent Information and Database Systems* (Bd. 7198, S. 426–435). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28493-9_45

Maheshwari, A. (2018). Report on Text Classification using CNN, RNN & HAN - Jatana - Medium. *Jatana*. https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f

Mazumder, S. (2021). *What is Imbalanced Data | Techniques to Handle Imbalanced Data*. https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/

Medvet, E., Bartoli, A. & Piccinin, G. (2014). Publication Venue Recommendation Based on Paper Abstract. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence* (S. 1004–1010). IEEE.

https://doi.org/10.1109/ICTAI.2014.152

Melville, P. & Sindhwani, V. (2011). Recommender Systems, 829–838. https://www.ime.usp.br/~jstern/miscellanea/seminario/Melville1.pdf

MLNerds (14. Februar 2019). What is the difference between word2Vec and Glove ? *MachineLearningInterview.com*. https://machinelearninginterview.com/topics/natural-language-processing/what-is-the-difference-between-word2vec-and-glove/

Mohamed, M. H., Khafagy, M. H. & Ibrahim, M. H. (2019, 19.–21. Februar). Recommender Systems Challenges and Solutions Survey. In *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)* (S. 149–155). IEEE. https://doi.org/10.1109/ITCE.2019.8646645

Mooney, R. J. & Roy, L. (2000). Content-based book recommending using learning for text categorization. In P. J. Nürnberg, D. L. Hicks & R. Furuta (Hrsg.), *Proceedings of the fifth ACM conference on Digital libraries - DL '00* (S. 195–204). ACM Press. https://doi.org/10.1145/336597.336662

Naeem, A. (15. Juli 2021). What is sigmoid and its role in logistic regression? *Educative*. https://www.educative.io/answers/what-is-sigmoid-and-its-role-in-logistic-regression

Ogundepo, O. (23. Juli 2021). Understanding Word Representations with Word2Vec | by Odunayo Ogundepo | Jul, 2021 | Medium | Analytics Vidhya. *Analytics Vidhya*. https://medium.com/analytics-vidhya/understanding-word2vec-39fabe660705

Peiris, D. & Weerasinghe, R. (2015, 24.–26. August). Citation network based framework for ranking academic Publications and venues. In *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)* (S. 146–151). IEEE. https://doi.org/10.1109/ICTER.2015.7377681

Pham, M. C., Cao, Y. & Klamma, R. (2010). Clustering Technique for Collaborative Filtering and the Application to Venue Recommendation. In *International Conference on Knowledge Management and Knowledge Technologies,* Graz, Austria. https://www.researchgate.net/publication/228446479

Pradhan, T. & Pal, S. (2019). A hybrid personalized scholarly venue recommender system integrating social network analysis and contextual similarity. *Future Generation Computer Systems*, *110*, 1139–1166. https://doi.org/10.1016/j.future.2019.11.017

Ricci, F., Rokach, L. & Shapira, B. (Hrsg.). (2010). *Recommender Systems Handbook*. Springer.

Safa, R., Mirroshandel, S., Javadi, S. & Azizi, M. (2017). Venue Recommendation Based on Paper's Title and Co authors Network. *Journal of Information Systems*

and Telecommunication (JIST), *2018*, Artikel 21, 33–40. http://jist.ir/Article/139605221631287527

Sarkar, D. (2019). *Text Analytics with Python*. Apress. https://doi.org/10.1007/978-1-4842-4354-1

Schedl, M. & Hauger, D. (2015). Tailoring Music Recommendations to Users by Considering Diversity, Mainstreaminess, and Novelty. In R. Baeza-Yates, M. Lalmas, A. Moffat & B. Ribeiro-Neto (Hrsg.), *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (S. 947–950). ACM. https://doi.org/10.1145/2766462.2767763

scikit-learn. (2022a). *1.1. Linear Models*. https://scikit-learn.org/stable/modules/linear_model.html

scikit-learn. (2022b). *sklearn.feature_extraction*. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

scikit-learn. (2022c). *sklearn.feature_extraction.text.TfidfVectorizer*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

scikit-learn. (2022d). *sklearn.neural_network.MLPClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Scimago Journal & Country Rank. (2022). *Journal Rankings on Physics and Astronomy*. https://www.scimagojr.com/journalrank.php?area=3100

Shao, C. (2020). Data classification by quantum radial basis function networks. *Physical Review A*, *102*(4), Artikel 042418, 321. https://doi.org/10.1103/PhysRevA.102.042418

Shekhar, S. (2021). *LSTM for Text Classification | Beginners Guide to Text Classification.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/

Siddharth M (20. Juli 2021). Feature Extraction and Embeddings in NLP: A Beginners guide to understand Natural Language Processing. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/07/feature-extraction-and-embeddings-in-nlp-a-beginners-guide-to-understand-natural-language-processing/

Springer Nature. (2020). *Springer Journal Suggester*. https://journalsuggester.springer.com/#

Sun, J., Ma, J., Liu, X., Liu, Z., Wang, G., Jiang, H. & Silva, T. (2013). A Novel Approach for Personalized Article Recommendation in Online Scientific Communities. In *2013 46th Hawaii International Conference on System Sciences* (S. 1543–1552).

IEEE. https://doi.org/10.1109/HICSS.2013.48

Turcotte, J., York, C., Irving, J., Scholl, R. M. & Pingree, R. J. (2015). News Recommendations from Social Media Opinion Leaders: Effects on Media Trust and Information Seeking. *Journal of Computer-Mediated Communication*, *20*(5), 520–535. https://doi.org/10.1111/jcc4.12127

Wang, D., Liang, Y., Xu, D., Feng, X. & Guan, R. (2018). A content-based recommender system for computer science publications. *Knowledge-Based Systems*, *157*, 1–9. https://doi.org/10.1016/j.knosys.2018.05.001

Wei, J. (2020). ELMo: Why it's one of the biggest advancements in NLP. *Towards Data Science*. https://towardsdatascience.com/elmo-why-its-one-of-the-biggest-advancements-in-nlp-7911161d44be

Yang, Z [Zaihan] & Davison, B. D. (2012, 12.–15. Dezember). Venue Recommendation: Submitting Your Paper with Style. In *2012 11th International Conference on Machine Learning and Applications* (S. 681–686). IEEE. https://doi.org/10.1109/ICMLA.2012.127

Bisher erschienene Schriften

Ergebnisse von Forschungsprojekten erscheinen jeweils in Form von Arbeitsberichten in Reihen. Sonstige Publikationen erscheinen in Form von alleinstehenden Schriften.

Derzeit gibt es in den Churer Schriften zur Informationswissenschaft folgende Reihen:
Reihe Berufsmarktforschung

Weitere Publikationen

Churer Schriften zur Informationswissenschaft – Schrift 147
Herausgegeben von Wolfgang Semar
Regina Eicher
Die Entwicklung inhaltlicher Sprachbegriffe für eine verbesserte Erschliessung von Kinder- und Jugendzeichnungen
Eine qualitative Inhaltsanalyse von 12 ausgewählten Märchen
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 148
Herausgegeben von Wolfgang Semar
Andrej Kilian
"Die Bibliotheksthematik hat sich in den letzten Jahren stark relativiert"
Interne Bibliotheken in der Deutschschweiz und in Lichtenstein – Versuch eines Einblicks
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 149
Herausgegeben von Wolfgang Semar
Sandra Freiburghaus
Untersuchung von Anzeige- und Reservationssystemen zur Lernplatzorganisation in Bibliotheken
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 150
Herausgegeben von Wolfgang Semar
Nicole Fässler
User Adoption bei der Einführung einer Kollaborations- und Kommunikationssoftware im Modern Workplace Umfeld
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 151
Herausgegeben von Wolfgang Semar
Marina Inglin
Re- und Upskilling-Empfehlung. Kriterien für die automatische Auswahl von Re- und Upskilling-Angebote
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 152
Herausgegeben von Wolfgang Semar
Lisa Heller
Zur Genese eines nationalen Bibliotheksprojekts: Swiss Library Service Platform (SLSP)
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 153
Herausgegeben von Wolfgang Semar
Antonin Friberg
Die Effektivität von Social Media Norms Nudging in der Customer Journey
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 154
Herausgegeben von Wolfgang Semar
Curdin Marxer
«Drug Repurposing» Wie können unstrukturierte Textdaten für die Ermittlung neuer «Drug Repurposing:
nutzbar gemacht werden und wie können sie Datenbanken ergänzen?
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 155
Herausgegeben von Wolfgang Semar
Samir Limani
Sicht der administrativen Mitarbeitenden von Bündner Spitälern und Kliniken auf den
Digitalisierungsstand ihres Unternehmens
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 156
Herausgegeben von Wolfgang Semar
Marina Lea Schürmann
Deep Learning für Part-of-Speech-Tagging.
Vergleich eines auf Transformers basierenden POS-Taggers mit bestehenden Modellen
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 157
Herausgegeben von Wolfgang Semar
Yannick Mireille Kaufmann
Einsatz von Unternehmenswikis als Wissens-management-Tool in einer Netzwerkorganisation
Evaluationsstudie zu «wikimia», eine Wissensdaten-bank in der schweizerischen Berufs-, Studien- und
ratung Masterthesis 2022
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 158
Herausgegeben von Wolfgang Semar
Franco Malacrida
Standortfindung von Schweizer Start-ups
Welche Standortfaktoren sind für Schweizer Start-ups am wichtigsten?
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 159
Herausgegeben von Wolfgang Semar
Josip Spec
From ISAD(G) to Records in Contexts – A new era
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 160
Herausgegeben von Wolfgang Semar
Loris Haller
Gemeinwohl fördern als Geschäftsmodell
Kriterien für die Entwicklung eines Frameworks für gemein-wohlorientierte Geschäftsmodelle
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 161
Herausgegeben von Wolfgang Semar
Céline Graf
"Ghostbusters Münstergasse"
Vermittlung von regionalen Onlineressourcen und Recherchekompetenzen mit einem digitalen
Educational Escape Room an der Bibliothek Münstergasse der Universitätsbibliothek Bern
Chur, 2023
ISSN 1660-945X

Über die Informationswissenschaft der Fachhochschule Graubünden

Die Informationswissenschaft ist in der Schweiz noch ein relativ junger Lehr- und Forschungsbereich. International weist diese Disziplin aber vor allem im anglo-ameri-kanischen Bereich eine jahrzehntelange Tradition auf. Die klassischen Bezeichnungen dort sind Information Science, Library Science oder Information Studies. Die Grundfragestellung der Informationswissenschaft liegt in der Betrachtung der Rolle und des Umgangs mit Information in allen ihren Ausprägungen und Medien sowohl in Wirtschaft und Gesellschaft. Die Informationswissenschaft wird in Chur integriert betrachtet.

Diese Sicht umfasst nicht nur die Teildisziplinen Bibliothekswissenschaft, Archiv-wissenschaft und Dokumentationswissenschaft. Auch neue Entwicklungen im Bereich Medienwirtschaft, Informations- und Wissensmanagement und Big Data werden gezielt aufgegriffen und im Lehr- und Forschungsprogramm berücksichtigt.

Der Studiengang Informationswissenschaft wird seit 1998 als Vollzeitstudiengang in Chur angeboten und seit 2002 als Teilzeit-Studiengang in Zürich. Seit 2010 rundet der Master of Science in Business Administration das Lehrangebot ab.

Der Arbeitsbereich Informationswissenschaft vereinigt Cluster von Forschungs-, Entwicklungs- und Dienstleistungspotenzialen in unterschiedlichen Kompetenzzentren:

- Information Management & Competitive Intelligence
- Collaborative Knowledge Management
- Information and Data Management
- Records Management
- Library Consulting
- Information Laboratory
- Digital Education

Diese Kompetenzzentren werden im Swiss Institute for Information Science (SII) zusammengefasst.

Impressum

Impressum

FHGR - Fachhochschule
Graubünden
Information Science
Pulvermühlestrasse 57
CH-7000 Chur

www.informationscience.ch
www.fhgr.ch
ISSN 1660-945X

Institutsleitung

Prof. Dr. Ingo Barkow
Telefon: +41 81 286 24 61
Email: ingo.barkow@fhgr.ch

Sekretariat

Telefon: +41 81 286 24 24
Fax: +41 81 286 24 00
Email: clarita.decurtins@fhgr.ch