Verbesserung der automatischen Dokument-Klassifikation für den Discovery Service LIVIVO von ZB MED

Bachelorarbeit zur Erlangung des Bachelor-Grades
Bachelor of Science DIS (Data and Information Science)
an der Fakultät für Informations- und Kommunikationswissenschaften
der Technischen Hochschule Köln

vorgelegt von: Max Prantz

eingereicht bei: Prof. Dr. Konrad Förstner
Zweitgutachter: Prof. Dr. Klaus Lippert

Köln, 31.03.2023

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Rechtsverbindliche Unterschrift

Inhaltsverzeichnis

Erklärung ***1-1***

i Kurzfassung (Abstract) ***1-3***

ii Abbildungsverzeichnis ***1-4***

iii Tabellenverzeichnis ***1-4***

iv Abkürzungsverzeichnis ***1-5***

1 Einleitung ***6***

1.1 Problemstellung und Motivation 6

1.2 Methodik 6

1.3 Zielsetzung 7

2 Theoretische Grundlagen ***7***

2.1 Aktueller Stand des Systems 7

2.2 Datengrundlage 7

2.3 Klassifizierungsmodelle 9

2.3.1 Fehlerwerte 10

2.3.2 Latent Dirichlet Allocation (LDA) 10

2.3.3 Term Frequency – Inverse Document Frequency (TF-IDF) 11

2.3.4 Stochastic Gradient Descent Classifier (SGDC) 12

3 Umsetzung und Klassifizierungen ***12***

3.1 Explorative Analyse 13

3.2 Aufbau des Text Corpus 17

3.3 Erstellung des Korpus 19

3.4 Einteilung in Trainings und Validations Daten 22

3.5 Klassifizierungsmodelle 24

3.5.1 Latent Dirichlet Allocation (LDA) 24

3.5.2 TF-IDF & Stochastic Gradient Descent Classifier (SGDC) 25

4 Ergebnis-Diskussion & Wahl des optimalen Models ***27***

4.1 Ergebnisse 27

4.1.1 Ergebnisse des LDA-Modells 27

4.1.2 Ergebnisse des SGDC 30

4.2 Probleme der einzelnen Modelle / Methoden 31

4.3 Vergleich der Modelle 32

5 Zusammenfassung und Schlussfolgerung ***33***

6 Anhang ***35***

7 Literaturverzeichnis ***41***

i Kurzfassung (Abstract)

Dieser Arbeit beschreibt, wie eine Grundlage geschaffen wird, um die Dokumentenklassifikation der Suchmaschine LIVIVO durch eine Eigenentwicklung der ZB-Med zu ersetzen. Das bisher eingesetzte System basiert auf einer proprietären Software der Averbis GmbH und bietet keine Möglichkeit, diese von ZB-Med anpassen oder erweitern zu lassen. Damit die Klassifikation der Dokumente innerhalb der Datenbank, der Suchmaschine LIVIVO, verbessert werden kann, soll ein neues System entwickelt werden.

Um dieses Ziel erreichen zu können, konzentriert sich diese Arbeit auf eine explorative Analyse der vorhandenen Daten sowie auf die Erstellung erster Klassifikationsmethoden und den damit verbundenen Aufbau eines Textkorpus. Diese neu erstellten Methoden basieren auf existierenden Klassifikationsmodellen wie Stochastic Gradient Descent Classifier (SGDC), Term-Frequenzy Inverse-Document-Frequenzy (TF-IDF) und Latent Dirichlet Allocation (LDA). Die Ergebnisse dieser Modelle werden diskutiert und evaluiert.

Die erstellten Leistungskurven der Modelle und Textkorpi können somit als Vergleich, sowie Grundlage für weitere Arbeiten am System verwendet werden.

ii Abbildungsverzeichnis

Abbildung 3-1: Klassenverteilung innerhalb der Datenbank nach Averbis 13
Abbildung 3-2: Dokumentenhäufigkeit nach Datenbank sortiert 14
Abbildung 3-3: Klassenaufteilung anhand der Datenbanken AGRICOLA und MEDLINE 15
Abbildung 3-4: Publisher & Menge an Veröffentlichungen 16
Abbildung 3-5: Menge an Veröffentlichungen ohne Publisher-Eintrag, soriert nach Sektor 17
Abbildung 3-6: Python-Module für die Vorverarbeitung und Tokenisierung der Dokumente 20
Abbildung 3-7: Python Funktion als Beispiel für die Prozess-Parallelisierung 20
Abbildung 3-8: Python Funktion für die Lemmatisierung und Tokenisierung der Dokumente 21
Abbildung 3-9: Python Code mit Funktionen zum Datenbankstreaming und vollständiger Vorverarbeitung der Dokumente 21
Abbildung 3-10: Python Beispiel für das Hochladen des fertiggestellten Textkorpus, für jedes Dokument 22
Abbildung 3-11: Python Code zur Textkorpusabfrage der Dokumente nach Klassen 22
Abbildung 3-12: Python Code zur Zusammenlegung der einzelnen Datenbankspalten, sowie die Filterung gänzlich leerer Einträge 23
Abbildung 3-13: Pythoncode zur Aufteilung des Trainingdatensatzes nach Klasse 23
Abbildung 3-14: Python Code zur Zählung der verfügbaren Dokumentenanzahl im Datensatz 23
Abbildung 3-15: Veranschaulichter Python Code indem die Klassenmengen ausgeglichen werden und der Datensatz in Test und Training aufgeteilt wird 24
Abbildung 3-16: Veranschaulichter Python Code indem die Schleife des LDA Trainings beschrieben wird 24
Abbildung 3-17: Veranschaulichter Python Code in dem die Schleife für das Training und die Validierung des SGDC beschrieben wird 25
Abbildung 4-1 Vergleich der Averbis Themenanzahl und LDA Topic Zuweisungen bei maximal 4 möglichen Topics 28
Abbildung 5-1: Aufteilung der Daten sortiert nach Dokumenten mit oder ohne Publisher, der gesamten Datenbank 35
Abbildung 6-2: Publisheranzahl und Menge der Veröffentlichungen 36
Abbildung 6-3: LDA-Klassifizierungsgenauigkeit bei 12 Topics 37
Abbildung 6-4: LDA-Klassifizierungsgenauigkeit bei 24 Topics 37
Abbildung 6-5: LDA-Klassifizierungsgenauigkeit bei 8 Topics 38
Abbildung 6-6: Confusion-Matrix des LDA Modells 38
Abbildung 6-7: Precision des SGD-Classifiers pro Klasse 39
Abbildung 6-8: Recall des SGD-classifiers pro Klasse 39
Abbildung 6-9: Python-Code zur gleichmäßigen Aufteilung der Daten in Test/Train 40
Abbildung 6-10: Python Code zum Training & Validierung des SGDC Modells 40

iii Tabellenverzeichnis

Tabelle 1 Relevante Inhalte der LIVIVO Datenbank zur Analyse und Klassifizierung 8
Tabelle 2 Veranschaulichung der möglichen Klassenaufteilung bei maximal acht Topics 27
Tabelle 3 Veranschaulichung der möglichen Klassenaufteilung bei maximal vier Topics 27

iv Abkürzungsverzeichnis

SQL	Structured Query Language
TF-IDF	Term Frequency Inverse Document Frequency
SGDC	Stochastic Gradient Descent Classifier
SVM	Support Vector Machine
LDA	Latent Dirichlet Allocation
NLTK	Natural Language Toolkit
ZB-Med	Deutsche Zentralbibliothek für Medizin
VM	Virtuelle Maschine
CPU	Prozessor

1 Einleitung

1.1 Problemstellung und Motivation

Im Rahmen der Literaturdatenbank ZB-Med werden Publikationen bereitgestellt, um den Forschenden einen schnellen und informativen Überblick über verschiedenste Themen zu ermöglichen. Damit die gespeicherte Literatur bei einer Recherche schnell und nach Relevanz gut einschätzbar darstellt werden kann, werden die Texte bzw. Abstracts und Titel der Publikationen klassifiziert. So werden die Dokumente derzeit mit einer Software eines Drittanbieters (Averbis GmbH) (Averbis GmbH 2022) in mehrere übergeordnete Themenbereiche eingeteilt und mit einem Symbol versehen. Dies soll den Suchenden helfen, schnell zu erkennen, ob das gefundene Dokument grob in das gewünschte Themengebiet fällt. Die vier Hauptkategorien sind Medizin, Ernährung, Umweltwissenschaften und Landwirtschaft.

Aufgrund der proprietären Natur der derzeitigen Software ist es nicht möglich, Einfluss auf die Kategorisierung der Texte zu nehmen. Einstellungen, die die Software positiv beeinflussen könnten, sind daher nicht möglich. Mit einer in ZB-Med entwickelten Software könnte eine genauere, schnellere, flexiblere und kostengünstigere Lösung geschaffen werden.

1.2 Methodik

Die Umsetzung des Projekts beinhaltet mehrere Lösungswege. Bei jedem dieser ist unklar, ob das gewünschte Ergebnis, einer genauen Klassifizierung, erzielt wird. Eine mögliche Kombination ist es ebenfalls möglich mehrere dieser Lösungswege zu Kombinieren. Der endgültige Aufbau der Pipeline ist somit noch nicht festgelegt und wird ggf. während der Umsetzung mehrfach verändert.

Als „Golden Record“ oder Benchmark wird die bereits vorhandene Klassifikation verwendet. Da diese Klassifikation nicht perfekt ist, wird als Ziel angestrebt, die Dokumente gleich gut oder besser klassifizieren zu können, als die bisher verwendete Software. Damit dieses Ziel erreicht werden kann, müssen nach dem Laden und Bauen eines Datensets aus der Datenbank von ZB-Med mehrere Methodiken verwendet werden.

Aus dem entstandenen Text-Corpus der Datenbank werden Titel und Abstracts in Trainings- und Validierungsdatensatz bzw. Testdatensatz unterteilt. Die Klassifizierungsmodelle mit denen gearbeitet wird sind Latent Dirichlet Allocation (LDA) und Stochastic Gradient Descent Classifier (SGDC), beruhend auf Basis einer Support Vector Machine (SVM) mit Vorfilterung eines TF-IDF (Term Frequenzy – Inverse Document Frequenzy) Modells. Die Leistungsfähigkeit und Qualität der Modelle werden anhand der Zuteilungsgenauigkeit und deren Menge gemessen. Diese werden in Graphen und anhand von F1-Scores dargestellt. Der für diese Arbeit entwickelte Code wird in einem GitHub Repository gespeichert und zur Weiterverwendung zur Verfügung gestellt.

1.3 Zielsetzung

Das Ziel der Arbeit ist es eine Grundlage für die Entwicklung einer hauseigenen Software zu schaffen, welche für die Themenklassifizierung innerhalb der LIVIVO Datenbank von ZB-Med Anwendung finden kann. Es sollen erste Klassifizierungsmodelle erstellt und die Datenbank explorativ auf ihre Eigenschaften untersucht werden. Die Inhalte der Datenbank sollen hingehend ihrer Eignung für Klassifikationsmodelle beurteilt und deren mögliches Potential diskutiert werden. Mit der Erstellung erster Klassifikationsmodelle soll ein Maß erschaffen werden, an denen sich zukünftige Projekte orientieren können. Die Erstellung der Modelle führt auch eine Entwicklung für eine Datenreinigungs- und Verarbeitungs-Pipeline mit sich. Die Ergebnisse der Modelle und der explorativen Analyse werden in dieser Arbeit festgehalten und diskutiert, der entwickelte Code wird in meinem GitHub Repository gespeichert und dokumentiert.

2 Theoretische Grundlagen

2.1 Aktueller Stand des Systems

Die Suchmaschine LIVIVO von ZB-Med ist eine der größten (lebenswissenschaftlichen) Suchmaschinen Europas. Sie stellt Informationen, Forschungsdaten und Literatur für Lebenswissenschaften bereit. Die Datenbank vereint die Fachgebiete Medizin, Gesundheitswesen, Ernährungs- Umwelt und Agrarwissenschaften (LIVIVO - Help - About LIVIVO 2023). Um diese Daten bei einer Suchanfrage übersichtlich und korrekt deklariert darstellen zu können sind, bei einer Suchanfrage, die einzelnen Dokumente mit Identifizierungsmerkmalen versehen. Diese Merkmale werden momentan (2023) von einer Dritt-software klassifiziert. Der Herausgeber der Software ist die Firma Averbis GmbH. Als Grundlage des Klassifizierungssystem von Averbis wird ein sogenannter „Morphosauris“ benutzt, das Ziel dieses Systems war ursprünglich die Verarbeitung und Wiederfindung von medizinischen Dokumenten und ist im Zusammenspiel mit der Deutschen Nationalbibliothek für Medizin entstanden. (Waldemar Dzeyk 2008)

2.2 Datengrundlage

Der bereitgestellte Datensatz besteht aus zwei Grundbausteinen. Der Hauptteil des Datensatzes beinhaltet verschiedenste Informationen über die verfügbaren Publikationen und wird mithilfe von „PostgreSQL“, einer Abwandlung von SQL[1], gespeichert und bearbeitet. Der zweite Bestandteil ist eine über IDs, verknüpfte Tabelle mit den schon bestehenden Klassifikationen von Averbis. Zusammen bilden diese Teile den gesamten Datensatzmit einer Speichergröße von ca. 64GB.
Um diese Daten verarbeiten zu können muss im Voraus eine explorative Analyse gestartet werden, in der der Aufbau der Datenbank, sowie der Inhalt genauer betrachtet werden. Durch das erlangte wissen kann dann das weitere vorhergehen bestimmt werden.

[1]SQL ist eine Standardsprache für die Erstellung und Bearbeitung von Datenbanken. MySQL ist ein relationales Datenbankprogramm, das SQL-Abfragen verwendet.

Die relevanten Daten, die für eine Klassifizierung und explorative Analyse benötigt werden, sind in den folgenden Tabelleneinträgen gespeichert:

Tabelle 1 Relevante Inhalte der LIVIVO Datenbank zur Analyse und Klassifizierung

Spaltenname	**Beschreibung**	**Datentyp / Beispiel:**
DBrecordid	Ein einzigartiger Schlüssel, über den jeder Eintrag im Datensatz identifiziert werden kann. Dieser ist wichtig, um später den Publikationen ihre neue Klassifikation zu zuweisen	String / M4216209
Doctype	Die Art der Veröffentlichung, unterschieden wird zwischen: Online-Artikeln, Artikeln und anderen	String, Liste / {ARTIKEL,ONLINE}
Class	Die Einstufung von Averbis in 5 Kategorien: Medizin, Umweltwissenschaften, Landwirtschaft, Ernährung, Rest	String / Medizin Landwirtschaft Umweltwissenschaften Ernährung Rest
Source	Gibt an, in welchem Journal das Dokument veröffentlicht wurde	String / Journal of Neurosurgery
Collectiontitle	Titel der Sammlung, aus dem der Eintrag stammt, sofern vorhanden	String / -
Publisher	Das Institut, Person oder Verlag, die den Eintrag veröffentlicht hat	String / International Food Policy Research Institute (IFPRI)
Title	Titel der Veröffentlichung	String / Root and tuber crops in Guadeloupe
Abstract	Prägnante Zusammenfassung der Veröffentlichung	String / The main root crops grown in […].
Keywords	Stichwörter zu Veröffentlichung, von früheren Klassifizierungen vergeben	String, Liste / ["Food prices; Biofuels; Agricultural subsidies; Agricultural development"]
Institution	Vorhanden, wenn das Paper durch eine Institution veröffentlicht wurde	String / -
Database	Ursprung der Daten	String / AGRIS
Subdatabase	Falls eine Datenbank in mehre Themen aufgeilt wurde, ist der Ursprung der Daten hier granularer aufgelistet	String / -

2.3 Klassifizierungsmodelle

Textklassifizierungsmodelle werden in der Textverarbeitung benutzt, um automatisiert Kategorien, Klassen oder Label einem Dokument zuweisen zu können. Die Kategorien, Klassen oder Label können verschiedenster Natur sein und dem Betrachter Hinweise auf die Stimmung des Autors geben, bzw. Textbausteinen einem bestimmten Thema zuordnen. Ziel ist es, meist große Datenmengen schnell analysieren und verarbeiten zu können, um zum Beispiel Spamfilter, Stimmungsanalysen oder Textübersetzungen zu ermöglichen. Die Vielfalt der Modelle ist enorm und reicht von „einfachen" statistischen Berechnungen bis hin zu komplexen neuronalen Netzen. Diese Modelle unterscheiden sich in vielen ihrer Eigenschaften, potentiellen Leistungsfähigkeit, Anwendungsfeldern und benötigten Vorarbeiten am Datensatz, sowie die minimale Anzahl an Dokumenten die zum Trainieren benötigt werden. (Text Classifiers in Machine Learning: A Practical Guide 2023; Lai et al. 2015)

Die einfachsten und schnellsten Modelle sind statistischer Natur, wie zum Beispiel die Support Vector Machine, die auch bei großen Datenmengen noch eine vergleichbar schnelle Verarbeitung ermöglicht. Die Größe des Trainingdatensatzes muss, wie im Abschnitt 4.1.2 Ergebnisse des SGDC beschrieben wird, nicht besonders groß sein, um schon gute und zuverlässige Ergebnisse zu liefern. Allerdings korreliert die Qualität der Ergebnisse stark mit der des Datensatzes. Auch die Aussagekraft der verwendeten Daten spielt hier eine Rolle, wenn wichtige Merkmale markanter sind, kann jedes Modell in der Regel auch bessere Ergebnisse liefern. Neuronale Netze hingegen brauchen oftmals keine aufwändige Textvorverarbeitung und können so mit verhältnismäßig wenig Aufwand schon gute Ergebnisse liefern (Text classification 2023). Dafür brauchen diese Modelle jedoch deutlich größere Datensätze und längere Trainingszeiten, um klassifizierungsrelevante Merkmale und Strukturen erkennen zu können. Der Ressourcen-Aufwand ist bei neuronalen Netzen somit deutlich höher, als der einer logistischen Regression kombiniert mit einer TF-IDF Gewichtung.

Es gilt für jedes Projekt also abzuwägen welcher Ansatz der richtige ist. Wenn Ressourcen, Trainingsdaten und Trainingszeiten im hohen Maße vorhanden sind, kann es Sinn ergeben neuronale Netze zu trainieren. Wenn die Datenlage, Zeit und Ressourcen begrenzt sind, ist es meist sinnvoller, einfachere Klassifikationsmethoden anzuwenden. Falls der Dateninhalt jedoch noch nicht erforscht oder eingegrenzt ist, lohnt es sich zuerst einen Überblick mithilfe einer explorativen Analyse und einer Themenmodellierung à la LDA Überblick zu verschaffen. (Blei et al. 2003)

Unterschieden werden muss allerdings zwischen zwei grundsätzlich verschiedenen Arten von Machine-Learning-Methoden: „Supervised Learning" und „Unsupervised Learning".

Unsupervised Modelle sind Machine Learning Verfahren, die Daten verarbeiten können, ohne dabei auf vorher festgelegte Label oder Kategorien zurückgreifen zu müssen. Das unsupervised Modell erkennt von selbst Muster, Eigenschaften und Zusammenhänge. Anhand dieser kann das Modell Daten in selbst erstellte Kategorien klassifizieren.

Supervised Modelle werden anhand von Trainingsdaten trainiert und mit einem Testdatensatz auf die Richtigkeit der Zuteilung überprüft. Die Trainingsdaten bestehen, sowie die Testdaten, aus einer Sammlung von Merkmalen und Eigenschaften, die zum Beispiel mit einer Kategorie verknüpft sind. Mithilfe der Trainingsdaten wird das Modell so trainiert, dass es anhand der Merkmale und Eigenschaften eine Kategorie erkennt. Der Testdatensatz wird nach dem Trainieren verwendet, um zu überprüfen mit welcher Treffsicherheit das zuvor trainierte Modell, die Kategorien den Daten zugewiesen wird.(Supervised vs. Unsupervised Learning: What's the Difference? 2021)

2.3.1 Fehlerwerte

Nachdem die Modelle mit einem Trainingsdatensatz trainiert wurden, werden diese auf einen Testdatensatz angewandt. Das bedeutet, dass das zuvor trainierte Modell die Input-Layer aus dem Testdatensatz interpretiert und den Dokumenten, aus den zuvor definierten Klassen, eine Klasse zuteilt. Da im Testdatensatz auch die Klassifikationen des Golden Records gespeichert sind kann überprüft werden, wie gut das Modell funktioniert.

Für die Überprüfung gibt es mehrere Kennzahlen, die relevanteste ist der sogenannte „F1-Score". Dieser Score setzt ist aus weiteren Kennzahlen zusammengesetzt und bildet somit das beste Gesamtergebnis ab. Die kombinierten Maße für die Errechnung des F1-Score sind die „Präzision" und der „Recall", welche direkt aus dem Testdatensatz errechnet werden können. Die Formel aus dem sich die F1-Score zusammensetzt ist:

$$F1 = 2 \times \frac{Präzision \ \times \ Recall}{Präzision + Recall}$$

Formel 2-1 Formel zur Errechnung der F1-Score eines Modells

Der Recall, auch „True Positive Rate oder Sensitivität", stellt einen Wert dar, der widerspiegelt wie viele positive Fälle das Model korrekt als positiv klassifiziert hat. Wenn das Modell ein Dokument mit der Klasse „Medizin" klassifizieren würde und der Golden Record dasselbe Dokument ebenfalls mit der Klasse „Medizin" abgespeichert hätte, wäre dieses Dokument ein „True Positive"-Fall. Ein „False Negative"-Falle wäre eine Fehlklassifizierung des Modells, dementsprechend würde sich die zugeteilte Klasse des Modells vom Golden Record unterscheiden. Bei nicht binären Klassifikationsproblemen, wie bei diesem, wird daher eine sogenannte „Confusion-Matrix" erstellt, in dem jede Klasse als binärer Wert behandelt wird. Dies spiegelt sich in den Ergebnissen wider, da jede der vier Klassen eine eigene F1-Score zugewiesen bekommt. Dementsprechend muss auch für jede Klasse der Recall und die Präzision bestimmt werden.
Die Formel zu Errechnung des Recalls setzt sich folglich aus den Werten der True und False Negatives zusammen.

$$\text{Recall} = \frac{True\ Positive}{\text{True Positive} + \text{False Negative}}$$

Formel 2-2 Formel zur Errechnung des Recalls eines Modells

Die zweite Evaluationsmetrik für die Errechnung des F1-Scores ist die Präzision, diese wird anhand der True Positives und False Positives bestimmt. Ein False Positive Fall wäre innerhalb der Confusion Matrix ein Dokument, welches nicht von Averbis als medizinisch klassifiziert wurde, das Modell aber vorhersagt, dass es zur Klasse Medizin gehört. Um der Präzision in ein verständliches Maß umzuwandeln wird folgende Formel benutzt:

$$\text{Präzision} = \frac{True\ Positive}{\text{True Positive} + \text{False Positive}}$$

Formel 2-3 Formel zur Errechnung der Präzision eines Modells

Anhand dieser Werte lässt sich die Leistung eines Modells sehr gut messen, die Leistung eines Modells ist also sehr gut, wenn der F1-Score möglichst nah am Wert eins liegt. Ein sehr ungenaues Modell tendiert dementsprechend Richtung null. (Oppermann 2021)

2.3.2 Latent Dirichlet Allocation (LDA)

LDA (Latent Dirichlet Allocation) ist eine probalistische Methode, um Texte zu klassifizieren, diese wird häufig verwendet, um Themen aus einer großen Sammlung von Dokumenten zu extrahieren. Dabei handelt es sich nicht um ein klassisches Zuweisen von Labeln, wie bei „supervised" Modellen, sondern um ein „unsupervised" Verfahren, welches aus der Dokumentensammlung eigene Themen erstellt.

Das Modell basiert auf der Annahme, dass jedes Dokument eine Sammlung von Themen ist, wobei jedes Thema eine bestimmte Verteilung von Wörtern hat. Die Themen, bzw. deren Wortverteilung wird im LDA-Modell mit Hilfe einer Bayesain-Statistik errechnet. Diese Wahrscheinlichkeitsverteilungen werden dann benutzt, um die Klassifizierung der Texte durchzuführen.

Dieses Vorgehen bringt mehrere Vorteile mit sich. Die Dokumente, die klassifiziert werden sollen, müssen zuvor nicht bekannt sein bzw. schon mit einem Label/einer Klasse versehen sein, um das Modell zu trainieren. So kann in der Theorie LDA dafür genutzt werden, aus gänzlich unbekannten Dokumenten Themen und eine Struktur zu extrahieren. Mit „supervised" Modellen ist dieses Vorgehen nicht möglich. Daher kann sich LDA auch gut für eine explorative Datenanalyse eignen, bei der nicht ein Ziel festgelegt ist. Auch sind die Ergebnisse leicht zu interpretieren, denn die Themen und deren Verteilung wird mit den dazugehörigen Wörtern dargestellt.

Der größte Vorteil von LDA ist allerdings ein Schwachpunkt des „unsupervised" Klassifizierungsverfahren. Durch das Fehlen von zuvor festgelegten Klassen und einem Trainings-/Testdatensatz können die Ergebnisse des LDA-Modells nur schwer validiert bzw. bewertet werden. Die Vergleichbarkeit zu anderen Modellen ist somit nur schwer möglich.
Bevor das Modell jedoch trainiert werden kann, muss der Datensatz gründlich bereinigt werden. Dazu gehört das Filtern von Stoppwörtern, Lemmatisierung oder Stemming und das Erstellen von Token aus den einzelnen Worten. Sobald der Textkorpus erstellt ist, kann das LDA Modell trainiert werden, jedoch sind die Trainings und Berechnungszeiten verhältnismäßig lang.

Ein LDA-Modell könnte im Falle der LIVIO Datenbank interessante Einblicke liefern und die Möglichkeit zu einer feineren und genaueren Klassifizierung öffnen. Die Qualität und Vollständigkeit der Daten werden einen großen Einfluss auf die Genauigkeit und Leistungsfähigkeit des Modells haben. Auch wird die Vergleichbarkeit zu anderen trainierten Modellen nicht einfach, da sich das LDA-Modell nicht an den vorgegebenen Klassen von Averbis orientieren wird. (Blei et al. 2003)

2.3.3 Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF steht für „Term Frequency – Inverse Document Frequency". Dies ist eine Methode zur Relevanzbestimmung eines Wortes in einem oder mehreren Textdokumenten.
TF-IDF besteht aus zwei Methoden, die Erste ist die Termfrequenz. Die Termfrequenz (TF) gibt an, wie oft ein Wort in einem Dokument vorkommt. Die zweite Rechenmethode ist, die der inversen Dokumentenfrequenz, diese gibt an, wie häufig das Wort in der gesamten Sammlung des Dokumentes vorkommt.
Das Ziel von TF-IDF ist es, seltene Wörter in Dokumenten überproportional zu gewichten, um so bessere Aussagen über den Inhalt machen zu können oder Schlüsselwörter extrahieren zu können.

Durch die Verwendung von TF-IDF in einer Klassifizierungspipeline, könnte das Ergebnis verbessert werden, da so Schlüsselwörter und andere seltene, aber ausschlaggebende Begriffe mehr gewichtet werden. Somit wird diesen wichtigen Wörtern eine höhere Relevanz zugeordnet und können somit den fachlichen Inhalt eines Dokumentes besser abbilden.

Die Methodik des TF-IDF ist weit verbreitet und findet in vielen Anwendungen Gebrauch, vor allem profitieren Suchmaschinen von der Verwendung. Es kann mithilfe dieser Methodik mit nur wenigen Stichworten oft schon ein grobes Thema aus Dokumenten herauskristallisiert werden. Gerade bei großen Datenmengen ist eine Reduzierung auf die wichtigsten Merkmale mit einem Leistungsgewinn verbunden. (TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSci 2023)

2.3.4 Stochastic Gradient Descent Classifier (SGDC)

SGDC steht für „Stochastic Gradient Descent Classifier“ und ist ein erweitertes Klassifizierungsmodell, welches unter anderem auf einer logistischen Regression oder einer SVM (Support Vector Machine) aufbauen kann. Der Unterschied zu einer logistischen Regression oder einer SVM ist der Zusatz des stochastischen Gradientenabstieg. Dieser Zusatz optimiert die zugrunde liegenden Funktionen, wie z.B. die logistische Regression oder eine SVM. (scikit-learn 2023b)

Eine Support-Vector-Machine (SVM) kann zur Lösung nichtlinearer Probleme eingesetzt werden. Dabei werden die Eingabedaten auf einen höherdimensionalen Raum abgebildet, in dem eine Trennlinie gefunden werden kann. Bei einem höherdimensionalen Raum handelt es sich um einen Raum mit mehr als den üblichen drei Dimensionen (Länge, Breite und Höhe), der durch die Verwendung zusätzlicher Dimensionen definiert wird, die über die drei traditionellen Dimensionen hinausgehen.

Durch die Projektion der Daten in einen höherdimensionalen Raum ist die SVM in der Lage, eine Trennlinie zu finden, die die verschiedenen Klassen in einem Datensatz möglichst gut voneinander trennt. Diese Trennlinie wird so gewählt, dass der Abstand zwischen den nächstgelegenen Beobachtungen in jeder Klasse (auch "Rand" genannt) maximiert wird.

Sobald die SVM trainiert wurde, kann sie verwendet werden, um neue Daten vorherzusagen, indem sie bestimmt, wo die neuen Datenpunkte relativ zu der Trennlinie in einem höherdimensionalen Raum liegen. Wenn eine neue Datenprobe auf die eine oder andere Seite der Trennlinie fällt, ist die SVM in der Lage, die Zugehörigkeit dieser Datenprobe zu einer Klasse zu bestimmen. Die SVM-Technik ist besonders effektiv bei der Lösung komplexer Klassifizierungsprobleme mit großen Datenmengen, da sie robust und schnell ist und auch dann noch gute Ergebnisse liefert, wenn die Eingabedaten eine sehr hohe Anzahl von Dimensionen aufweisen. Diese Eigenschaften sollten die SVM zu einem geeigneten Kandidaten für das vorliegende Klassifizierungsproblem machen. (Saxena 2021)

Das Klassifizierungsmodell (SGDC) ist ein sogenanntes „supervised learning“ Modell und kann daher nur trainiert werden, wenn es einen Datensatz gibt, der schon mit Labeln bzw. Klassen markiert wurde. Benutzt wird daher der „Golden Record“ von Averbis. Mit den vordeklarierten Daten kann ein Trainingsdatensatz aufgebaut werden an dem das SGD-Modell trainiert wird.

3 Umsetzung und Klassifizierungen

In diesem Abschnitt wird die Zusammensetzung der Datenbank und welche Teile dieser verwendet wurden beschrieben. Es wird, mit Blick auf die Averbis-Klassen, darauf eingegangen ob Strukturen und Muster schon mit einfachen Analyseschritten erkannt werden können.
Behandelt wird auch, wie genau der Textkorpus aufgebaut ist und welche Klassifizierungsmodelle benutzt wurden. Anhand von Code-Beispielen wird gezeigt wie diese trainiert und ausgeführt wurden. Verwendet werden die im Abschnitt 2 beschriebenen Modelle, LDA und SGDC. Die Trainierten Modelle werden anhand des F1-Maßes bewertet und beschrieben. Probleme, sowie mögliche Lösungsansätze, werden diskutiert.

3.1 Explorative Analyse

Anhand des erlangten Wissens über die Struktur der Datenbank kann nun zuerst ermittelt werden, wie viele Publikationen von Averbis in die jeweiligen Überthemen eingeordnet wurden, mit einer Abfrage nach der Anzahl der Einträge in der Datenbank und deren Zugehörigkeit zu einem der fünf Hauptthemen (Abbildung 3-1).

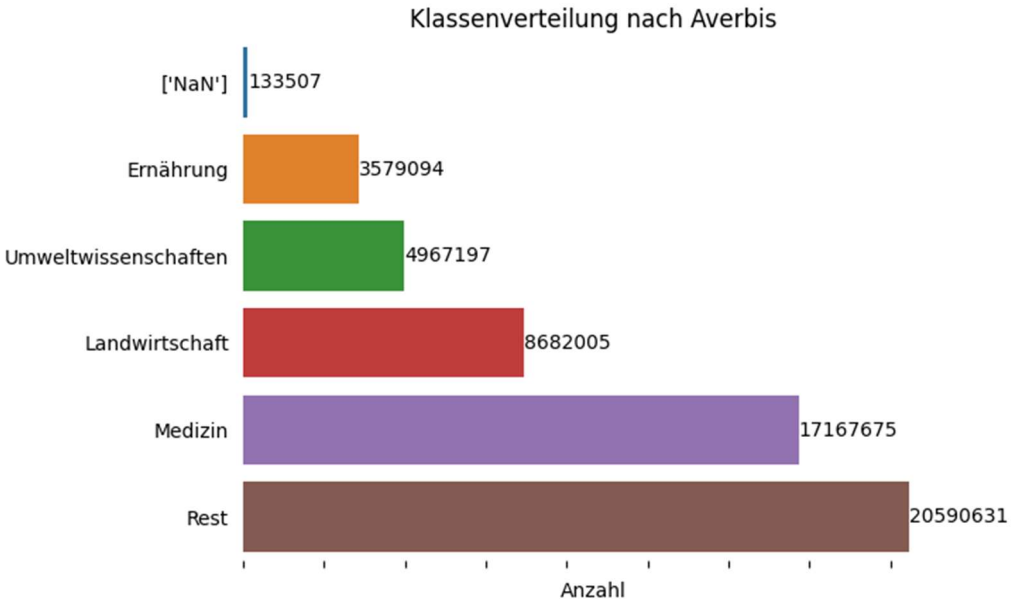

Abbildung 3-1: Klassenverteilung innerhalb der Datenbank nach Averbis

Wie in der daraus entstandenen Abbildung zu erkennen ist, die Datenbank in sechs Kategorien eingeteilt. Der Großteil der Daten ist mit der Klasse „Rest“ eingeteilt worden, mit 20.590.631 Einträgen. Die nächstgrößere Kategorie ist „Medizin“ mit, 17.167.675 Einträgen, gefolgt von „Landwirtschaft“ mit 8.682.005 Einträgen. „Umweltwissenschaften“ und „Ernährung“ stellen einen kleineren Teil der Datenbank dar, mit 4.967.197 und 3.579.094 Einträgen. Auffällig ist jedoch auch ein kleiner Teil der Datenbank, der in keine der Kategorien eingeteilt wurde und in der Datenbank ohne jegliche Information in der Spalte „Class“ versehen ist. In Relation zur Größe des Datensatzes ist der kategorisierte Teil mit gerade einmal 133.507 Einträgen ein sehr geringer Prozentsatz von 0,2443%. Die Gesamtanzahl aller Einträge der Datenbank beläuft sich also auf 54.589.109.

Des Weiteren ist auch die Menge an Veröffentlichungen in den Ursprungsdatenbanken interessant, hier könnten Regelmäßigkeiten zwischen Thema und Ursprungsdatenbank versteckt sein. Wenn aus einer der Ursprungsdatenbanken immer nur für ein Thema veröffentlicht wird, kann daraus schon die erste Regelmäßigkeit für eine Klassifizierungsmethode abgeleitet werden, bevor mit komplizierten und leistungshungrigen Algorithmen gerechnet wird (Abbildung 3-2).

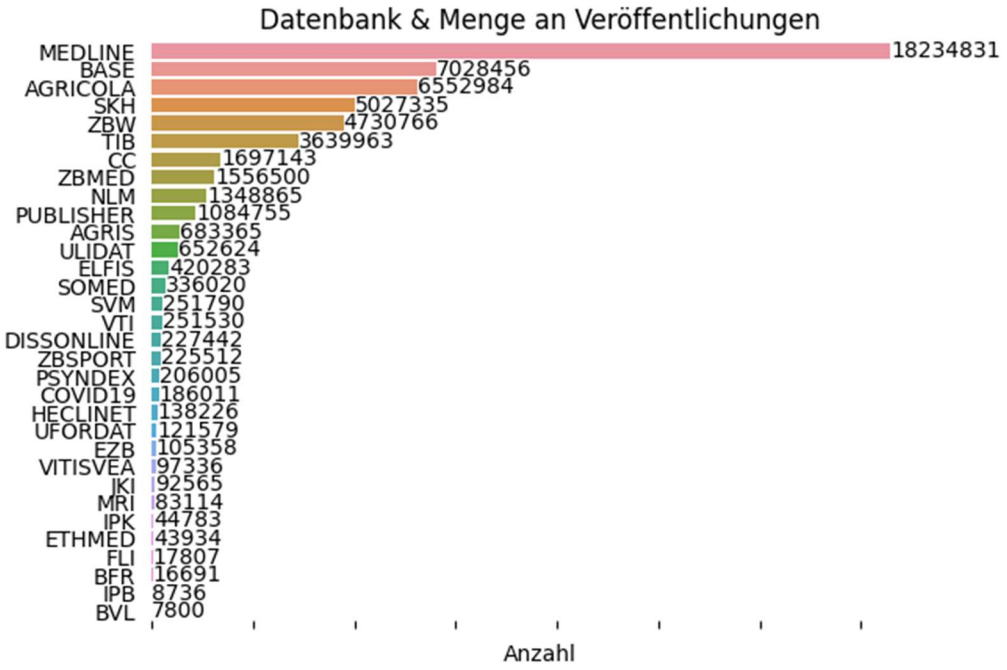

Abbildung 3-2: Dokumentenhäufigkeit nach Datenbank sortiert

Zunächst wird die Aufteilung der Veröffentlichungen aus den jeweiligen Ursprungsdatenbanken betrachtet. Der größte Anteil an Veröffentlichungen, auf den LIVIVO zugreifen kann, entspringt der Datenbank „MEDLINE“. Diese ist mit knapp 18,25 Millionen Einträgen die mit Abstand größte Datenbank. Die nächstgrößere Quelle von Daten stellt „BASE“ dar, mit ca. 7 Millionen Einträgen, dicht gefolgt von „AGRICOLA“ mit ca. 6,5 Millionen Daten. Anhand der ersten drei großen Datenbanken ist die Möglichkeit für eine Klassifizierung nach dem Ursprung der Daten also denkbar. Vor allem mit Blick auf den Hauptschwerpunkt der Datenbanken „MEDLINE“ und „AGRICOLA“.

Denn „MEDLINE“ oder auch „MEDical Literature Analysis and Retrieval System OnLINE [...] ist die Metadatenbank der US-amerikanischen Zentralbibliothek für Medizin National Library of Medicine (NLM) und enthält […] Nachweise wissenschaftlicher internationaler Zeitschriftenartikel aus dem Bereich der Lebenswissenschaften mit Schwerpunkt Biomedizin […]“ (ZB MED - Informationszentrum Lebenswissenschaften 2023), somit sollten die Publikationen dieser Datenbank größtenteils oder ausschließlich medizinischer Natur sein.

„AGRICOLA“ hingegen beinhaltet „[…] den Zugang zu Informationen über Landwirtschaft und verwandte Wissenschaftsbereiche. Die Sammlung der NAL beinhaltet mehr als 3,5 Millionen Titel, die das gesamte Spektrum der Landwirtschaft abdecken.“ (LIVIVO - Informationen über Datenquellen 2023)
Die Annahme, dass die Datenbanken tatsächlich ausschließlich Artikel bzw. Publikationen in ihrem Hauptfachgebiet gespeichert haben, ist allerdings ein Trugschluss. Wenn auch der Hauptbestandteil der Publikationen im angegebenen Fachgebiet liegen, so wurde zumindest mit Hilfe der Klassifikation von Averbis, welche unserer „Golden Record“ ist, eine so diverse Verteilung angelegt, die es nicht möglich macht, Klassen rein nach Datenbanken zu klassifizieren.

Die Sortierung der Daten nach Datenbankursprung und zugeteilten Klassen lässt sich gut in der Abbildung 3-3 erkennen. So wäre nach einer fixen Klassifizierung nach Ursprungsdatenbank bei Einträgen der „MEDLINE“-Datenbank etwa die Hälfte (ca. 9.8 Millionen von ca. 18,23 Millionen) aller zugeteilten Klassen falsch, vorausgesetzt jeder Eintrag würde die Klasse „Medizin“ zugeteilt bekommen. Im Falle der Datenbank „AGRICOLA“ wäre nach einer fixen Zuteilung nach „Landwirtschaft“ sogar mehr als die Hälfte falsch klassifiziert (ca. 2,6 Millionen von ca. 6,55 Millionen. Daraus lässt sich schließen, dass sich diese simple Methode zur Klassifizierung nicht eignet. Der Ursprung der Daten wäre also nur als weitere Information bzw. als weiterer „Input-Layer“ für eine komplexere Klassifizierungsmethode nützlich, allerdings auch nur mit einer geringen Gewichtung um die Ergebnisse möglichst „Bias[2]“-frei zu lassen.

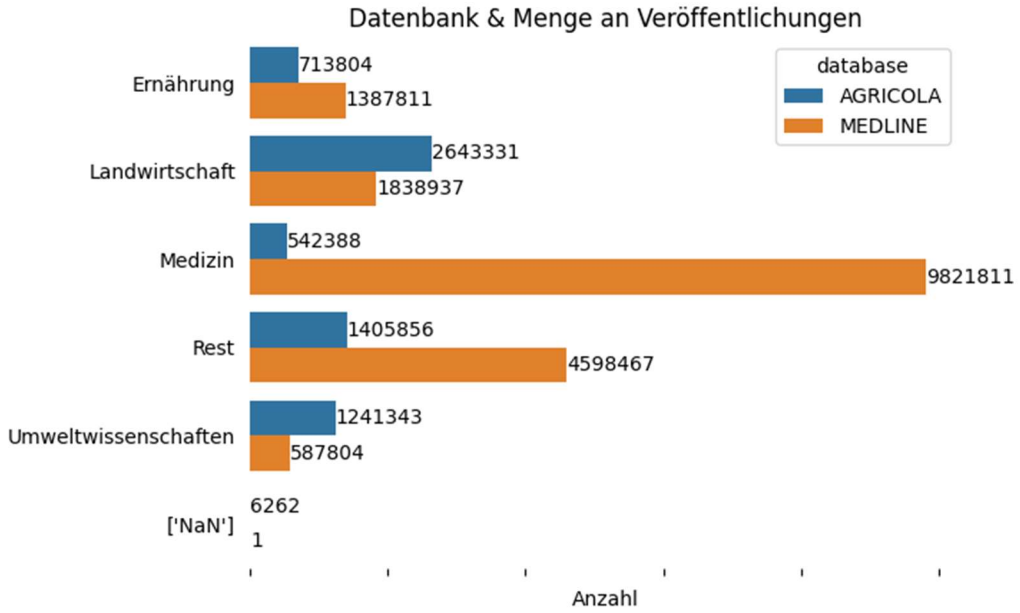

Abbildung 3-3: Klassenaufteilung anhand der Datenbanken AGRICOLA und MEDLINE

Jedoch ist auch hier anzumerken, dass der „Golden Record“ durchaus fehlerhaft sein kann. Eine Überprüfung der Richtigkeit der zugteilten Klassen von Averbis würde sich nur mit einer zielgerichteten explorativen Analyse überprüfen lassen.

Die nächstfeinere Stufe, die eine weitere simple Klassifizierungsmethode darstellen könnte, wäre die Einteilung der Klassen nach Publishern. Zu überprüfen wäre also, ob eindeutige Verknüpfungen zwischen einem bestimmten Verleger und einer von Averbis zugeteilten Klasse existieren.

[2] „Eine signifikante Verzerrung […] weist darauf hin, dass ein Fehler in Ihrem Modell vorliegt, da dies darauf hinweist, dass das Modell falsch hinsichtlich der Häufigkeit positiver Labels ist.“ (Google Developers 2022.)

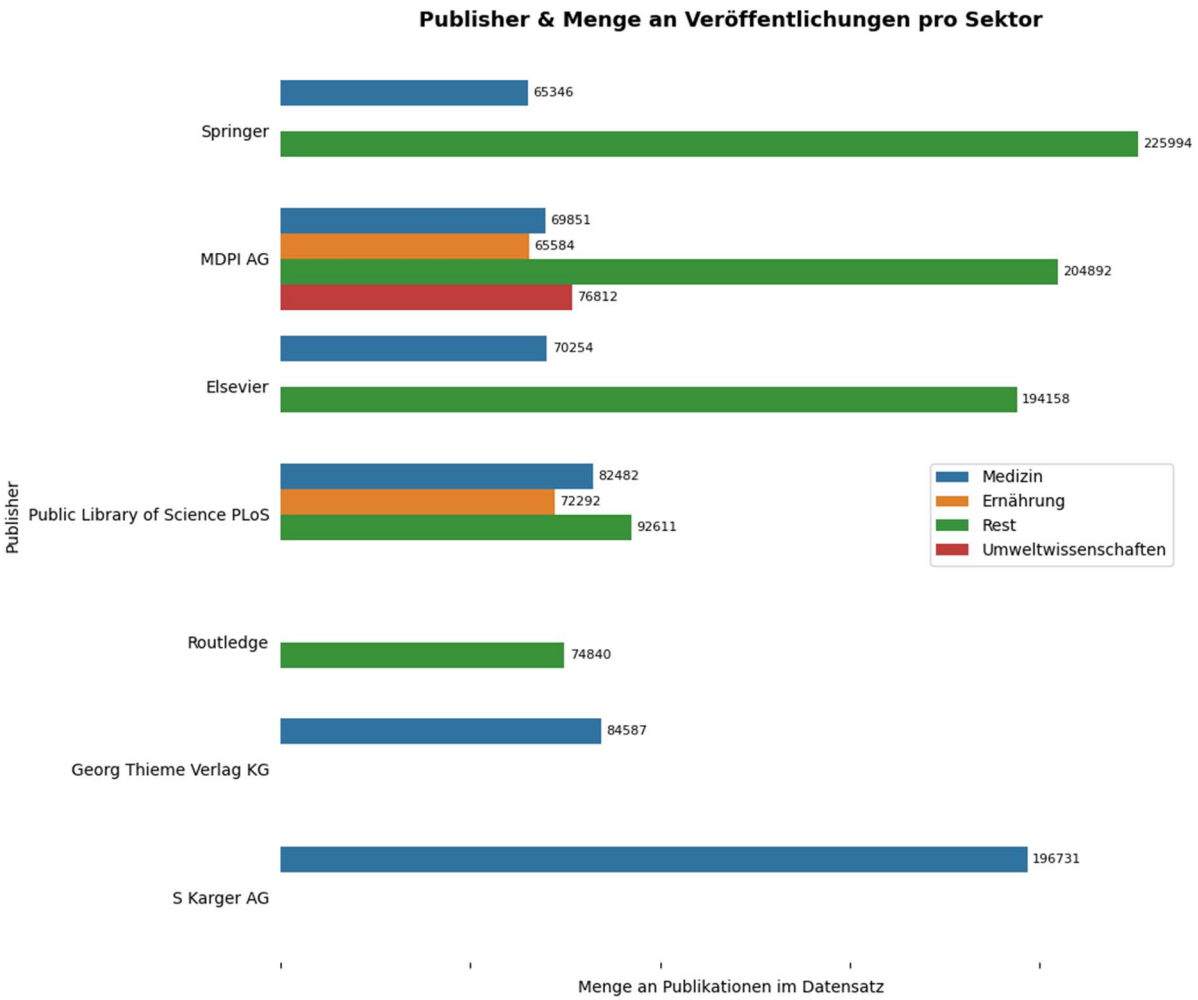

Abbildung 3-4: Publisher & Menge an Veröffentlichungen

Wie der Abbildung 3-4 zu entnehmen ist, gibt es tatsächlich Verleger, die in nur einer Klasse publizieren, diese gelten allerdings als eine Minderheit. Die Abbildung zeigt nur Verleger mit mehr als 60.000 Publikationen. Es gibt kleinere Verleger, die nur in einer Klasse veröffentlichen, bei jenen ist die Anzahl der Publikationen nur sehr beschränkt. Zur Veranschaulichung wurde daher eine Grenze von 60.000 Publikationen gewählt. Die Anzahl der Verleger mit nur einer Fachrichtung, ist mit dieser Filterung bei einem Verhältnis von drei zu sieben. Jedoch ist der Verleger „Routledge“ ausschließlich mit der Kategorie „Rest“ klassifiziert worden und bietet bei einer neuen Klassifizierungsmethode keinen Mehrwert. Die Verleger „Georg Thieme Verlag KG“ und „S Karger AG“ veröffentlichen nur in der Klasse „Medizin“. Mit insgesamt 281.318 Veröffentlichung stellen sie im Verhältnis mit den restlichen Verlegern einen Anteil von 17,85% dar. Bei einer Klassifizierungsmethode die als „Input-Layer“ die Verleger berücksichtigen würde, könnten die Ergebnisse so verbessert werden. Allerdings wurde diese Möglichkeit nicht weiterverfolgt, unter anderem wegen einer komplizierten Umsetzung für den Aufbau eines Korpus mit Metadaten für das LDA-Modell. Zudem könnten diese Metadaten auch zu einem „Bias“ mit unerwünschten Ergebnissen führen, die schwer nachzuvollziehen wären. Eine andere Möglichkeit wäre Verleger, die nur einer Klasse publizieren, im Vorhinein zu filtern und mit einem anderen Programm direkt zu klassifizieren.

Ein weiteres Argument, das gegen die Verwendung von Publisher-Einträgen als Metadaten zu verwenden spricht kann Abbildung 3-5 entnommen werden. Denn insgesamt sind 34.675.199 Dokumente / Datenbankeinträge ohne einen Verlag abgelegt. Dies entspricht einem Gesamtanteil von ca. 62.91% (Abbildung 6-1). Ein Großteil dieser Daten wurden von Averbis der Kategorie „Rest“

eingestuft mit ca. 13.2 Millionen Einträgen, die nächstgrößere Kategorie ist wieder „Medizin“ mit knapp 9.8 Millionen Einträgen. Gefolgt von „Landwirtschaft“ mit ca. 6,1 Millionen Einträgen, Umweltwissenschaften mit ca. 2.9 Millionen und „Ernährung“ mit ca. 2.5 Millionen Publikationen (Abbildung 3-5).

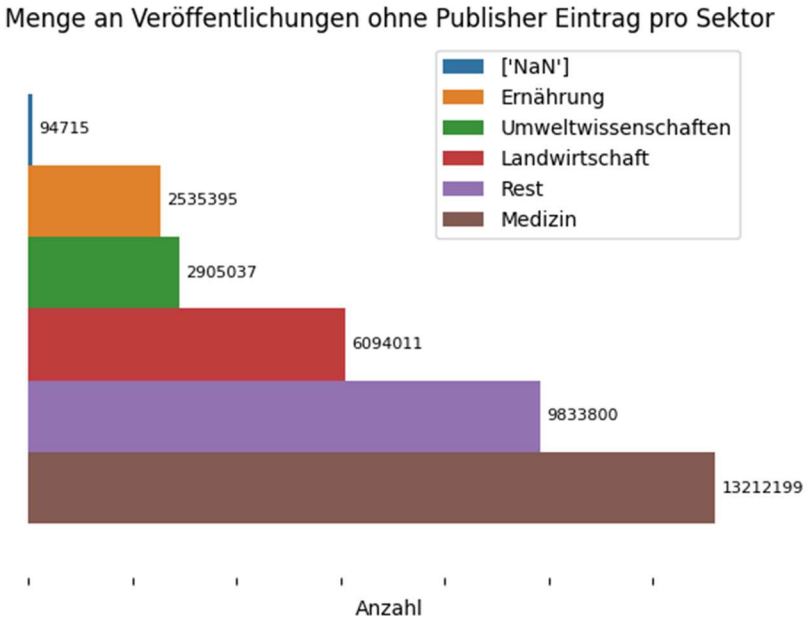

Abbildung 3-5: Menge an Veröffentlichungen ohne Publisher-Eintrag, soriert nach Sektor

Auch wenn ca. 37% der Dokumente in der Datenbank mit einem Eintrag für die Spalte „publisher“ besitzen, ist dieser Publisher in den häufigsten Fällen kein großer Verlag mit vielen tausend Dokumenten eines einzigen Fachgebiets. Abbildung 6-2 zeigt ganz, dass die meisten Publisher 4 oder weniger Dokumente veröffentlicht haben. Der größte Teil der Publisher, mit ca. 34,39% Anteil, ist mit nur einer Veröffentlichung in der Datenbank gespeichert. Publisher mit mehreren tausend Dokumenten machen nur einen Bruchteil der Datenbank aus. Die Wahrscheinlichkeit, aus einem Input-Layer der Publisher eine Verbesserung für Vorhersagemodelle zu schaffen, dürfte daher gering sein. Die Datengrundlage ist hier nicht ausreichend, um über viele Publikationen Regelmäßigkeiten so hervorheben zu können, dass ein Vorhersagemodell zuverlässige Aussagen treffen kann.

3.2 Aufbau des Text Corpus

Ein Textkorpus ist eine Sammlung von Texten, die ihren Ursprung aus mehreren Dokumenten bzw. Quellen haben können. Bei kleineren Anwendungen kann der Textkorpus auch aus nur einem Dokument entstehen. Für den Textkorpus, der für das Trainieren und Anwenden der Klassifizierungsmethoden, der LIVIVO Datenbank verwendet wurden. Wurden mehrere Bestandteile der Datenbank verwendet, um diesen zu Bilden. Die verwendeten Dokumente bzw. Texte sind die Tabellenspalten der Titel, Abstracts und Kollektionstitel. Diese drei Spalten beinhalten die meisten Informationen, die für das Trainieren und Testen der Modelle relevant sind. Andere Informationen wie die Ursprungsdatenbank oder Publisher wurden nicht berücksichtigt, um den Aufbau der Klassifizierungspipelines so simpel wie möglich zu halten. Es gibt an dieser Stelle also noch Optimierungsmöglichkeiten für zukünftige Klassifizierungsversuche.

Die Tabellenspalten „Titel“, „Abstract“ und „Kollektionstitel“ müssen für das Trainieren des LDA-Modells also vorverarbeitet werden, um mit der Python Bibliothek von „gensim“[3] zu funktionieren.

Um die Funktion der Bibliothek „gensim“ zu gewährleisten, müssen die Daten zuerst „gesäubert“ werden. Die einzelnen Texte werden zuerst nach Sonderzeichen durchsucht, die nicht zum Satzbau gehören, sprich alles außer Punkten, Fragezeichen, Ausrufezeichen, Anführungszeichen, etc. Danach wird der Text konvertiert, sodass alle Wörter klein geschrieben sind. Als nächstes können die einzelnen Worte in den Texten lemmatisiert oder einem „Word Stemming“ unterzogen werden. Bei der Lemmatisierung werden Wörter auf ihre Grundform reduziert. Für jedes Wort in einem Satz wird also der Wortstamm erkannt und durch einen entsprechenden Wörterbucheintrag ersetzt. Das Wörterbuch kann aus verschiedenen Quellen zusammengestellt werden. Für einen schnelleren Erfolg können aber auch vorgefertigte Wörterbücher benutzt werden, wie zum Beispiel aus der Python Bibliothek „NLTK“ (Natural Language Toolkit). „Word Stemming“ ist ein ähnlicher Ansatz, nur werden hier die Wörter anders verarbeitet. „Word Stemming“ also Wortstammbildung versucht die Wörter in Texten auf ihr Stammform zu reduzieren. Texte werden vereinfacht, indem Suffixe und Präfixe entfernt werden. Aus „Läufer“ würde nach dem Reduzieren auf den Wortstamm „lauf“ werden. Mit beiden Methoden können Suchen oder Klassifizierungen in großen Texten verbessert werden, da nicht nach allen möglichen Variationen eines Wortes gesucht werden muss. Mithilfe der Lemmatisierung soll die Genauigkeit der Klassifizierungsmodellen verbessert werden.

Eine Problematik, die sich mit dem erhaltenen Datensatz allerdings ergibt, ist die Vielzahl an Sprachen, in denen die Publikationen verfasst wurden. Die häufigsten Sprachen sind Englisch, Deutsch, Spanisch, Französisch, Italienisch, aber auch Texte mit kyrillischen Schriftzeichen und anderen Alphabeten sind in der Datenbank hinterlegt. Durch diese Vermischung der Texte können bei der Textlemmatisierung oder der Wortstammbildung Überschneidungen entstehen, die einen negativen Einfluss auf die Genauigkeit der Klassifizierung haben, vor allen zwischen verwandten Sprachen wie der romanischen. Aus dem deutschen Wort „für“ kann mit einer englischen Lemmatisierung oder Wortstammbildung „fur“ (Fell), werden. Ein eindeutiger Identifier in welcher Sprache die Publikation verfasst wurde, wäre also ein weiterer Schritt für die Verbesserung der Genauigkeit der Modelle. Durch das Fehlen dieser Indikatoren innerhalb des Datenbankauszuges und einem zeitlich erheblichen Aufwand diese Indikatoren selbst zu erstellen, wurden alle Texte mit einer Zusammenstellung aus einem mehrsprachigen Wörterbuch lemmatisiert. (Saumyab271 2022)

Nach dem Schritt der Lemmatisierung oder der Wortstammbildung können Stoppwörter entfernt werden. Stoppwörter sind die häufigsten Wörter in Texten, die keinen „Inhalt“ bieten. Sie lassen einen Text flüssig und „rund“ erscheinen. Allerdings ist ihre Existenz für eine Themenanalyse oder Textklassifizierung vollkommen überflüssig, da sie kein „Wissen“ übermitteln. Stoppwörter sind zum Beispiel: „der“, „die“, „das“ oder „oder“. Wichtige Wörter in einem medizinischen Text könnten zum Beispiel „Covid“, „Neuronen“ oder „Krebs“ sein. Durch das Entfernen von Stoppwörtern wird in einem Text sozusagen das Wissen „verdichtet“ und nur auf die essentiellen Wörter reduziert. Das hilft einerseits die Verarbeitungsgeschwindigkeit zu erhöhen, aber auch die Genauigkeit der Modelle, da diese beim trainieren nicht versuchen, das Wort „und“ mit einem Label oder einer Klasse zu verknüpfen. Moderne Neuronale Netze benötigen viele dieser Schritte im Voraus nicht mehr, brauchen dafür aber mehr Rechenleistung und deutlich größere Trainingsdatensätze als zum Beispiel eine logistische Regression. (Ganesan 2014, 2019)

[3] Gensim ist eine Open-Source Software Bibliothek für Textverarbeitung (Gensim: topic modelling for humans 2022).

Zu einer weiteren Verbesserung des Trainingsdatensatzes bzw. des Textkorpus, kann die Erstellung von Bi/N-Grams führen.

N-Grams sind eine Art in der Textanalyse um Wortfolgen zu untersuchen. Sie sollen dabei helfen, Zusammenhänge zwischen Wörtern deutlicher abzubilden. Es wird bei der Erstellung der N-Grams gezählt wie oft bestimmte Kombinationen von aufeinanderfolgenden Worten existieren. Aus mehreren medizinischen Texten könnten beispielsweise folgende N-Grams entstehen: „DNA Sequenzierung“, „schwerer Covid Verlauf“, „long Covid“.

N-Grams sind hilfreich, den Kontext in Texten besser zu erfassen und darzustellen, um Sprachvorhersagemodelle oder weitere Modelle noch genauer zu machen. Auch das neue Klassifizierungsmodell für LIVIVO könnte von der Erstellung von N-Grams profitieren. Aus zeitlichen Gründen wurde für diese Arbeit allerdings auf die Erstellung von N-Grams verzichtet. (XRDS 2017; Ganesan 2018)

Der finale Schritt zur Korpus erstellung ist die Tokenisierung der Texte, dazu wird der Text in einzelne Worte aufgeteilt. Jedes Wort wird an der ursprünglichen Position innerhalb des Textes gespeichert, jedoch mit einem eindeutigen Indikator von anderen Wörtern getrennt. Getrennt werden die Worte z.B. anhand von Leerzeichen, Kommas, Punkten oder anderen Satzzeichen. Aus den aufgeteilten Texten werden Listen oder Tupel erstellt, die den ursprünglichen Text als Token abspeichern. Ein solcher Tupel könnte wie folgt aussehen: [„Dieser“, „Text“, „ist“, „ein“, „Tupel“] (ohne Entfernung von Stoppwörtern oder einer Lemmatisierung) mit einem zweiten Eintrag pro Schlüssel z.B.: [„Tupel“, {POS: nomen}]. (Menzli 2022)

3.3 Erstellung des Korpus

Die Erstellung des Textkorpus verlief iterativ und wurde nach und nach weiterentwickelt. Für die ersten Iterationen der Klassifizierungsmodelle (LDA und SGDC) wurde der Textkorpus immer wieder neu aufgebaut. Die Vorarbeiten, die im Abschnitt 3.1 beschrieben wurden, mussten für jeden Test in der Entwicklung der Modelle neu durchgeführt werden. Je nach Trainingsdatensatzgröße war der zeitliche Aufwand variabel, je größer der Datensatz, desto länger dauerte der Aufbau des Korpus. Um Zeit zu sparen, wurde nach einigen Iterationen, ein Textkorpus für jedes vorhandene Dokument in der Datenbank erstellt. So hat jedes Dokument einen eigenen tokenisierten Eintrag in der PostgreSQL Datenbank. Mit diesem Schritt konnte viel Zeit gespart werden. Für ein Modelltraining müssen so nur noch die fertig tokenisierten Abschnitte, die von Averbis zugeteilten Klassen und optional die Record ID’s geladen werden.

Der zugrunde liegende Prozess für die Erstellung der Tokens ist im folgenden Abschnitt festgehalten. Die benötigten Daten werden aus der bereitgestellten PostgreSQL Datenbank geladen und enthalten die Tabellenspalten „recordid“, „title“, „collection_title“ und „abstract“. Die Spalte „recordid“ dient lediglich dazu die Daten nach der Verarbeitung wieder mit ihren Originaldokumenten verknüpfen zu können. Die Spalten „title“, „collection_title“ und „abstract“ enthalten die benötigten Daten, um aus den Dokumenten die Textkorpusse zu erstellen.

Ein Problem welches bei der Verarbeitung einer so großen Datenmenge (ca. 64 GB) entsteht, ist der benötigte Arbeitsspeicher und die Verarbeitungszeit. Die benutzte (private) Maschine, eine 24 CPU-Kerne und 188 Gb RAM starke VM, hatte in Verbindung mit Python mehrere Komplikationen, die Pipeline zu parallelisieren. Eine Parallelisierung ist nicht zwingend notwendig, jedoch ist die theoretische Beschleunigung der Verarbeitung zumindest einen Versuch wert. Die verwendete Pipeline konnte allerdings mit 20 der 24 CPU-Kerne teilweise parallelisiert werden und somit, wo

möglich, eine 20-fache Beschleunigung erfahren. Um den Arbeitsspeicher nicht zu überfluten, wurden die Daten in Teilstücken von der Datenbank geladen und nach dem Verarbeiten wieder zu einer großen Datenbank zusammengefügt.

```
import pandas as pd
import sqlalchemy
from sqlalchemy import text
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from pandarallel import pandarallel
from tqdm import tqdm
stop_words = set(stopwords.words('english') + stopwords.words('german') +
stopwords.words('french') + stopwords.words('spanish'))
nltk.download('wordnet')
from sqlalchemy import create_engine, select
engine = create_engine(INSTERT DB-Connection)
pandarallel.initialize(progress_bar=False,nb_workers=20)
tqdm.pandas()
```

Abbildung 3-6: Python-Module für die Vorverarbeitung und Tokenisierung der Dokumente

Die für diesen Abschnitt benötigten Python Bibliotheken (Abb. 3-6) belaufen sich auf Pandas als Werkzeug für das Tabellen-Management innerhalb von Python. Sqlalchemy [4]um eine Verbindung mit PostgreSQL aufbauen zu können.

NLTK[5] für das Entfernen der Stoppwörter, Tokenisierung und Lemmatisierung der Texte. Als Lemmatisierung wurde NLTK mitunter ausgewählt, weil diese Bibliothek eine multilinguale Textverarbeitung anbietet. Für die Stoppwortfilterung werden ebenfalls die NLTK Bibliotheken herunter geladen, für die Sprachen Deutsch, Englisch, Französisch und Spanisch. Tqdm[6] ist ein Modul zur Anzeige eines Fortschrittbalkens. Pandarallel[7] dient dazu, Python-Skripte mit möglichst wenig Aufwand zu parallelisieren. Bei großen Datensätzen wie diesem ist die Zeitersparnis teilweise enorm, da anstatt von nur einem Rechenkern, alle physischen Rechenkerne des Prozessors verwendet werden können.

Nach der Initialisierung der verschiedenen Python Bibliotheken müssen Funktionen deklariert werden damit „Pandarallel“ alle verfügbaren Rechenkerne nutzen kann. Hier ist es wichtig, dass innerhalb der Funktionen wieder die benötigten Bibliotheken initialisiert werden, da diese nicht aus dem Hauptprozess an die von Pandarallel erstellten „Helferprozesse“ übergeben werden können. (Pandarallel documentation 2022)

```
def remove_special_chars(text):
    import re
    return re.sub('(^\{\")|(\"\}$)|(^\{)|(\}$)', '', text)
    return x
```

Abbildung 3-7: Python Funktion als Beispiel für die Prozess-Parallelisierung

[4] https://www.sqlalchemy.org/
[5] https://www.nltk.org/
[6] https://github.com/tqdm/tqdm
[7] https://github.com/nalepae/pandarallel

Die in Abbildung 3-7 dargestellten Funktionen dienen der Säuberung der Texte und entfernen überflüssige Sonderzeichen, die aus der SQL-Abfrage entstanden sind. Es werden die Textzeichenfolgen ({“ ; “} ; { ; }) am Anfang und Ende der Strings gelöscht. Die Funktion bekommt aus jeder Zeile die Spalten als Strings übergeben und gibt die gesäuberten Texte wieder als String in den Prozess zurück.

Die Funktion (Abbildung 3-9) übernimmt Strings, diese werden mit NLTK verarbeitet um die Texte zu tokenisieren, lemmatisieren und jedes Wort in kleine Buchstaben zu konvertieren. In den Hauptprozess wird ein String übergeben, in dem jedes Wort tokenisiert ist.

```
def lemma(x):
    import nltk
    from nltk.tokenize import word_tokenize
    from nltk.stem import WordNetLemmatizer
    w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
    lemmatizer = nltk.stem.WordNetLemmatizer()
    x = x.lower()
    x = [lemmatizer.lemmatize(w) for w in
    w_tokenizer.tokenize(x)]
    return x
```

Abbildung 3-8: Python Funktion für die Lemmatisierung und Tokenisierung der Dokumente

Der Codeausschnitt in Abbildung 3-8 zeigt, wie mit Hilfe der „sqlalchemy“ Funktion „stream_results“ die SQL-Abfrage in 5.5 Millionen Stücke aufgeteilt wurde, um Komptabilitätsprobleme mit Pandarallel zu vermeiden. Jedes Teilstück der Datenbank wird in einem „Dataframe“ gespeichert, durch das eine „For-Schleife“ spaltenweise iteriert. Zuerst werden die Spalten „title“, „collectiontile“ und „abstract“

```
dfs=[]
result = conn.execution_options(stream_results=True).execute(select(
([text("""dbrecordid,collectiontitle,
abstract,
title
FROM  ke_stage.bachelor_mp_raw_ke""")] )))
while chunk:= result.fetchmany(5500000): ## only get x rows at a time
  df = pd.DataFrame (chunk)
  for column in df:
    print(column)
    df[column] = df[column].astype(str)
    df[column] = df[column].parallel_apply(remove_special_chars)
    if column in collist:
        print("lemma")
        df[column] = df[column].parallel_apply(lemma)
        print("gensim")
        df[column] = df[column].parallel_apply(gensim_pre)
        df[column] = df[column].progress_apply(lambda x: ','.join([word for word in
x if word not in (stop_words)]))
  dfs.append(df)
df_res = pd.concat(dfs, ignore_index=True)
```

Abbildung 3-9: Python Code mit Funktionen zum Datenbankstreaming und vollständiger Vorverarbeitung der Dokumente

in ein Stringformat umgewandelt, damit die späteren Funktionen ihre Textverarbeitung erfüllen können. Würde dieser Schritt wegfallen, würde der Code an mehreren Stellen abbrechen, da im Dataframe nicht alle Reihen „strings“ enthalten würden. Die erste Funktion „remove_special_chars“ kann allerdings nur „strings“ verarbeiten und stürzt bei jedem anderen Datenformat ab. Nachdem mit Hilfe der Funktion „remove_special_chars“ die überflüssigen Sonderzeichen aus der SQL-Abfrage gelöscht wurden, wird jedes Textfeld in den Spalten „title“, „collection_title“ und „abstract“ tokenisiert, lemmatisiert und in kleine Buchstaben konvertiert. Im Anschluss werden noch alle anderen Satzzeichen bzw. Sonderzeichen gänzlich entfernt, und mit der Funktion dann in eine Liste für die tokenisierten Wörter geschrieben zu werden. Diese Liste wird wieder in das aktuelle Dataframe übergeben. Nach jedem Durchlauf der Teilstücke wird das aktuelle Dataframe an ein „gesamtes“ Dataframe angefügt, in dem die gesamte Datenbank repräsentiert ist.

Die Laufzeit für den gesamten Verarbeitungsprozess beträgt mit einer ähnlichen VM etwa einen Tag bei ca. 55 Millionen Datenbankeinträgen. Das erstellte Dataframe mit allen Einträgen wird nach dem Durchlauf in eine neue Tabelle der Datenbank („db_tokenized_all“) auf dem konfigurierten PostgreSQL Server hochgeladen (Abbildung 3-10).

```
engine =
create_engine('postgresql+psycopg2://postgres:****@192.168.0.137:5432/ba1')

df_res.to_sql("db_tokenized_all", engine,schema="ke_stage")
```

Abbildung 3-10: Python Beispiel für das Hochladen des fertiggestellten Textkorpus, für jedes Dokument

In der Tabelle enthalten sind die Spalten „dbrecordid“, zur Verknüpfung mit den Originaldaten; „title_token“, „collectiontitle_token“ und „abstract_token“ enthalten die verarbeiteten und tokenisierten Texte der entsprechenden Einträge.

3.4 Einteilung in Trainings und Validations Daten

Nach der gesamten Vorverarbeitung der Daten können nun Auszüge aus der Datenbank geladen und direkt für Modelltrainings benutzt werden. Dies geschieht wieder über eine SQL-Abfrage mittels slqalchemy. Um das zu trainierende Modell mit gleichgroßen Datensätzen pro Thema versorgen zu können, muss die SQL-Abfrage die Einträge mit entsprechendem Thema zählen. Um die SQL-Befehle einfach zu halten, werden für alle vier Themen einzelne SQL-Abfragen gestellt, die mit einem „LIMIT“-Befehl die maximale Anzahl der geladenen Einträge beschränken (Abbildung 3-11).

```
lim=100000 #Datensatzgrößenlimit
size = lim
#Lade von jedem Thema die maximal zulässige Anzahl (lim)
df_med = sql_read("'Medizin'",lim)
df_land = sql_read("'Landwirtschaft'",lim)
df_umwelt = sql_read("'Umweltwissenschaften'",lim)
df_ern = sql_read("'ErnÃ¤hrung'",lim)
```

Abbildung 3-11: Python Code zur Textkorpusabfrage der Dokumente nach Klassen

Nachdem Daten für alle vier Kategorien in getrennten Dataframes heruntergeladen sind, werden die einzelnen Dataframes zu einem zusammengefügt. Im Dataframe mit allen Kategorien werden leere Einträge mit „NaN“ markiert, um diese nach Vollständigkeit überprüfen zu können. Wenn eine Spalte der drei tokenisierten Spalten Text beinhaltet, wird der Dataframeeintrag behalten.

Im Falle eines Eintrags ohne Inhalte in den Spalten „title_token“, „abstract_token“ und „collectiontitle_token”, wird dieser aus dem Dataframe gelöscht. Anschließend werden die mit „Nan” gefüllten Zellen wieder in leere Strings konvertiert, um beim Zusammenfügen der Spalten keine falschen Einträge zu übernehmen (Abbildung 3-12).

```
df = pd.concat([df_med, df_land,df_umwelt,df_ern])
df = df.replace('', np.nan)
df = df.drop(df[pd.isna(df['collectiontitle_token'])
& pd.isna(df['abstract_token'])
& pd.isna(df['title_token'])].index)
df = df.replace(np.nan,'')
```

Abbildung 3-12: Python Code zur Zusammenlegung der einzelnen Datenbankspalten, sowie die Filterung gänzlich leerer Einträge

Durch das Löschen der leeren Einträge wird verhindert, dass das Modell beim Trainieren ein leeres Feld für einen Indikator einer bestimmten Klasse hält. Wenn z.B. bei 50% der Klasse „Landwirtschaft“ leere Felder als Token gespeichert sind, besteht eine hohe Chance, dass das Modell zukünftig leere Felder als Kategorie „Landwirtschaft“ interpretiert. Zudem kann bei leeren Einträgen nicht überprüft werden, wie gut das Modell die Token bzw. den Textinhalt interpretiert. Schließlich kann ein Eintrag ohne Daten nicht korrekt kategorisiert werden, schlicht und einfach, weil es keine Anhaltspunkte gibt.

Die nächste wichtige Aufgabe bei der Vorbereitung für die Trainings und Testdaten ist die gleichmäßige Verteilung der Klassen im Datensatz. Ohne gleichmäßige Verteilung könnten Klassen teilweise oder sogar gar nicht im Testdatensatz vorkommen. Ein einfaches „kürzen“ des Dataframes ist also nicht ausreichend, um die Daten gleichmäßig aufzuteilen.

Stattdessen wird gezählt, wie viele Einträge jede Klasse im Dataframe hat. Die Klasse mit den geringsten Einträgen gibt danach das Höchstmaß der verfügbaren Daten an. Durch die Zählung und Beschränkung der Einträge im Datensatz, kann verhindert werden, dass das Modell mit einem „Bias“ trainiert wird. Auch kann so sichergestellt werden, dass tatsächlich jede Kategorie berücksichtigt wird und nicht nur die, die in den ersten n-Zeilen des Trainingsdatensatzes stehen.

In Abbildung 3-13 ist der Code der Datenaufteilung und Zählung zu sehen. Zuerst werden die Klassen auf die maximal angegebene Gesamtgröße des gewünschten Datensatzes begrenzt und aufgeteilt (Abbildung 3-13 & 3-14/6-10).

```
df_med = df.loc[df['class'] == "Medizin"].head(int(size))
```

Abbildung 3-13: Pythoncode zur Aufteilung des Trainingdatensatzes nach Klasse

Danach wird die tatsächliche Größe der Klassen überprüft, denn nicht immer sind zwischen allen Klassen die leeren Einträge gleichmäßig aufgeteilt.

```
counted, lowest_c = count_class_pop(df)
def count_class_pop(x):
      Siehe Abbildung 6-10
```

Abbildung 3-14: Python Code zur Zählung der verfügbaren Dokumentenanzahl im Datensatz

Nach der Zählung der Einträge pro Klasse werden die klassenspezifischen Dataframes auf die Größe des kleinsten klassenspezifischen Dataframe reduziert. Anschließend werden sie wieder in ein gesamtheitliches Dataframe gespeichert und auf Richtigkeit überprüft. Nachdem die Klassen alle gleich

viele Einträge im Datensatz haben, wird mit Hilfe einer „SciKit-Learn“ das Dataframe gleichmäßig über alle Klassen in ein Training und Testdatensatz aufgeteilt (Abbildung 6-10 & vereinfacht Abbildung 3-15).

Die Einträge in diesen Datensätzen sollen durch das aufwändige Vorbereiten und Aufteilen in gleich große Klassen einen möglichst geringen „Bias“ im Modelltraining hervorrufen.

```
df_med = df.loc[df['class'] == "Medizin"].head(int(lowest_c))
df_land = df.loc[df['class'] =='Landwirtschaft'].head(int(lowest_c))
df_umwelt = df.loc[df['class'] =='Umweltwissenschaften'].head(int(lowest_c))
df_ern = df.loc[df['class'] =='ErnÃ¤hrung'].head(int(lowest_c))
df = pd.concat([df_med, df_land,df_umwelt,df_ern])
counted, lowest_c = count_class_pop(df)
df_train, df_test = train_test_split(df, test_size=0.25)
```

Abbildung 3-15: Veranschaulichter Python Code indem die Klassenmengen ausgeglichen werden und der Datensatz in Test und Training aufgeteilt wird

3.5 Klassifizierungsmodelle

3.5.1 Latent Dirichlet Allocation (LDA)

Wie schon im Abschnitt 2.3.2 diskutiert, ist LDA eine Klassifizierungsmethode, welche ohne „Aufsicht“ arbeitet. Es wird versucht, anhand der vorhandenen Texte Verbindungen zu finden, die ein Kluster bzw. ein Thema bilden. Die Methode kann aus ihrer Funktionsweise heraus nicht mit den Klassen von Averbis bzw. des „Golden Records“ trainiert werden. Die Themen werden also automatisch erstellt und zugeteilt.

```
num_topics = 4
max_df_size = 1000000
min_df_size = 1000
modi = 4
while min_df_size < max_df_size:
    while num_topics < 25:
        lda = gensim.models.ldamulticore.LdaMulticore(
                                corpus=train_corpus,
                                num_topics=num_topics,
                                id2word=train_id2w,
                                chunksize=1000,
                                workers=10, # Num. Processing Cores - 1
                                passes=30,
                                eval_every = 6,
                                per_word_topics=False)
            num_topics = num_topics + 4
```

Abbildung 3-16: Veranschaulichter Python Code indem die Schleife des LDA Trainings beschrieben wird

Um herauszufinden, inwieweit die Themenfindung eines LDA-Modells mit der Klassifizierung des Golden Records übereinstimmt, wurde mit Hilfe einer Python-Pipeline eine Schleife entwickelt, die

durch verschiedene Datensatzgrößen und maximaler Themenanzahl iteriert. Zugrunde liegt das LDA-Modell von gensim[8].

Die minimale Anzahl an Themen wurde gleich der des Golden Records gewählt: vier Themen. Als maximale Anzahl an möglichen Themen wurden 24 gewählt. Die maximale Anzahl an Themen wurde aus mehreren Gründen relativ hoch angesetzt. Da das LDA-Modell die Themen selbst zuteilt, kann es sein, dass das Modell tatsächlich Themengebiete granularer eingrenzen kann. Darin können auch identische die Themen, aber in verschiedenen Sprachen auftauchen. Die vorhandene Vielfalt an Sprachen und fehlenden Möglichkeit, nach diesen zu filtern, muss also auch berücksichtigt werden. Mittels der Iterationen durch die maximale Anzahl der Themen soll also versucht werden, ein möglichst gutes Abbild der vorhandenen Möglichkeiten zu schaffen.

Ein weiterer Ansatz, um die Themenfindung vom LDA-Classifier zu überprüfen, ist die Integrierung durch verschieden große Textkorpi. Hier soll geprüft werden, ob sich die Klassifizierungsgenauigkeit im Verhältnis zum Golden Record verändert und ob schon ein kleines Modell reicht, um die Veröffentlichungen in der Datenbank korrekt einzustufen.

Das kleinste Modell wird wie in Abbildung 3-16 mit 1000 Einträgen pro Klasse trainiert und das größte mit 1.000.000 Einträgen pro Klasse. Mit jedem Durchlauf durch die Schleife wird der Datensatz vervierfacht. Der Größenmodifikator vier erlaubt der Schleife also 16 Durchläufe mit jeweils 6 verschiedenen Einstellungen für die maximale Themenanzahl (4, 8, 12, 16, 20, 24). Insgesamt werden also 96 (16 x 6) Modelle trainiert und auf ihre Genauigkeit überprüft.

3.5.2 TF-IDF & Stochastic Gradient Descent Classifier (SGDC)

Der statistische Ansatz von SciKit-Learn Modell „SGDC“[9] ist im Gegensatz zum LDA-Modell eines, welches mit vordefinierten Klassen arbeitet. Wie zuvor im Abschnitt 2.3.3 und 2.3.4 beschrieben arbeitet diese Pipeline mit den Klassen des Golden Records. So werden die tokenisierten Texte

```
lim=100000

size = lim
while size > 100:
   text_clf = Pipeline([
      ('vect',
      CountVectorizer(lowercase=False,stop_words=None,tokenizer=None,min_df=3)),
      ('tfidf', TfidfTransformer(use_idf=True,norm="l2")),
      ('clf', SGDClassifier(loss='hinge', penalty='l2',
                            alpha=1e-3, random_state=42,
                            max_iter=5, tol=None)),
      ])
   text_clf.fit(df_train['combined'], df_train['class'])
   predicted = text_clf.predict(df_test['combined'])
```

eingelesen und zuerst in Vektoren umgewandelt (CountVectorizer), danach werden die Texte mit Hilfe

Abbildung 3-17: Veranschaulichter Python Code in dem die Schleife für das Training und die Validierung des SGDC beschrieben wird

[8] https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html#sphx-glr-auto-examples-tutorials-run-lda-py

[9] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

von TF-IDF auf die wichtigsten Wörter untersucht. Das Ergebnis des TF-IDF Modells wird anschließend von einem „Stochastic Gradient Descent Classifier“ (auf Basis einer Support Vector Machine) verwendet, um statistische Zusammenhänge zwischen den Wörtern bzw. Vektoren und den vordefinierten Klassen zu finden. Ähnlich wie bei dem Training des LDA-Modells wird hier durch die Datensatzgröße iteriert. Die Trainingspipeline lädt zuerst den gesamten Datensatz herunter, die Größe des gesamten Datensatzes für das Training wird im Voraus bestimmt. In dem gekürzten und veranschaulichten Code in Abbildung 3-17 (und Abbildung 6-9) wird die Größe mit der Variable „lim“ (Limit) gesetzt.

Auch hier gilt das Limit für eine Klasse der Ursprungsdatenbank. Bei einem Limit von 100.000 ist die gesamte maximale Größe des Datensatzes auf 400.000 Einträge beschränkt. Nach dem Einteilen der Daten in Trainings und Testdaten (wie beschrieben in Abschnitt 3.3) wird die SciKit-learn Pipeline mit Hilfe der Trainingsdaten trainiert und anhand der Testdaten auf die Klassifizierungsgenauigkeit im Vergleich zum Golden Record bewertet.

4 Ergebnis-Diskussion & Wahl des optimalen Models

4.1 Ergebnisse

In diesem Teil wird beschrieben und aufgezeigt, wie die jeweiligen Klassifizierungsmodelle im Vergleich zum Golden Record der Averbis Klassifizierung abgeschnitten haben.

Wie schon zuvor erläutert, wird die Klassifizierungsmethode von Averbis als Grundlage genommen, um die Genauigkeit der neu erstellten Modelle zu bewerten. Die neuen Modelle wurden mit vier der fünf Klassen trainiert, die fünfte Klasse „Rest“ wurde nicht verwendet, damit Überschneidungen mit den anderen fünf Klassen vermieden werden können. Wie in Abbildung 3-1 aufgezeigt, ist der größte Teil der Datenbank als „Rest“ klassifiziert. Somit beinhaltet „Rest“ wohlmöglich auch einen großen Wortschatz, der zu einem ungewollten Bias innerhalb der neu Trainierten Modelle führen kann. Zudem gibt diese Entscheidung die Freiheit die Averbis-Klasse „Rest“ neu zu klassifizieren. An dieser Stelle wäre allerdings, im Anschluss, eine weitere explorative Analyse nötig, um bewerten zu können, wie gut diese Themen eingeteilt wurden. Hier könnte ein weiteres LDA Modell Abhilfe schaffen, um die repräsentierten Themen übersichtlich darstellen zu können. Anhand dieser Ergebnisse könnte schließlich entschieden werden, ob es Sinn ergibt die Klasse „Rest“ mit einem neuen Modell zu klassifizieren oder ob es feiner aufgeteilte Themen braucht, um diesen Teil der Datenbank sinnvoll abbilden zu können. Wegen dieses komplexen und zeitaufwendigen Problems wurde diese Klasse nicht in Betracht gezogen und bei dem Modelltraining, sowie Auswertung, außer Acht gelassen.

4.1.1 Ergebnisse des LDA-Modells

Wie in Abschnitt 2.3.2 diskutiert wurde, ist LDA eine Klassifizierungsmethode, welche „unsupervised“ ist. Das bedeutet, dass das Modell ohne die Referenzwerte der Averbis-Klassifizierung arbeitet. Es werden numerische, unbeschriftete Themen erstellt. Um den verschiedenen Sprachen und Themenmöglichkeiten Fläche zu bieten, wurde das Modell in getrennten Durchläufen mit einer unterschiedlichen Maximalanzahl an Themen trainiert. Andere mögliche Parameter für das Modelltraining wurden nicht von den Standarteinstellungen abgewandelt. Die maximale Anzahl Themen war: 4 ,8 ,12 ,16, 20 und 24. Ein wünschenswertes Ergebnis wäre bei maximal vier Topics eine klare Überschneidung zu den Averbis-Klassen (Themen). Wenn es mehr Topics als Themen geben, sollte sich eine Gruppierung von Topics bei entsprechenden Themen zeigen. Bei maximal acht Topics könnte eine Aufteilung wie folgt aussehen:

Tabelle 2 Veranschaulichung der möglichen Klassenaufteilung bei maximal acht Topics

Averbis-Klasse (Thema/Themen)	**Topics (nummeriert)**
Ernährung	1, 3
Medizin	2, 6, 8
Landwirtschaft	4, 5
Umweltwissenschaften	7,

Bei maximal vier möglichen Topics wäre eine wünschenswerte Aufteilung:

Tabelle 3 Veranschaulichung der möglichen Klassenaufteilung bei maximal vier Topics

Averbis-Klasse (Thema/Themen)	**Topics (nummeriert)**
Ernährung	4
Medizin	2
Landwirtschaft	1
Umweltwissenschaften	3

Die Abbildung 4 1 zeigt die Aufteilung der Topics innerhalb des Testdatensatzes. Der obere Graph beschreibt die Aufteilung der Averbis-Klassen innerhalb der Testdaten. Der untere Graph stellt die Zuteilung und Anzahl (in Prozent) der Topics im Testdatensatz dar. Auffällig ist die ungleiche Verteilung der Averbis-Klassen bei 100 Einträgen im Datensatz. Der Grund dafür ist die geringe Anzahl an geladenen Testdaten. Da leere Daten aus dem Datensatz entfernt wurden ist die Anzahl an Themen zu gering gewesen, um eine gleichmäßige Aufteilung zu gewährleisten. Eine komplexere Datenhandhabung könnte in dem Fall aushelfen, da sich dieses

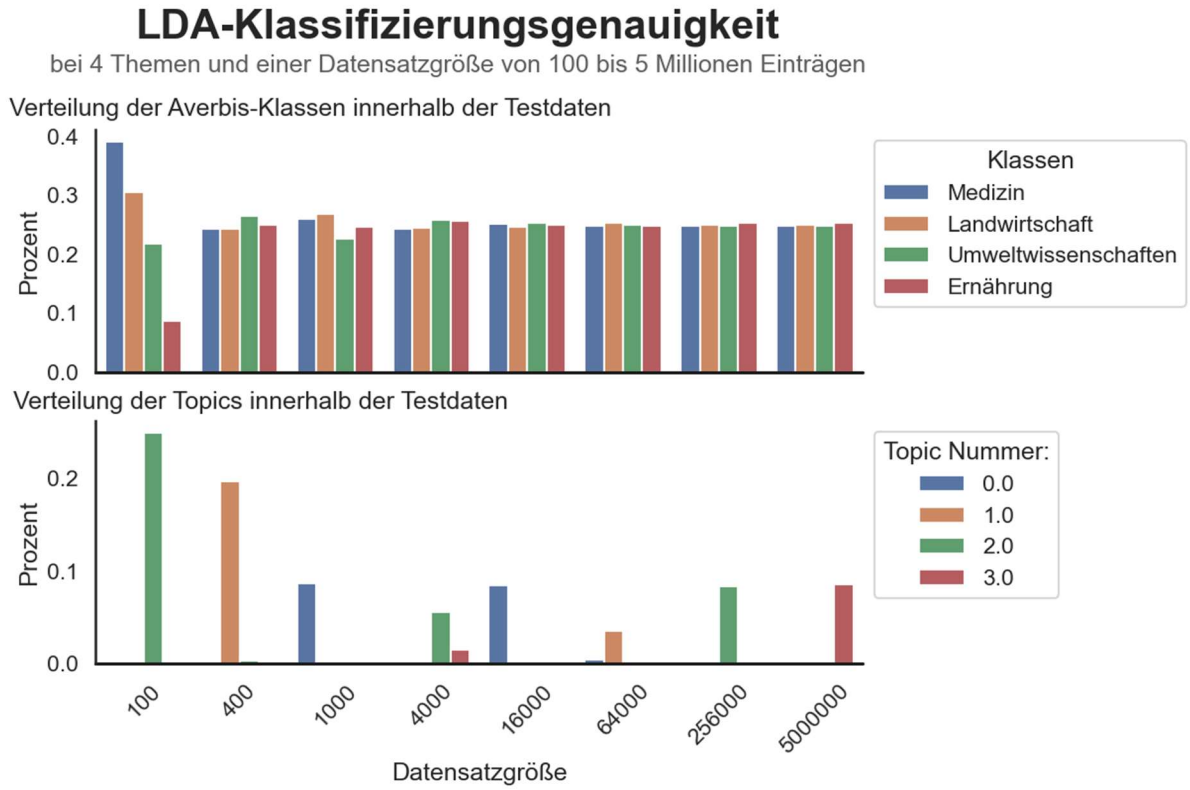

Abbildung 4-1 Vergleich der Averbis Themenanzahl und LDA Topic Zuweisungen bei maximal 4 möglichen Topics

Problem allerdings nur auf einen so kleinen Datensatz beschränkt, wurde keine Verbesserung implementiert. Die Verteilung der Topics ab einer Datensatzgröße von 400 Einträgen zeigt allerdings einige Probleme auf.

Das LDA-Modell klassifiziert, durch alle Datensatzgrößen, ohne Übereinstimmung mit dem zuvor festgelegten Golden Record. Die Verteilung der Topics repräsentiert keine gleichmäßige Aufteilung zwischen den verfügbaren Themen, lediglich bei den Datensatzgrößen von 400, 4000 und 64000 klassifiziert das LDA-Modell mit zwei Topics. Auch werden nicht alle Dokumente im Testdatensatz klassifiziert. Der Grund dafür ist die Möglichkeit des LDA-Modells, eine „certainty“ für das jeweilige Dokument auszugeben. Ab einem bestimmten Schwellenwert (50%) wird dem Dokument kein Topic zugeteilt. Die Anzahl der Klassifizierten Dokumente beläuft sich auf einen Bereich von 4 bis 20% (den Datensatz mit 100 Dokumenten ausgeschlossen). Die Ausbeute an Klassifizierungen lässt somit zu wünschen übrig. Die Verteilung der Themen lässt weiterhin darauf schließen, dass das Modell bei fast allen Testgrößen einen starken Bias für ein Topic entwickelt. Somit werden die Klassifizierten Dokumente nur einem Topic zugeteilt.

Auch das Modell mit maximal acht möglichen Themen (Abbildung 6-5) schneidet schlecht ab. Ähnlich wie bei dem zuvor diskutierten Modell werden nur kleine Teile des Testdatensatzes überhaupt einem

Topic zugeteilt. Die erwartete Gruppierung der Topics zu bestimmten Averbis-Klassen kann nicht beobachtet werden. Der Versuch, das Modell an die Komplexität des Datensatzes mit einem Vielfachen der verfügbaren Klassen anzupassen ist nicht geglückt. Die weitere Vervielfachung der maximal verfügbaren Topics führt weiterhin zu keinem anderen Ergebnis (Abbildung 6-3 & Abbildung 6-4). Im Gegensatz zu den kleineren Modellen wird die Anzahl der Klassifizierten Dokumente, im Vergleich zum gesamten Testdatensatz, geringer. Durch den definierten Schwellenwert für die Topic-Sicherheit von 50% der trainierten Modelle lässt auf eine steigende „Unsicherheit“ schließen, die Modellgenauigkeit nimmt also tatsächlich ab. Das schlechte Ergebnis des LDA-Modells lässt sich weiterhin auch mit einer Stichprobe für die Errechnung des F1-Scores unterstützen.

Das Modell ist klassifiziert bei einer Testdatensatzgröße von 51,200 Dokumenten nur 8.792 Dokumente. Die verwendeten Topics sind in diesem Fall „1“ und „3“. Diese werden allerdings jeder Averbis-Klasse zugeteilt. Die Confusion-Matrix des Modells ist in Abbildung 4 4 dargestellt. Der Grund dafür lässt sich gut mit einer Wörterwolke visualisieren (Abbildung 4 2). In dieser sind die Terme am größten dargestellt, die am schwersten gewichtet sind. Es scheint auch noch ein Fehler im Textkorpus zu sein. Der Begriff „wa“ scheint ein fehlerhaftes Überbleibsel der Textvorverarbeitung zu sein.

Die anderen Begrifflichkeiten lassen sich jedoch auf ein medizinisches Umfeld eingrenzen, mit wenigen Ausnahmen. Problematisch zur korrekten Klassifizierung sind auch solche Begriffe, die einem sehr generellen Umfeld zugeteilt werden können, wie „study“, „level“, „group“, „data“, „functional“ oder „age“. Durch diese schlechte Klassenerkennung ist der F1-Wert dementsprechend niedrig – mit 0.0456.

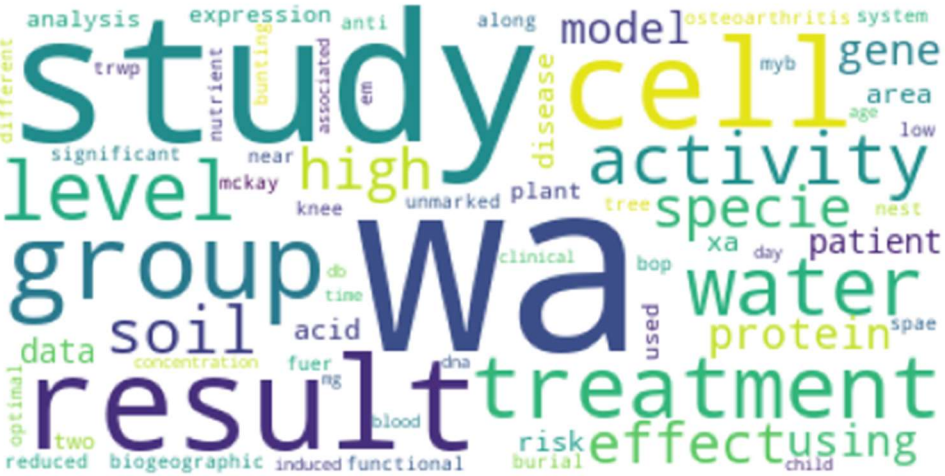

Abbildung 4-2: Wörterwolke-LDA-Modell

4.1.2 Ergebnisse des SGDC

In Abschnitt 2.3.2 und 2.3.3 wurde das Zusammenspiel und Funktionsweise des SGDC und der TF-IDF Funktion erläutert. Bei diesem Modell ist die Vergleichbarkeit zum Golden Record einfacher. Denn das Modell ist ein „supervised“ Modell, grob umrissen lernt es also bestimmte Muster oder Eigenschaften einem vorbestimmten Label (Averbis-Klasse) zuzuweisen. Die Genauigkeit im Vergleich zu Averbis kann mit einem Scoring-System bewertet werden. Wie in Abschnitt 2.3.1 beschrieben, werden als Hilfsmittel die „F1-Score“, „Precision“ und der „Recall“ genutzt, um vergleichen zu können, wie genau das SGDC-Modell die Dokumente bewertet. Der F1-Score ist das kombinierte Maß aus Precision und Recall. Die Bewertung gibt somit eine zuverlässige Aussage über die Richtigkeit der Klassifizierung. In Abbildung 4-9 wird der „F1-Score“ des SGDC-Modells bei verschieden großen Testdatensätzen beschrieben. Die einzelnen Klassen, für die das Modell trainiert wurde, werden getrennt bewertet, um eine granularere Darstellung der Modellleistung zu gewährleisten. Bemerkenswert ist hier die schon hohe Genauigkeit bei kleinen Datensätzen. Mithilfe der geglätteten Leistungskurve lässt sich ein eindeutiger Trend erkennen.

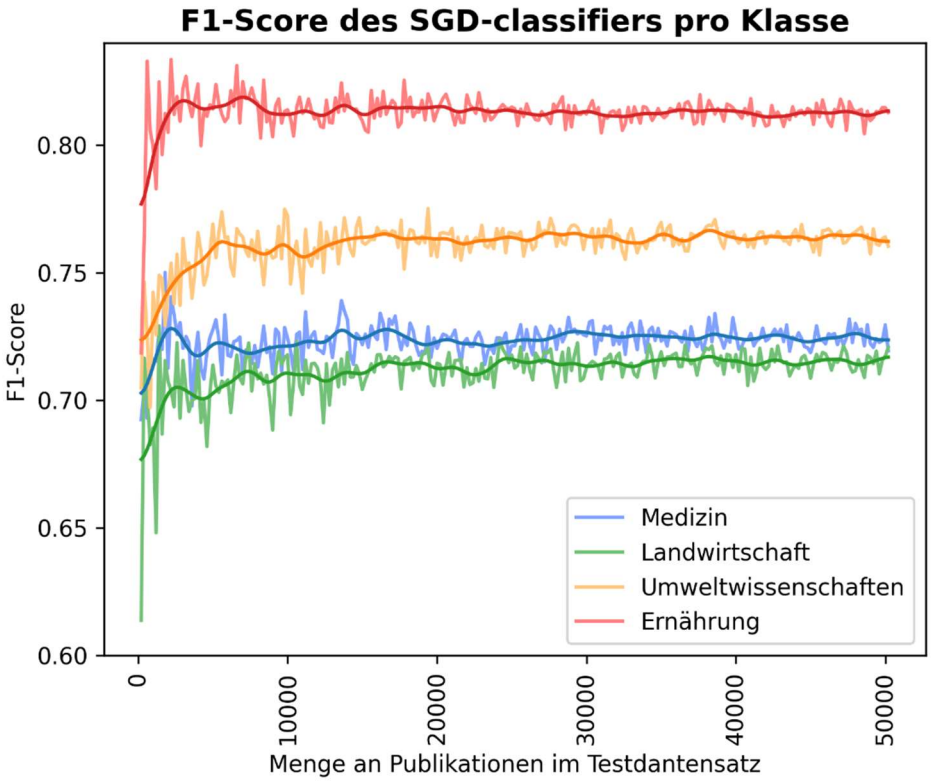

Abbildung 4-9: F1-Score des SGDC-Modells, nach Klassen sortiert

Das Modell kann zwar bei kleinen Trainings- und Testdatensätzen schon gut klassifizieren, doch erst bei ca. 25.000 Daten erreicht das Modell ein stabiles Leistungsniveau über alle Klassen. Auch werden die Ausreißer um die geglättete Kurve kleiner, was auf zuverlässigere Ergebnisse schließen lässt. Die Klassifizierung wird also zuverlässiger.

Der Recall (True Positive Rate) ist ein Maß, um zu bestimmen, wie viele tatsächlich positive Fälle das Modell korrekt als positiv identifiziert hat. Bei Betrachtung des Recalls, des SGDC-Modells (Abbildung 6-8), ist zu sehen, dass die richtig positiven Fälle einer ähnlichen Kurve folgen, wie die des F1-Scores. Eine logische Verknüpfung, da das Maß des F1-Scores mithilfe des Recalls errechnet wird. Aus der Grafik lässt sich aber auch ableiten, dass das Modell bei den Klassen Ernährung und Umweltwissenschaften, die Positiv-Fälle häufig richtig erkennt und klassifiziert. Medizin und

Landwirtschaft schneiden jedoch schlechter ab, mit Werten unterhalb von 70%. Diese Verteilung kann darauf hinweisen, dass die Klassen Ernährung und Umweltwissenschaften besondere Merkmale aufweisen und diese für das Modell einfacher zu erkennen sind. Durch die leicht schlechteren F1-Scores der jeweiligen Klassen kann darauf geschlossen werden, dass das Modell aber auch häufiger falsch-positive Werte zuordnet. Auch die Verteilung der „Precision“ des Modells (Abbildung 6-7) weist darauf hin, dass die Klassen Umweltwissenschaften und Ernährung häufiger falsch-positive Werte zugeteilt bekommen. Bei den Precision-Werten der Klasse „Medizin“ kann auch erkannt werden, dass diese Klasse am häufigsten mit den korrekten Werten eingestuft wird. Daraus kann geschlussfolgert werden, dass das Modell einen gewissen Bias gegenüber den Klassen Umweltwissenschaften und Ernährung gelernt hat und diese häufiger den Dokumenten zuordnet als Medizin oder Landwirtschaft. Es herrscht also noch eine ungleiche Verteilung zwischen den Klassen. Um dieser entgegen wirken zu können, müssten genauere Analysen und Veränderungen am Datensatz vorgenommen werden. Generell ist die Leistung des Modells mit seiner simplen Natur und schnellen Berechnung bemerkenswert.

4.2 Probleme der einzelnen Modelle / Methoden

Bei beiden Modellen sind klare Defizite zu erkennen. Die Ergebnisse des LDA-Modells zeigt vor allem wie schwierig ist, mit dieser Klassifizierungsmethode, gut Ergebnisse zu erzielen. Alleine die geringe Menge an klassifizierten Dokumenten zeigt, wie „unsicher“ das Modell bei der Einstufung der Dokumente ist. Bei einer Mindes-„Sicherheit“ (certainty) von 50% sind bei allen Datensatzgrößen noch nicht einmal 30% der Daten überhaupt einem Topic zugewiesen. Der Wert könnte natürlich weiter heruntergesetzt werden, soweit dass alle Dokumente ein Topic zugewiesen bekommen. Jedoch würde das die Qualität der Klassifizierung nicht verbessern. Ein Münzwurf dürfte dann ähnlich präzise Ergebnisse liefern. Um das Modell zu verbessern, könnte an anderen Stellen gearbeitet werden. Zu einem können die Lern-Parameter des Modells mit Hilfe einer „Gridsearch“ an optimalere Einstellungen angenähert werden, je nach Parameter Menge und Modifikatoren ist mit beträchtlichen Berechnungszeiten zu rechnen. (scikit-learn 2023c) Als Beispiel für die Dauer eines Lerndurchlaufes kann die verwendete Pipeline dienen. Ein Durchlauf, bei einer Datensatzgröße von insgesamt 4 Millionen Dokumenten, dauerte für das LDA-Modell etwa 1200-1300 Minuten. Mit einer Gridsearch von 4 Parametern und 4 möglichen Modifikatoren würde bei 4 Millionen Dokumenten ein Durchlauf etwa 320 Stunden dauern (4 x 4 x 1200[min]). Eine deutliche Ergebnisverbesserung ist hier auch fragwürdig. Bei kleineren Datensätzen ist die Berechnungsdauer für einen Durchlauf deutlich kürzer, aber trotzdem nicht schnell, im direkten Vergleich zu anderen, besseren Modellen.

Um das Modell und die Ergebnisse weiter zu verbessern sollte, wenn überhaupt, an einer Erweiterung des Textkorpus gearbeitet werden. Hier könnte dem LDA-Modell geholfen werden, indem mehr prägnante Wörter hervorgehoben werden. Hierfür würde sich ggfs. auch eine TF-IDF Filterung oder eine Schlüsselwort-Extraktion anbieten. Themenrelevante Wörter könnten so besser gewichtet werden. Eine Unterteilung des Datensatzes in die verschiedenen Sprachen könnte auch helfen, so könnte für jede Sprache ein Modell trainiert werden, welches nicht noch andere Sprachen, für dieselbe Themenanzahl interpretieren muss.

Die Klassifizierungsmethode LDA würde sich wahrscheinlich besser für eine weitere explorative Analyse eignen.

Das SGDC-Modell hingegen ist für seinen simplen Aufbau sehr präzise, die TF-IDF Gewichtung innerhalb der Pipeline dürfte hierfür eine große Rolle spielen. Denn hier werden nur die seltensten Wörter stark gewichtet, somit kann bei einer geringen Datenmenge schon relativ präzise klassifiziert werden. Der Bias hinzu Ernährung und Umweltwissenschaft benötigt weitere Analysen, um herauszufinden welche Wörter bzw. Merkmale hier das Modell am meisten beeinflussen. Eine

Anpassung am Textkorpus könnte danach auch zu weiteren Leistungsverbesserungen führen. Weiterhin sollte auch hier die Unterteilung in die jeweiligen Sprachen und schwerer gewichtete Schlüsselwörter eine Verbesserung mit sich bringen. Wobei die Schlüsselwörter wahrscheinlich schon durch die TF-IDF Filterung weitestgehend abgedeckt sind. Um komplexere Fälle in den Dokumenten besser interpretieren zu können, kann das Einfügen von „N-Grams“ hilfreich sein. Bei „N-Grams“ werden häufig aufeinanderfolgende Wortketten als weiterer „Input-Layer“ für das Klassifikationsmodell eingefügt. Ein N-Gram könnte so bei gewissen Themen eine bessere Differenzierung mit sich bringen. Bei Dokumenten mit Textinhalten, die für beide Klassen gelten könnten, können Wortketten einen besseren Kontext liefern.

Wie auch bei dem LDA-Modell, kann für das SGDC-Modell eine weitere Gridsearch-Parametersuche durchgeführt werden, die Laufzeitdauer ist deutlich geringer, sodass eine komplexere Parametersuche über Nacht durchgeführt werden könnte. Das Training und die Bewertungen für das SGDC-Modell, über alle Datensatzgrößen (20 Millionen Dokumente bis 400 in 14 Schritten), wurde in unter zwei Stunden absolviert, bei kleineren Modellen sind die Trainingszeiten kaum länger als ein paar Sekunden. Auch sollte eine Gridsearch hier theoretisch einfacher zu implementieren sein, da dies als fertiges Modul von SciKit-Learn angeboten wird. (scikit-learn 2023c; Team 2020)

4.3 Vergleich der Modelle

Beide trainierten Modelle haben ihre Stärken und Schwächen, das LDA-Modell ist allerdings nicht der richtige Weg, um die Datenbank von LIVIO zuverlässig klassifizieren zu können. Hier würde es sich eher anbieten, die einzelnen Klassen genauer, mithilfe von LDA, anzuschauen. Das LDA-Modell ist durch seinen Aufbau besser für eine explorative Analyse geeignet, als zu versuchen es in bestehendes Format „zu pressen“. Die Ergebnisse zeugen allerdings auch davon, dass das LDA-Modell nicht besonders gut mit dem Textkorpus zurechtkommt. Mit weiteren Vorbereitungen am Textkorpus sollte auch das LDA Modell besser Themen finden können. Hier wäre, wie schon zuvor beschrieben, ein Versuch mit TF-IDF und N-Grams die Zeit gewiss wert. Ein weiterer großer Nachteil bleibt allerdings immer noch die um ein Vielfaches längere Trainingszeit. An dieser Stelle glänzt das SGDC-Modell, durch den simplen Aufbau der Pipeline kann das Modell in wenigen Sekunden große Datenmengen verarbeiten und klassifizieren, dass auch schon bei einem relativ simplen Textkorpus Aufbau und nur einem Input-Layer. Für ein produktives System dürfte dieser Faktor schon ein großer Pluspunkt zur Verwendung des SGDC-Modells sein. Auch sollte geprüft werden, inwieweit die Klassifizierungen des SGDC-Modells falsch sind. Die Klassifizierungsgrundlage von Averbis ist nicht bekannt, hier könnten also auch schon Fehler im „Golden Record“ sein, die das Ergebnis des SGDC-Modells verfälschen könnten, wenn auch unwahrscheinlich.
Grundsätzlich ist eine Implementierung des eines LDA-Modells zur Klassifizierung der LIVIVO Datenbank nicht empfehlenswert. Die Ergebnisse zeigen, wenn eines der Modelle aus dieser Arbeit Verwendung finden sollte, dann das SGDC-Modell. Der F1-Score ist mit Werten von 0.71 bis 0.81, je nach Klasse, schon recht gut. Vor allem in Anbetracht des doch recht simplen Textkorpus und der einfachen Funktionsweise des Modells.

5 Zusammenfassung und Schlussfolgerung

Die nächsten Schritte, um eine neue Klassifizierungsmethode für LIVIVO zu erstellen, sind vielfältiger Natur. Einerseits muss der Textkorpus erweitert werden, es müssen weitere Features hinzugefügt werden, andererseits besteht die Möglichkeit, die bestehende SGDC-Pipeline weiter zu verwenden und zu verbessern, oder aber andere Klassifizierungsmodelle auf ihre Leistung zu untersuchen. Features die dem Textkorpus oder als weitere Inputlayer[10] hinzugefügt werden können, sind N-Grams bzw. Bi-Grams, die Unterteilung der Daten in ihre Sprache, die Möglichkeit bei einigen Verlegern eine eindeutige Klasse zu zuteilen und eine Schlüsselwort Extraktion. (Beliga et al. 2015) Die Klassenzuordnung nach Verleger muss allerdings genau beobachtet werden, es könnten hier wieder ungewollte Neigungen im Modell zu einer bestimmten Klasse entstehen. Auch sollte untersucht werden, warum das SGDC-Modell einen leichten Bias gegenüber den Klassen Ernährung und Umweltwissenschaften entwickelt. Schuld daran könnten gewisse Wörter haben, die auch in anderen Klassen Anwendung finden, allerdings in einem geringeren Ausmaß, sodass das Modell von den Wörtern schnell dazu gebracht wird die Dokumente in den falschen Klassen zuzuordnen. In dem Fall könnten allerdings Schlüsselwörter und N-Grams Abhilfe schaffen, indem der kontextuale Inhalt des Dokumentes besser bewahrt wird. Weiterhin könnte ein Inputlayer aus dem Datenbankeintrag „Source“ entstehen, in diesem wird der Ursprung des Dokumentes festgehalten, genauer werden hier die Journalnamen gespeichert. Doch auch hier könnte wieder eine ungewünschte Klassifizierungsneigung entstehen, die manche Klassen bevorzugt verteilt. Eine aufwändige, aber möglicherweise Sinnvolle Erweiterung der Inputlayer wäre die Suche nach Entitäten, zum Bespiel Institutionen und Personen.

Um das SGDC-Modell etwas zu verbessern, könnte eine Gridsearch angewandt werden, hier wird das Modell mehrfach trainiert, jedoch jedes Mal mit etwas anderen, aber vordefinierten, Parametern. Die Datensatzgröße und die darin gespeicherten Daten bleiben dieselben, um eine möglichst hohe Vergleichbarkeit garantieren zu können. Die kurze Trainingszeit, vor allem bei kleinen Datensätzen würde den zeitlichen Aufwand entsprechend gering halten.

Eine weitere Chance, die vom SGDC-Modell klassifizierten Dokumente zu analysieren, wäre eine explorative Analyse mithilfe eines LDA-Modells. Hier würden die Stärken des unsupervised Modell zum Vorschein kommen und eine schnell interpretierbare Darstellung der Daten ermöglichen. So könnte möglicherweise schnell erkannt werden, welche Themen gut oder schlecht erkannt werden. Mit einem klassifizierten Datensatz des SGDC-Modells könnte eine LDA-Analyse aber auch zeigen, ob sich bei einer bestimmten Klasse eine andere Dokumentenklasse mit einschmuggelt. Anhand dieser Daten könnte dann versucht werden, die Input-Layer und den Textkorpus zu verbessern. Auch könnte mithilfe dieser Methode versucht werden, die Dokumente in feiner aufgeteilte Themengebiete zu klassifizieren. Zuvor sollte die Treffergenauigkeit des SGDC-Modells oder anderer gesteigert werden.

Das LDA bietet jedoch leider keine wirkliche Alternative zum SGDC oder Averbis-Modell, hierfür sind die gefundenen Topics des LDA-Modells nicht geeignet. Das SGDC-Modell ist offensichtlich auch keine wirkliche Alternative zum Klassifizierungsmodell von Averbis, bietet aber für weitere Arbeiten eine gute Grundlage für Leistungsvergleiche.

Weiterhin sollten auch andere Klassifizierungsmethoden auf die Daten angewandt werden, die Auswahl an Modellen ist schließlich gegeben. Andere interessante Vergleiche wären zum Beispiel Entscheidungsbäume, „Random Forrest“-Modelle, Transformer-Modelle, neuronale Netze und „Voting-Classifier“. (scikit-learn 2023a)

[10] Klassifikationsmodelle können mit mehreren Variablen verschiedener Natur gespeist werden, diese Eingangsvariablen werden unter anderem als „Inputlayer“ bezeichnet

Voting-Classifier sind eine interessante Möglichkeit, verschiedene Modelle miteinander zu verbinden. Die im Voting-Classifier verwendeten Modelle werden zuvor einzeln am Datensatz trainiert und gemeinsam auf den Testdatensatz angewendet. Bei der Anwendung wird den einzelnen Modellen eine Stimme gegeben, mit dieser „wählen“ die Modelle ihre errechnete Klasse und geben ihre Stimme an den Voting-Classifier weiter. Über das Mehrheitsprinzip wird dann eine Klasse bzw. vom Voting-Classifier vergeben. Die Chance bei diesem Verfahren ist, die Mehrheit der Modelle bestimmen zu lassen, so kann in manchen Fällen ein Bias eines einzelnen Modells „unterdrückt“ werden. Wichtig für einen Voting-Classifier ist, dass dieser mit mehreren Modellen arbeitet, ein Voting-Classifier mit nur zwei Modellen ist wenig sinnvoll. Dann kann bei entgegensetzten Stimmen das Modell keine sinnvolle Aussage treffen. Es werden somit mindestens 3 verschiedene Modelle benötigt, um einen Voting-Classifier zum Laufen zu bringen. Durch diese Mindestvoraussetzung ist der Aufbau des Klassifizierungsmodell aufwändig, auch sollten die verwendeten Modelle schon weitestgehend gute Leistungen vorweisen. Denn aus drei schlechten Modellen kann der Voting-Classifier kein gutes Modell machen. Grundsätzlich gibt es noch viel Optimierungs- und Verbesserungspotential am Textkorpus bzw. an den Inputlayern für die zukünftigen Modelle. Eine Leistungsgrundlage ist geschaffen, an der sich andere Modelle messen und verbessern können. Der verwendete Code wird in GitHub zur Verfügung gestellt und kann so weiterverwendet werden (siehe Anhang).

6 Anhang

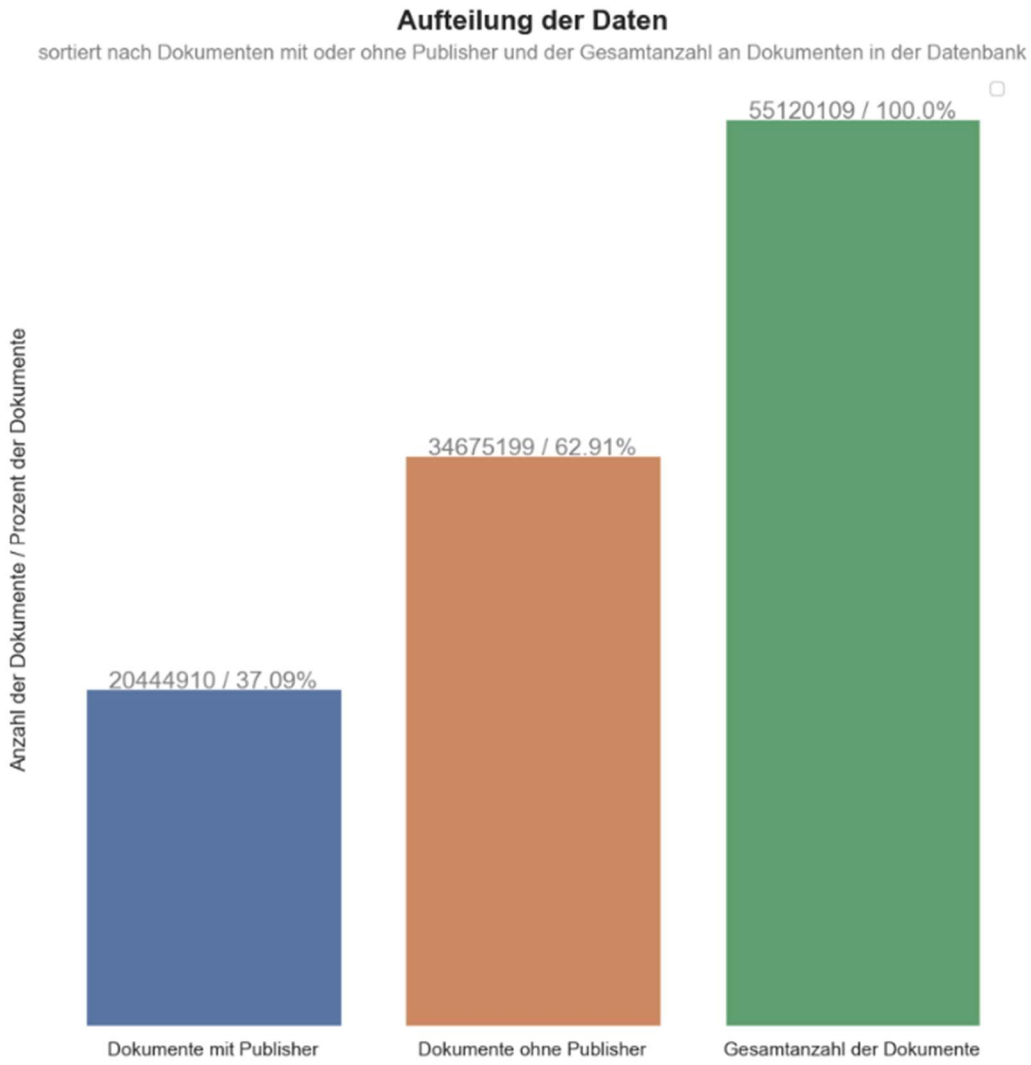

Abbildung 5-1: Aufteilung der Daten sortiert nach Dokumenten mit oder ohne Publisher, der gesamten Datenbank

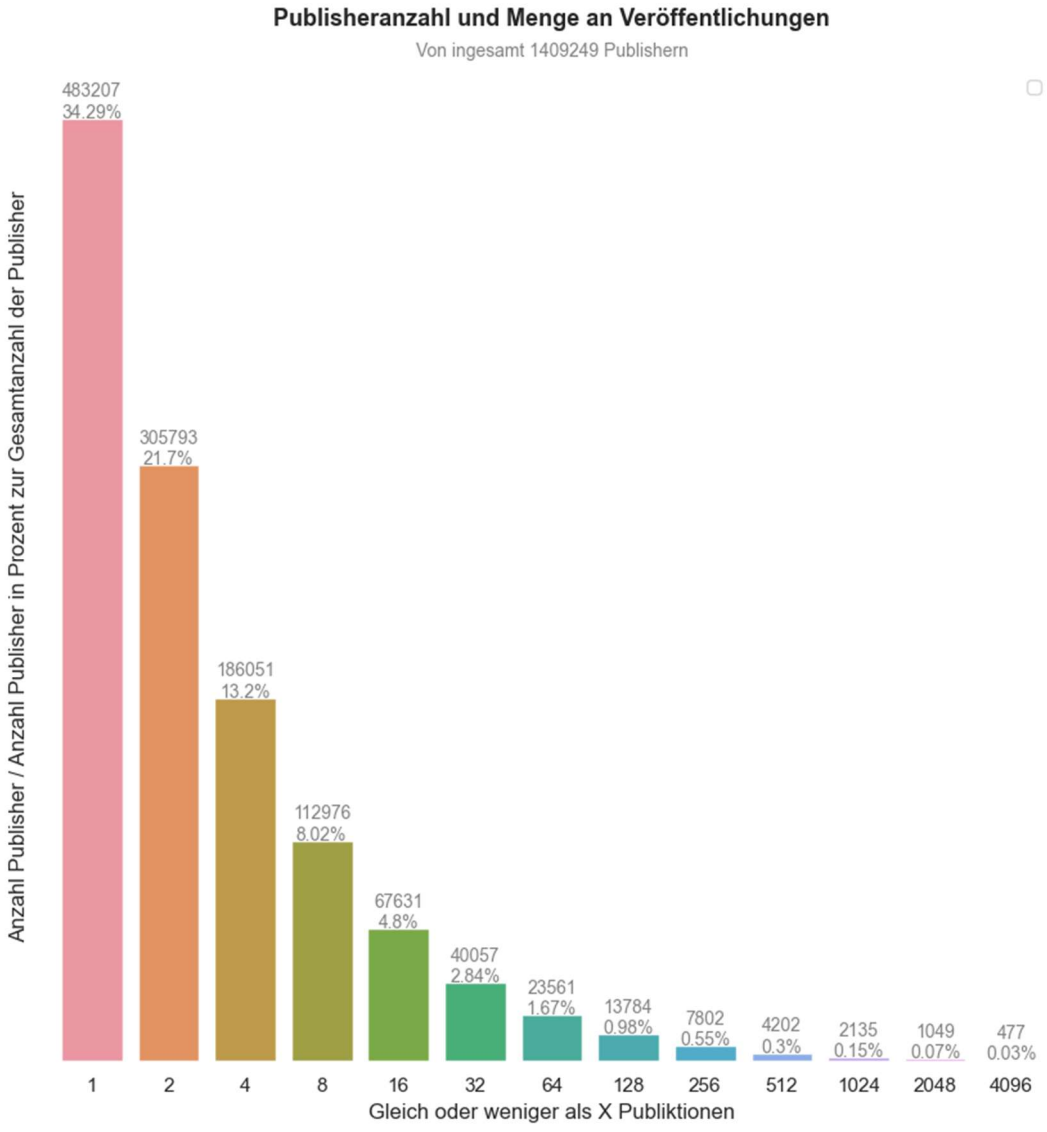

Abbildung 6-2: Publisheranzahl und Menge der Veröffentlichungen

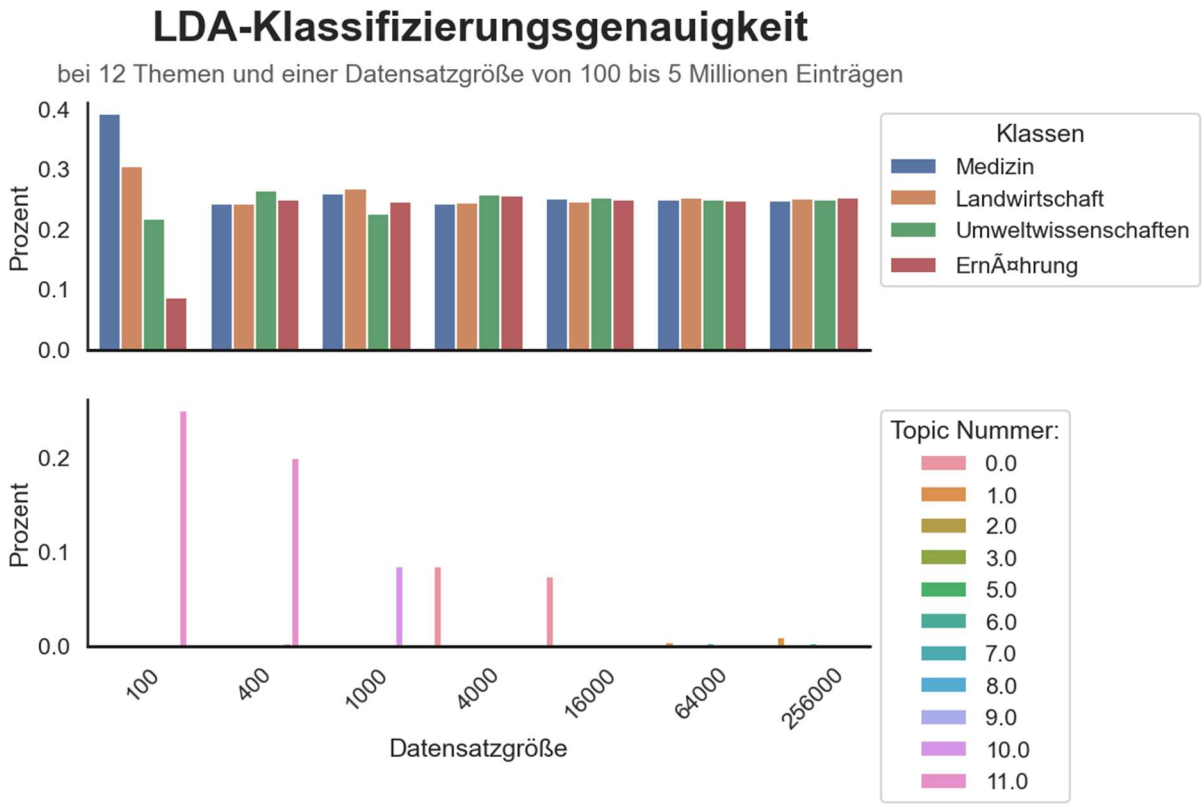

Abbildung 6-3: LDA-Klassifizierungsgenauigkeit bei 12 Topics

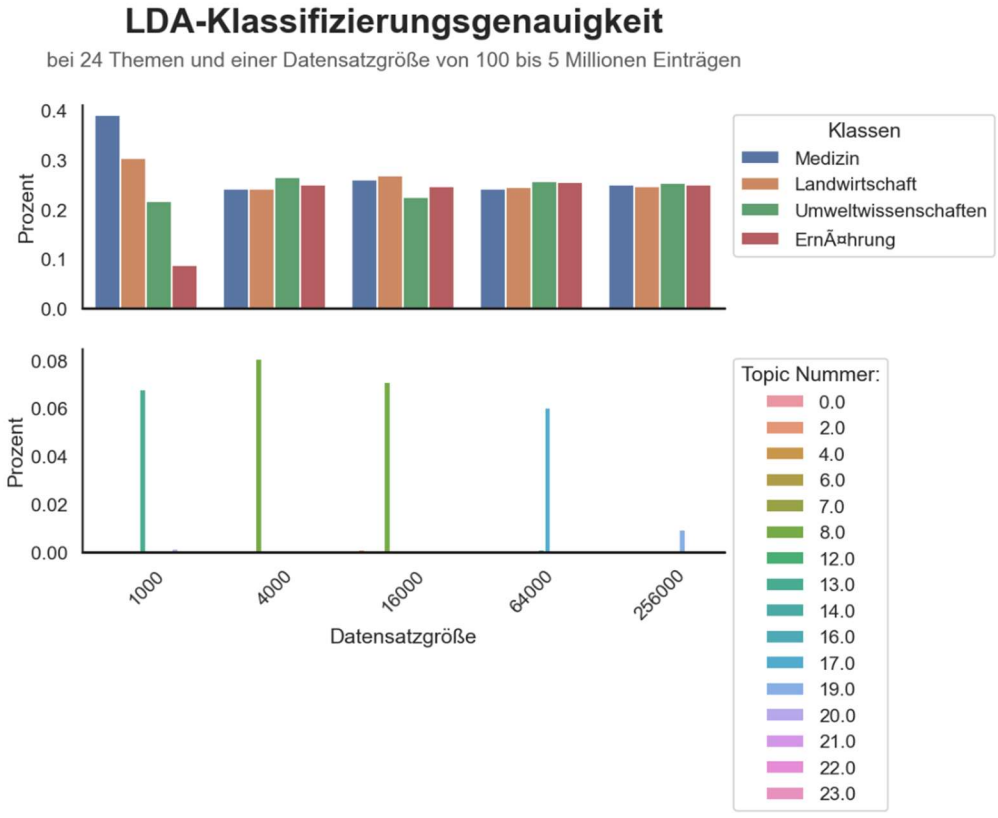

Abbildung 6-4: LDA-Klassifizierungsgenauigkeit bei 24 Topics

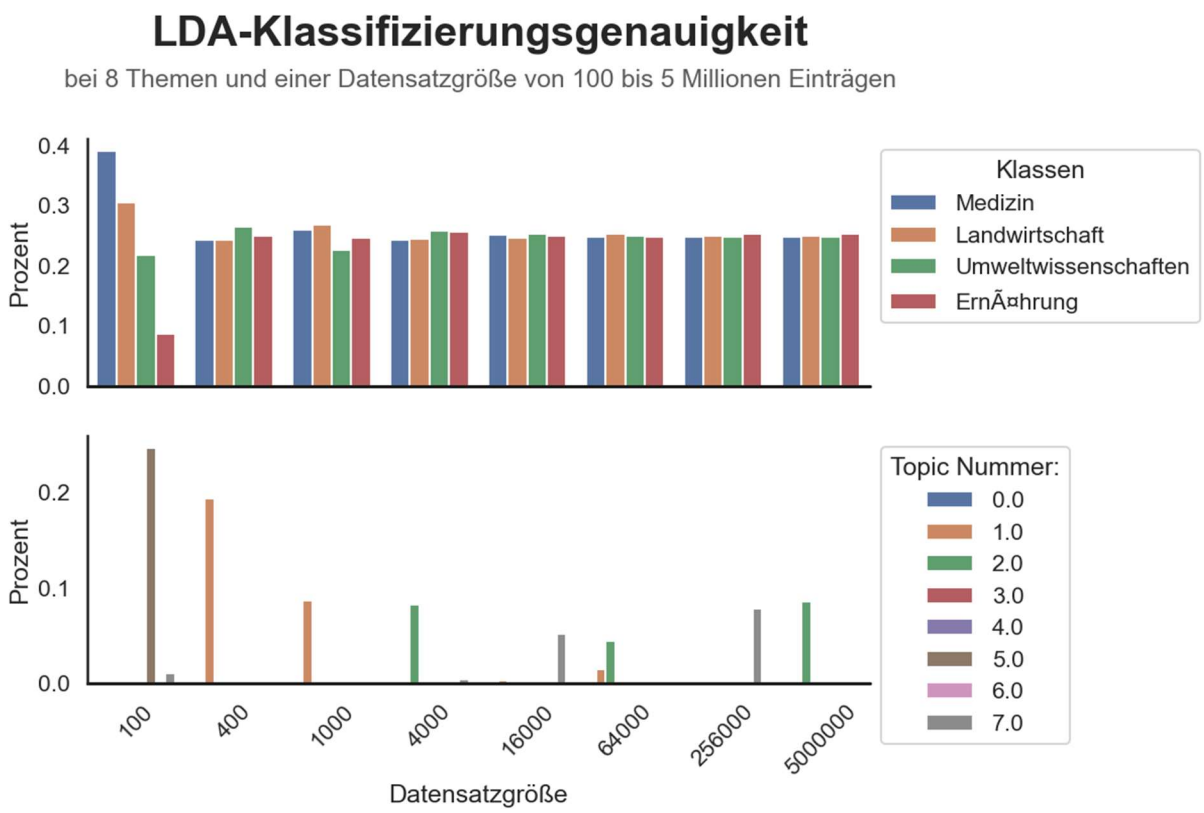

Abbildung 6-5: LDA-Klassifizierungsgenauigkeit bei 8 Topics

topic	1	3	nan
class			
Ernährung	2197	11	2
Landwirtschaft	2196	9	2
Medizin	2168	12	1
Umweltwissenschaften	2177	22	3

Abbildung 6-6: Confusion-Matrix des LDA Modells

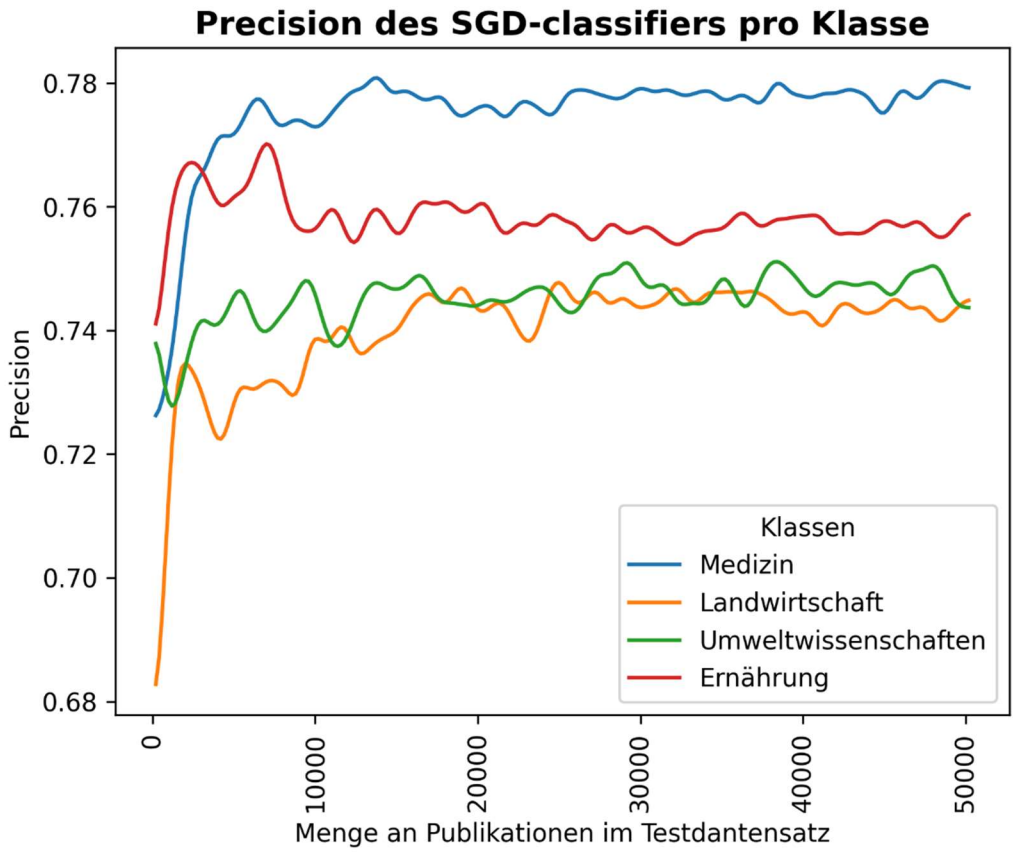

Abbildung 6-7: Precision des SGD-Classifiers pro Klasse

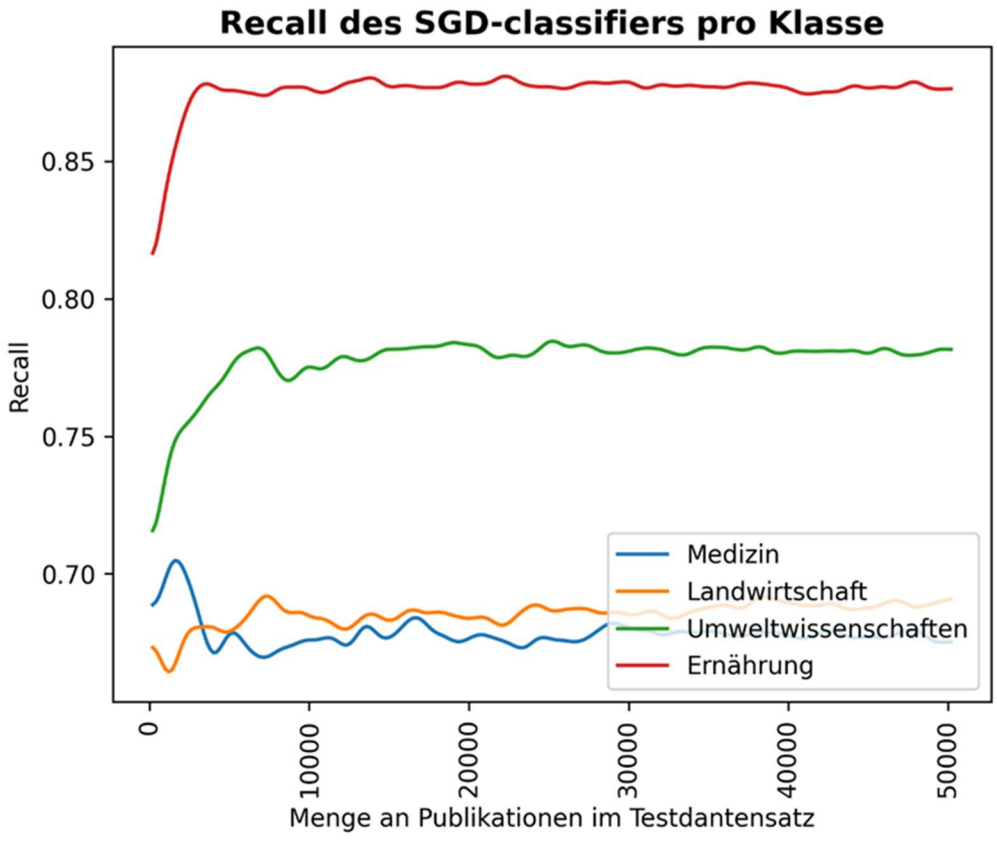

Abbildung 6-8: Recall des SGD-classifiers pro Klasse

```
text_clf = Pipeline([
      ('vect',
CountVectorizer(lowercase=False,stop_words=None,tokenizer=None,min_df=3)),
      ('tfidf', TfidfTransformer(use_idf=True,norm="l2")),
      ('clf', SGDClassifier(loss='hinge', penalty='l2',
                            alpha=1e-3, random_state=42,
                            max_iter=5, tol=None)),
   ])
   text_clf.fit(df_train['combined'], df_train['class'])
   predicted = text_clf.predict(df_test['combined'])
   f1_matrix = metrics.classification_report(df_test['class'], predicted,
      target_names=['Medizin',
'Landwirtschaft','Umweltwissenschaften','ErnÃ¤hrung'],output_dict=True)
   f1_matrix = pd.DataFrame(f1_matrix).transpose()
   f1_matrix.to_csv("sgd_score_cvs\F1_matrix_"+str(size)+".csv")
   size = size - 200
```

Abbildung 6-10: Python Code zum Training & Validierung des SGDC Modells

```
df_med = df.loc[df['class'] == "Medizin"].head(int(size))
df_land = df.loc[df['class'] =='Landwirtschaft'].head(int(size))
df_umwelt = df.loc[df['class'] =='Umweltwissenschaften'].head(int(size))
df_ern = df.loc[df['class'] =='ErnÃ¤hrung'].head(int(size))
df = pd.concat([df_med, df_land,df_umwelt,df_ern])
counted, lowest_c = count_class_pop(df)
df_med = df.loc[df['class'] == "Medizin"].head(int(lowest_c))
df_land = df.loc[df['class'] =='Landwirtschaft'].head(int(lowest_c))
df_umwelt = df.loc[df['class'] =='Umweltwissenschaften'].head(int(lowest_c))
df_ern = df.loc[df['class'] =='ErnÃ¤hrung'].head(int(lowest_c))
df = pd.concat([df_med, df_land,df_umwelt,df_ern])
counted, lowest_c = count_class_pop(df)
print("Anzahl der Publiktationen pro Klasse:" ,int(lowest_c), "| ingesamt:" ,
int(lowest_c*4)," Publikationen")
df_train, df_test = train_test_split(df, test_size=0.25)
```

Abbildung 6-9: Python-Code zur gleichmäßigen Aufteilung der Daten in Test/Train

GitHub Repository:
https://github.com/MTPKo/Bachelor-Arbeit

7 Literaturverzeichnis

Averbis GmbH (2022): Information Discovery - Averbis GmbH. Online verfügbar unter https://averbis.com/de/information-discovery/, zuletzt aktualisiert am 02.11.2022, zuletzt geprüft am 31.03.2023.

Beliga, Slobodan; Ana, Meštrović; Martinčić-Ipšić, Sanda.: An Overview of Graph-Based Keyword Extraction Methods and Approaches. In: Journal of Information and Organizational Sciences, Bd. 39, S. 1–20.

Blei, David M.; Ng, Andrew Y.; Jordan, Michael I. (2003): Latent Dirichlet Allocation. In: *Journal of Machine Learning Research* 3 (Jan), S. 993–1022. Online verfügbar unter https://jmlr.csail.mit.edu/papers/v3/blei03a.html.

Ganesan, Kavita (2014): What are Stop Words? In: *Kavita Ganesan*, 19.10.2014. Online verfügbar unter https://kavita-ganesan.com/what-are-stop-words/, zuletzt geprüft am 30.03.2023.

Ganesan, Kavita (2018): What are N-Grams? In: *Kavita Ganesan*, 02.11.2018. Online verfügbar unter https://kavita-ganesan.com/what-are-n-grams/, zuletzt geprüft am 30.03.2023.

Ganesan, Kavita (2019): What are Stop Words? In: *Opinosis Analytics*, 06.04.2019. Online verfügbar unter https://www.opinosis-analytics.com/knowledge-base/stop-words-explained/, zuletzt geprüft am 30.03.2023.

Gensim: topic modelling for humans (2022). Online verfügbar unter https://radimrehurek.com/gensim/, zuletzt aktualisiert am 21.12.2022, zuletzt geprüft am 30.03.2023.

Google Developers (2022): Klassifizierung: Gewichtung nach Vorhersage | Machine Learning | Google Developers. Online verfügbar unter https://developers.google.com/machine-learning/crash-course/classification/prediction-bias?hl=de, zuletzt aktualisiert am 27.09.2022, zuletzt geprüft am 31.03.2023.

Lai, Siwei; Xu, Liheng; Liu, Kang; Zhao, Jun (2015): Recurrent Convolutional Neural Networks for Text Classification. In: *AAAI* 29 (1). DOI: 10.1609/aaai.v29i1.9513.

LIVIVO - Help - About LIVIVO (2023). Online verfügbar unter https://www.livivo.de/app/misc/help/about, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

LIVIVO - Informationen über Datenquellen (2023). Online verfügbar unter https://www.livivo.de/app/misc/dbinfo?LANGUAGE=de&dbid=AGRICOLA, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

Menzli, Amal (2022): Tokenization in NLP: Types, Challenges, Examples, Tools. In: *neptune.ai*, 21.07.2022. Online verfügbar unter https://neptune.ai/blog/tokenization-in-nlp, zuletzt geprüft am 30.03.2023.

Oppermann, Artem (2021): Accuracy, Precision, Recall, F1-Score und Specificity - KI Tutorials. Online verfügbar unter https://artemoppermann.com/de/accuracy-precision-recall-f1-score-und-specificity/, zuletzt aktualisiert am 04.11.2022, zuletzt geprüft am 30.03.2023.

Pandarallel documentation (2022). Online verfügbar unter https://nalepae.github.io/pandarallel/, zuletzt aktualisiert am 15.03.2022, zuletzt geprüft am 31.03.2023.

Saumyab271 (2022): Stemming vs Lemmatization in NLP: Must-Know Differences. In: *Analytics Vidhya*, 28.06.2022. Online verfügbar unter

https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/, zuletzt geprüft am 30.03.2023.

Saxena, Shipra (2021): Beginner's Guide to Support Vector Machine(SVM). In: *Analytics Vidhya*, 08.03.2021. Online verfügbar unter https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/, zuletzt geprüft am 30.03.2023.

scikit-learn (2023a): 1.11. Ensemble methods. Online verfügbar unter https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 31.03.2023.

scikit-learn (2023b): sklearn.linear_model.SGDClassifier. Online verfügbar unter https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

scikit-learn (2023c): sklearn.model_selection.GridSearchCV. Online verfügbar unter https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

Supervised vs. Unsupervised Learning: What's the Difference? (2021), 12.03.2021. Online verfügbar unter https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning, zuletzt geprüft am 30.03.2023.

Team, Great Learning (2020): Hyperparameter Tuning with GridSearchCV. Online verfügbar unter https://www.mygreatlearning.com/blog/gridsearchcv/, zuletzt aktualisiert am 13.06.2022, zuletzt geprüft am 30.03.2023.

Text classification (2023). Online verfügbar unter https://huggingface.co/docs/transformers/tasks/sequence_classification, zuletzt aktualisiert am 31.03.2023, zuletzt geprüft am 31.03.2023.

Text Classifiers in Machine Learning: A Practical Guide (2023). Online verfügbar unter https://levity.ai/blog/text-classifiers-in-machine-learning-a-practical-guide, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSci (2023). Online verfügbar unter https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

Waldemar Dzeyk, Kornél Markó (2008): Optimizing and evaluating the MEDPILOT search engine. Boosting medical information retrieval by using a morpheme thesaurus. Journal of the European Association for Health Information and Libraries, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.

XRDS (2017): An Introduction to N-grams: What Are They and Why Do We Need Them? - XRDS. Online verfügbar unter https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/, zuletzt aktualisiert am 21.10.2017, zuletzt geprüft am 30.03.2023.

ZB MED - Informationszentrum Lebenswissenschaften (2023): LIVIVO-Suchportal. Online verfügbar unter https://www.zbmed.de/recherchieren/livivo, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 30.03.2023.