# Grundprojekt: Digital Journalism
# Automatisierte Dossier-Erstellung mittels Textmining

Jan Paul Assendorp

Hamburg University of Applied Sciences, Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany
janpaul.assendorp@haw-hamburg.de

## I. Einleitung

Im Journalismus gibt es zahlreiche Prozesse, die anhand von Methodiken des *Knowledge Discovery in Databases (KDD)* und damit vor allem anhand des *Textmining* optimiert werden können. Eine große Herausforderung ist dabei die gewinnbringende Verzahnung der Fachkenntnis von Domänen-Experten in bekannte Mining-Prozesse. Die hier beschriebene Projektarbeit behandelt das semi-automatisierte Erstellen von Pressedossiers in Form eines Vorschlagsystems, das über maschinelles Lernen den Prozess der Dossier-Erstellung verbessern soll.

Im Detail werden in dieser Ausarbeitung die Ergebnisse des Grundprojektes und ein Konzept für die kommende Projektarbeit vorgestellt. Im Abschnitt II wird die Fragestellung formuliert, die der Projektarbeit zugrunde liegt. In Bezug auf diese Fragestellung werden zunächst in III aus der Literatur relevante Projekte zusammengetragen und hinsichtlich der Fragestellung untersucht. Darauf folgend wird im Abschnitt IV eine Konzeption vorgestellt, welche zu der beschriebenen Fragestellung Ergebnisse liefern soll. Im letzten Abschnitt V werden dazu erste Versuche beschrieben, die im Zuge des Grundprojektes bereits erfolgt sind.

## II. Fragestellung

Es soll eine Gesamtsystem entwickelt werden, welches anhand eines geeigneten Klassifikators als Vorschlagsystem für die Erstellung von Pressedossiers dient.

Als Dossier wird dabei eine Zusammenstellung semantisch verknüpfbarer Dokumente verstanden, die unter einem ausgewählten Leitthema zusammengefasst werden. Bei den darin enthaltenen Dokumenten handelt es sich um journalistische Artikel aus einem gegebenen Dokumentenkorpus. In ein solches Dossier sollen eine ausgewählte Anzahl von Artikeln des Korpus einfließen, die sich nach einem bestimmten Maßstab hinreichend in den entsprechenden thematischen Hintergrund des Dossiers einordnen lassen.

Dossiers werden durch einen Experten der Fachdomäne erstellt, welcher den Themenbezug festlegt und dazu aus dem Dokumentenkorpus passende Artikel auswählt. Ein Dossier besteht z.B. aus 30 journalistischen Artikeln. Inhaltlich kann das Thema eines Dossiers u.a. kulturelle oder politische Ereignisse umfassen.

Unter der Annahme, dass der betrachtete Dokumentenkorpus beliebig groß sein kann und die Zahl der Dokumente stetig steigt, ist die Selektion von relevanten Artikeln zur Erstellung eines Dossiers besonders zeitaufwändig und kompliziert. An dieser Stelle soll das Vorschlagsystem Abhilfe schaffen, indem es dem Domäne-Experten eine Vorselektion von relevanten Artikeln zu dem gegebenen Leitthema zur Verfügung stellt. Diese Vorselektion soll durch benutzerspezifische Modelle zur Klassifikation von Dokumenten erstellt werden. Das Ziel ist, die Erstellung von Dossiers hinsichtlich der Vorselektion zu automatisieren und dadurch den Prozess der Dossier-Erstellung zu optimieren.

Das Erstellen eines Dossiers soll nach wie vor in letzter Instanz durch einen Domänen-Experten redaktionell erfolgen. Die vollständige Automatisierung der Expertise eines Journalisten wird als ein unrealistisches Ziel eingeschätzt. Vielmehr wird auf die Fachkenntnis des Domänen-Experten aufgebaut und dieser durch das Vorschlagsystem in seiner Arbeit unterstützt. Nach diesem Ansatz ist es daher wichtig, dass alle potentiell relevanten Artikel zu einem Thema eines Dossiers erfasst werden. False-Positives sind durch die finale Selektion des Benutzers unproblematisch. Dagegen sind False-Negatives als kritisch zu betrachten. Das Ziel ist es, die relevanten Artikel zu erfassen und darüber hinaus weitere, ähnliche Dokumente anzubieten. Aus dieser reduzierten Menge kann der Domänen-Experte mit reduziertem Aufwand Dokumente auswählen und in einem Dossier zusammenstellen.

## III. Literature Review

Aus der aktuellen Forschung im Bereich des Textmining stechen einige Projekte durch ihren Umgang mit Text und den daraus resultierenden Ergebnissen besonders hervor. Im Folgenden soll eine Auswahl dieser Projekte vorgestellt und hinsichtlich der vorangehenden Fragestellung dieser Arbeit untersucht werden. Dabei handelt es sich auf der einen Seite um Projekte, welche direkt in Bezug auf die Fragestellung angewendet werden können. Auf der anderen Seite werden Projekte vorgestellt, welche indirekt, durch die verwendete Methodik zur Verarbeitung von Text für diese Fragestellung, relevant sind. Dazu zählt das Entity-Linking-Framework *Dexter*, das im folgenden Abschnitt III-A vorgestellt wird. Des weiteren wird in III-B *Carrot2* als Tool zum Clustern eines Dokumentenbestandes betrachtet. In III-C wird das *Overview-Project* vorgestellt, welches sich direkt an Journalisten zur Optimierung ihrer Prozesse richtet. Die genannten Projekte bauen auf der Bibliothek zur Verarbeitung und Analyse von Text *Apache Lucene* auf. Aus diesem Grund erfolgt schließlich in III-D eine detaillierte Betrachtung der Mechanismen und Funktionsweise der Lucene-Bibliothek.

### A. Dexter

Bei dem Tool Dexter handelt es sich um ein Open-Source *Entity-Linking* Framework zur semantischen Verknüpfung von unstrukturierten Daten mit Einträgen aus einer Wissensdatenbank [1]. Dexter analysiert dazu Texte wie z.B. Nachrichten- oder Blog-Einträge und verknüpft darin enthaltene, relevante Terme mit dazugehörigen Artikeln, die aus dem Wikipedia-Dump bezogen werden.

Dexter ist für diese Arbeit vor allem durch die grundlegende Architektur zur Verarbeitung von Texten interessant. Um die zuvor beschriebenen Terme für den Entity-Linking-Prozess zu identifizieren, muss der Text zunächst anhand von Textmining-Praktiken analysiert werden. Zur Extraktion von Features des Textes wird dabei auf Funktionalitäten von Apache Lucene zurückgegriffen[1].

Im Gegensatz dazu verwendet Dexter allerdings keine Textbasierten Verfahren zur Bestimmung der Ähnlichkeit zwischen z.B. Wikipedia-Artikeln wie das im Textmining weit verbreitete Kosinus-Ähnlichkeitsmaß [2], weil diese für den Anwendungsfall zwar gute Ergebnisse liefern, dabei aber Ressourcenlastiger sind [3]. Dagegen wird ein Machine-Learning-Ansatz verwendet, welcher auf den Anwendungsfall optimiert ist [3].

### B. Carrot2

*Carrot2* ist ein Open Source Framework für Anwendungsfälle im Bereich des Information Retrival und Webmining. Besonders die Funktionalität von Carrot2 zur Cluster-Analyse von Text findet Anwendung zur visuellen Darstellung von z.B. Suchergebnissen in Produktivsystemen oder Tools wie z.B. Apache Solr zur Suche und Verarbeitung von Text.

Der in Carrot2 integrierte Cluster-Algorithmus *Lingo* transformiert zunächst die zu verarbeitenden Texte in das Vector-Space-Model und damit in Feature-Vektoren in einem mehrdimensionalen Feature-Raum. Besonders interessant ist das Reduzieren dieser Feature-Vektoren anhand des auf *Latent-Semantic-Indexing* (LSI) basierenden Verfahrens *Singular Value Decomposition* (SVD) welches eine algebraische Methode zur Dekomposition von Matrizen zur Bestimmung der orthogonalen Basis der Matrix beschreibt [4]. Darüber lassen sich aus der Term-Dokument-Matrix einzelne Terme oder Gruppen von Termen bestimmen, welche sich besonders voneinander abheben und somit entscheidende Features des Textbestandes darstellen [4].

Carrot2 implementiert einige benötigten Methodiken zum Lösen der häufig auftretenden Aufgaben im Textmining wie z.B. das Tokenizing, das Entfernen von Stopwords oder das Stemming von Wörtern in verschiedenen Sprachen [5].

Der Clustering-Algorithmus von Carrot2 wurde bereits in ersten Versuchen verwendet, deren Ergebnisse im Abschnitt V beschrieben sind.

### C. Overview Project

Das Open Source Tool *Overview*[2] dient der Optimierung von Prozessen im Journalismus, indem Dokumente eines großen Datenbestandes automatisch nach Leitthemen sortiert werden. Bekanntheit erlangte das Tool z.B. durch eine Analyse der Snowden-Files [6].

Overview verwendet ein Clustering-Verfahren, anhand dessen die Dokumente in hierarchischen Clustern übersichtlich visualisiert werden können [7]. Dazu wird im Hintergrund das Tool ElasticSearch[3] verwendet, welches auf der im Abschnitt III-D vorgestellten Bibliothek Lucene zur Textsuche basiert. Das Overview Projekt zeichnet sich vor allem durch sein Team aus Entwicklern und Journalisten aus. So ist z.B. der Projektleiter *Jonathan Stray* fachlich in den beiden Domänen Informatik und Journalismus angesiedelt.

Für diese Arbeit ist Overview eine gutes Beispiel, wie anhand von Textmining Prozesse des journalistischen Umfeldes optimiert werden können. Auch ähnelt die Herangehensweise der Verarbeitung von Text anhand von Lucene der im Abschnitt IV vorgestellten Konzeption. Durch den offenen Quellcode von Overview lassen sich dadurch interessante Konzepte im Detail betrachten.

### D. Lucene Search Library

Die Search-Library Lucene wurde 1997 von Doug Cutting entwickelt und 2001 der ASF[4] gespendet [8]. Seitdem ist Apache Lucene eine weit verbreitete Open Source Bibliothek die im Bereich der Verarbeitung und Analyse von Text Anwendung findet und stetig verbessert wird. In dem folgenden Abschnitt III-D1 werden die Grundlagen von Apache Lucene beschrieben und in den Abschnitten III-D2 und III-D3 auf Projekte der ASF eingegangen, die in einem Ökosystem um Lucene als Teil der Infrastruktur verwendet werden.

*1) Apache Lucene:* Apache Lucene bietet als Open Source Java-Bibliothek zahlreiche Funktionen zum Indexing, Querying, Hightlighting und der Analyse von Text [8]. Die Funktionalitäten von Lucene lassen sich grundlegend in die folgenden Kategorien einordnen:

- Die Analyse von Dokumenten anhand von verschiedenen Mechanismen
- Das Indizieren von Dokumenten und persistente Speichern des Indexes
- Die Suche anhand einer Query unter Verwendung des Indexes

Um eine Textsuche anhand von Lucene zu realisieren wird zunächst ein Index zu einem Bestand an Daten erzeugt. Dieser Index bietet die Möglichkeit, Datenbestände performant zu durchsuchen. Dazu werden die Dokumente des Datenbestandes in eine interne Repräsentation konvertiert. Die von dem Anwender definierten Dokumente können dabei aus einem oder mehreren Feldern mit Inhalt bestehen, die durch Lucene unterschiedlich verarbeitet werden können [8]. Typische Felder bei journalistischen Dokumenten können z.B. Einleitung, Hauptteil oder die Zusammenfassung eines Artikels sein.

Der Prozess des Indexing ist in Abbildung 1 schematisch dargestellt. Felder eines Dokumentes werden anhand eines Tokenitzers in einen Token-Strom gewandelt. Ein Token des Stroms wird repräsentiert durch eine Menge an Attributen [8]. Neben dem tatsächlichen Term des Tokens können Attribute z.B. die Position des Tokens im Dokument in Form eines Offsets sein. Auch können so anwendungsspezifische Attribute wie z.B. *Part-of-Speech*-Tags abgebildet werden.

[1] Siehe Dexter Sourcecode: https://github.com/dexter/dexter
[2] Overview Project: http://overview.ap.org
[3] Siehe Overview Sourcecode: https://github.com/overview
[4] Apache Software Foundation

Auf den Strom aus Token können verschiedene Token-Filter angewendet werden. Zu diesen Token-Filtern gehören z.B. Filter zum Entfernen von Stopwords, Funktionen zur Normalisierung und Stemming des durch das Token repräsentierten Terms oder das Ersetzen von Synonymen.

Lucene verwendet einen invertierten Index als interne Repräsentation der Dokumente [8]. Das Prinzip des invertierten Indexes wurde von C.J. van Rijsbergen in [9] vorgestellt. Der Datenstrom aus einzelnen Termen des Dokumentenkorpus wird invertiert, sodass sich über einen Term auf alle Dokumente schließen lässt, in denen dieser vorkommt. Dazu lassen sich Informationen, wie z.B. die Term-Frequency oder der Offset speichern, welcher die Position des Terms in einem Feld bestimmt. Der invertierte Index ist zudem sortiert, damit Zugriffe auf einen Schlüssel möglichst effizient sind [10].

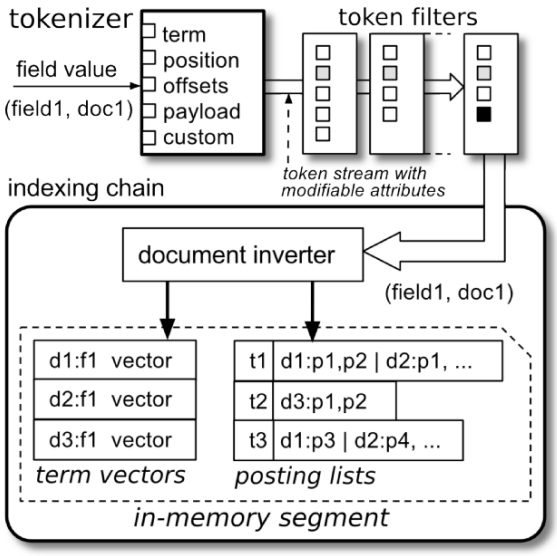


Abbildung 1. Lucene Indexing-Prozess [8]

Lucene invertiert zunächst den Strom aus Token zu einem Feld, der durch einen Tokenizer zu einem Dokument geliefert wird. Abbildung 1 zeigt zudem, wie dazu Term-Vektoren der Felder einzelner Dokumente gespeichert werden. Zu einzelnen Termen werden Posting-Lists erstellt bzw. erweitert, in denen zu einem Term als Key eine Liste der Attribute des Terms gespeichert werden [8].

Ein Anwender kann nun in Form einer Query im Dokumentenbestand suchen. Die Query kann dazu eine Menge an Termen sein, oder ein Dokument mit verschiedenen Feldern. Die Verarbeitung der Query durch Lucene erfolgt dabei analog zu der Indizierung von Dokumenten. Die Query wird zunächst durch einen Tokenizer in einen Strom an Token transformiert. Diese werden anhand von Filter normalisiert und reduziert. Nach diesem Prozess sind die Terme der Token kompatibel zu den im invertierten Index gespeicherten Terme. Anhand eines Algorithmus zum Vergleich einer Menge an Token können somit im Index enthaltene Dokument identifiziert werden, welche der Menge von Token aus der Query ähnlich sind. Die Informationen über ähnliche Dokumente können daraufhin dem Benutzer als Ergebnis geliefert werden.

*2) Apache Tika:* Apache Tika war als Komponente zunächst ein Teil von Lucene, wurde allerdings als eigenständiges Projekt ausgegliedert. Bei Tika handelt es sich um einen umfangreichen Parser zum Verarbeiten von Dokumenten, die in verschiedenen Formaten vorliegen können. Auch kann Tika dazu verwendet werden, Metainformationen aus Dokumenten zu extrahieren

Apache Tika erkennt zahlreiche Dateiformate anhand von Glob-Pattern, Magic-Byte-Pattern, oder XML-Root-Elementen [11]. Aus diesem Grund wird Tika häufig in einer Infrastruktur Apache Lucene vorgeschaltet, um Text aus heterogenen Dokumenten zu extrahieren, die anhand von Lucene verarbeitet werden können.

*3) Apache Solr:* Das ASF Projekt Apache Solr integriert Lucene in eine eigenständige Anwendung, die ohne viel zusätzlichen Aufwand ausgeliefert und ausgeführt werden kann.

Dazu besteht Solr aus einer Webanwendung inklusive einer umfangreichen REST-API, die dem Anwender den vollen Zugriff auf die Funktionalitäten von Lucene bietet. Solr wurde im Rahmen dieser Arbeit für erste Versuche verwendet, die im Abschnitt V beschrieben werden.

## IV. Konzeption

Zu der im Abschnitt II beschriebenen Fragestellung soll ein Vorschlagsystem realisiert werden. Anhand dieses Systems soll die Erstellung von Dossiers auf Grundlage von journalistischen Artikeln eines Dokumentenkorpus teilweise automatisiert werden. Dennoch wird der Anwender vor allem in Bezug auf seine Fachkenntnis der Domäne in die Erstellung von Dossiers einbezogen. Das System soll zu einem gegebenen Leitthema bzw. Focal-Point eine Menge relevanter Artikel anbieten.

Im folgenden Abschnitt IV-A wird der hier betrachtete Anwendungsfall und damit auch der zu verwendende Dokumentenkorpus betrachtet. In IV-B wird die Architektur des zu entwickelnden Vorschlagsystems dargestellt. Darauf folgend wird in IV-C die Methodik der Klassifikation und in IV-D die Anwendung eines künstlichen neuronalen Netzes zur Klassifikation untersucht.

### A. Beschreibung des Anwendungsfalls

Als Dokumentenkorpus soll der Bestand des journalistischen Netzwerkes *Eurozine*[5] verwendet werden. Eurozine ist ein Netzwerk europäischer Kulturzeitschriften, bestehend aus etwa 80 Partnerzeitschriften mit explizitem Interesse an transnationalen Debatten [6]. Der englische Korpus von Eurozine umfasst 2500 Artikel, welche kulturelle und gesellschaftliche Debatten aus europäischer Perspektive behandeln [6].

Die Artikel liegen im XML-Format in semi-strukturierter Form vor. Dabei weisen die Artikel zum Teil Unstimmigkeiten bzgl. des spezifizierten Schemas auf. So sind z.B. Metainformationen nicht konsistent gekennzeichnet [6]. Im Zuge der vorangehenden Arbeit in [12] wurde allerdings der Korpus bereits hinreichend angepasst, sodass Textmining-Verfahren angewendet werden können.

Das Eurozine-Netzwerk bietet Dossiers in Form von *Focal-Points* an. Diese bündeln zwischen 20 und 60 Artikel zu einem aktuellen und übergreifenden Thema. Diese Zusammenstellungen werden redaktionell und unter großem zeitlichen Aufwand erstellt, wodurch Focal-Points zu tagesaktuellen Themen zu aufwändig werden [6].

[5]Eurozine: http://www.eurozine.com

Der Dokumentenkorpus von Eurozine bietet keine Verschlagwortung. Allerdings wurden in [6] bereits Versuche durchgeführt, die Schlagwörter des Archivs zu extrahieren und in Form von Tag-Clouds zu visualisieren. Auf die daraus resultierenden Erkenntnisse kann in dieser Arbeit aufgebaut werden.

### B. Architektur des Recommender Systems

Das zu realisierende Vorschlagsystem soll in der Lage sein, anhand eines geeigneten Klassifikators Dokumente eines Dokumentenkorpus in vorgegebene Klassen einzuordnen. Unter Verwendung der in IV-A vorgestellten Focal-Points soll einen Klassifikator trainiert werden. Dieser soll demnach in der Lage sein, vorgegebene Focal-Points zu reproduzieren und diesen weitere ähnliche Dokumente zuzuordnen.

In Abbildung 2 sind die dazu notwendigen Komponenten schematisch aufgeführt. Die Dokumente eines Focal-Points werden anhand von geeigneten Mechanismen zur Feature-Selection und Feature-Extraction in das in [13] vorgestellt Vector-Space-Model (VSM) als interne Datenrepräsentation umgewandelt. Das Vector-Space-Model transformiert Dokumente in Vektoren in einem Feature-Raum, dessen Dimension der Anzahl an eindeutigen Termen des Dokumentenkorpus entspricht. Dadurch wird ermöglicht, das Problem des Vergleichens von Text als Vergleich von algebraischen Vektoren im mehrdimensionalen Raum zu betrachten [4].

Für diesen können Funktionalitäten der in III-D beschriebenen Bibliothek von Apache Lucene eingesetzt werden. Lucene ist dabei hinsichtlich der Verwendung des Vector-Space-Models optimiert [14]. Die Reduzierung der Dimensionalität des Feature-Raumes ist zur Verbesserung der Ergebnisse der Klassifikation eine elementare Aufgabe. Anhand von reduzierten Feature-Vektoren der jeweiligen Dokumente wird ein Klassifikator trainiert. Als Klassifikator soll ein künstliches neuronales Netz eingesetzt werden.

Die Verwendung eines künstlichen neuronalen Netzes als Klassifikator verspricht im Vergleich zu anderen Verfahren eine Verbesserung der Ergebnisse [15]. Allerdings existieren in der Literatur auch Versuche wie z.B. in [16], bei denen die Performance von neuronalen Netzen als Klassifikator anderen Verfahren nachstanden. Dies gilt in den kommenden Arbeiten zu beachten.

Eine Herausforderung bei der Realisierung eines Vorschlagsystems ist, das Vertrauen der Anwender zu gewinnen. Der Benutzer soll der Überzeugung sein, dass ihm das System zuverlässig alle Dokumente von Interesse zur Verfügung stellt [17]. Es gilt hier einen geeigneten Grad an Transparenz bezüglich der Vorschläge zu realisieren.

Im folgenden Abschnitt IV-C wird zunächst der Prozess der Klassifikation von Text behandelt. Im Abschnitt IV-D wird dazu die Anwendung eines *Multilayer-Perceptrons*, also eines vorwärts gerichteten neuronalen Netzes als Klassifikator beschrieben.

### C. Feature Reduktion und Klassifikation

Das Klassifizieren von Dokumenten bedeutet weitgehend, dass ein Dokument einer oder mehreren zur Verfügung stehenden Klasse zugeordnet wird. Dazu werden Klassen vorgegeben, zu denen ein Modell durch einen überwachten Lernprozess trainiert wird. Neben neuronalen Netzen gibt es

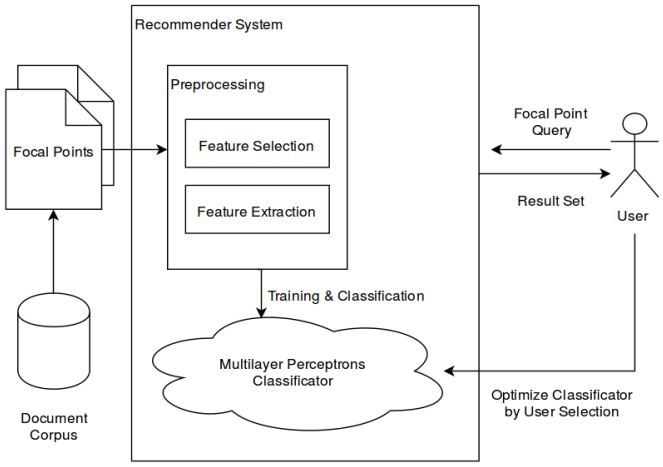


Abbildung 2. Konzept des Empfehlungssystems

verschiedene Algorithmen zur automatischen Kategorisierung von Dokumenten. Geläufig ist z.B. das *k-nearest-Neighbor*-Verfahren oder die Anwendung eines *TF-IDF-Categorizers* [14]. Diese Algorithmen verwenden das Vector-Space-Model, auf das die Funktionen von Apache Lucene optimiert sind.

Eine wichtige Aufgabe zur Klassifikation von Textdokumenten ist das Reduzieren des Feature-Raumes, welcher sich aus dem Dokumentenkorpus ergibt. Eine unzureichend implementierte Selektion von Features aus Dokumenten kann die Effizienz von guten Algorithmen nachhaltig beeinträchtigen [14]. In [12] wurden bereits gute Erfahrungen mit einer Gewichtung anhand eines TF-IDF-Maßes gemacht, über das die weniger wertvollen Feature ausgeschlossen werden können. Dabei wurde der *Bag-of-Words*-Ansatz verwendet, nachdem ein Dokument als eine Ansammlung von semantisch unabhängigen Termen begriffen wird, deren Reihenfolge keine Rolle spielt [18]. Diese wurden als Feature-Vektoren im Vector-Space-Model zusammengefasst und über das Kosinus-Ähnlichkeitsmaß miteinander verglichen. Das Ähnlichkeitsmaß eliminiert die Problematik von Feature-Vektoren unterschiedlicher Länge [12].

Anhand des TF-IDF-Maßes (Term Frequency $\times$ inverse Document Frequency) lassen sich Terme bestimmen, die in einem einzelnen Dokument des Korpus häufig vorkommen, in dem gesamten Dokumentenkorpus allerdings selten sind [19]. Diese Terme sind als Features besonders wertvoll.

Eine weitere Möglichkeit zur Verbesserung der Auswahl an Features ist die Verwendung der *n-grams*-Analyse zur Identifikation von wichtigen Wort-Kombinationen [14]. Dadurch lassen sich Kombinationen von Wörtern als einzelne Features extrahieren, wodurch die Performance von Algorithmen zur Klassifikation verbessert werden kann.

Auch lassen sich Tools wie das in [20] vorgestellte *Wordnet* oder das kommerzielle *Cogito*[6] zur Term-Expansion verwenden. Diese bieten eine hierarchische Ontologie von Sprache, die z.B. Synonymen und Hyponymen beinhaltet, anhand derer die Aussagekraft einzelner Features verbessert werden kann. Ebenso lassen sich Meta-Informationen zu Dokumenten wie

[6]Cogito: http://www.expertsystem.com/cogito/

z.B. Autor oder Quellenverweise dazu verwenden, die Gewichtung der Feature-Vektoren der Artikel zu optimieren.

Sobald der Feature-Raum hinreichend auf die wichtigsten Features reduziert ist, kann der Klassifikator trainiert werden. Hier ist es besonders wichtig, die zur Verfügung stehenden Dokumente zu den Klassen in Trainings- und Testdaten aufzuteilen. Sinnvoll ist z.B. ein 70/30-Verhältnis der zur Verfügung stehenden Trainingsdaten einer Klasse zu den reservierten Testdaten [14].

### *D. Multilayer Perceptrons zur Klassifikation*

Bei den geläufigen Algorithmen zur Klassifikation von Daten werden grundsätzlich die folgenden beiden Ansätze unterschieden [15]:

- Bei dem Instanz-basierten Ansatz werden neue Datensätze direkt unter Verwendung von Beispieldatensätzen klassifiziert.
- Nach dem Modell-Ansatz wird dagegen zunächst anhand von vorhandenen Beispieldatensätzen ein Modell erzeugt werden, welches zur Klassifikation neuer Datensätze herangezogen werden kann.

*Multilayer Perceptrons* (MLP) oder auch *Backpropagation-Netze* sind vorwärts gerichtete künstliche neuronale Netze, die als Wissensspeicher nach einem erfolgreichen Lernprozess als Modell zur Klassifikation von neuen Datensätzen verwendet werden können [15].

*1) Architektur von Multilayer Perceptrons:* Die Architektur eines künstlichen neuronalen Netzes wird nach den Vorgaben des Anwendungsfalls entwickelt. Grundlegend setzt sich ein neuronales Netz aus einer Eingangsschicht, einer oder mehreren verdeckten Zwischenschichten und einer Ausgangsschicht zusammen. Die Größe der Ausgangsschicht lässt sich zur Klassifikation über die Anzahl der vorgegebenen Klassen bestimmen [15]. Nach einem Durchlauf der Klassifikation lässt sich die Klasse des an die Eingangsschicht angelegten Datensatzes anhand des am stärksten angeregten Neurons der Ausgangsschicht ablesen. Bei einer binären Ausgangsschicht wird lediglich die Ergebnis-Klasse angeregt.

In der Theorie sind in den meisten Anwendungsfällen nicht mehr als zwei Zwischenschichten notwendig [21]. Die Anzahl der benötigten Neuronen in der Eingangsschicht ergibt sich aus der Menge an Features der Dokumente des Anwendungsfalls. Die Anzahl leitet sich direkt aus der Dimension der reduzierten Feature-Vektoren als Repräsentation der Textdokumente ab. Die Trainingsdaten müssen damit in ihrer Dimension dem Feature-Raum des Dokumentenbestandes entsprechen, damit sie zum Trainieren des Klassifikators verwendet werden können. Ohne eine Reduktion des Feature-Raumes entspricht die benötigte Anzahl der Eingangsneuronen einem Wörterbuch, welches alle bekannten Terme im Rahmen der Ontologie der verwendeten Domäne beinhaltet [22].

In einem ersten Entwurf des Netzes kann eine einzelne innere Schicht von Neuronen verwendet werden. Die Anzahl der Neuronen dieser Schicht kann zunächst auf die Hälfte der Eingangs-Neuronen beschränkt werden und im Falle von nicht zufriedenstellenden Ergebnissen bis zur gesamten Anzahl der Eingangs-Neuronen vergrößert werden [15]. Bei Bedarf kann eine zweite innere Schicht an Neuronen hinzugefügt werden. Die Gewichte der Verbindungen werden anfangs zufällig gewählt [23] und im späteren Trainingsprozess angepasst. Abbildung 3 zeigt ein vorwärts gerichtetes künstliches neuronales Netz nach der beschriebenen Architektur.

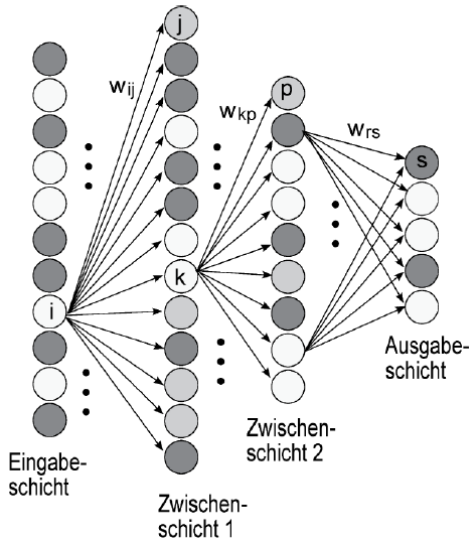


Abbildung 3. Architektur eines vorwärts gerichteten KNN [15]

*2) Training von Multilayer Perceptrons:* Das Training von Multilayer Perceptrons erfolgt unter Verwendung passender Trainingsdaten. Die Feature-Vektoren der Trainingsdaten müssen in ihrer Dimension an die Eingabeschicht des neuronalen Netzes angepasst sein [15]. Der Trainingsprozess wird i.d.R. anhand eines *Backpropagation-of-error-Algorithmus* durchgeführt [15] [23]. Dieser Prozess des *überwachten Lernens* basiert auf der Annahme, dass zu den Trainingsdaten die gewünschten Ausgangsklassen bekannt sind und daher eine Differenz des Soll- zu dem Ist-Ergebnis an den Ausgangs-Neuronen ermittelt werden kann. Der *Netzfehler* kann über die Abweichung des Wertes des Ausgabe-Neurons $o_k$ zu dem zu erwartenden Trainingswert $t_k$ bestimmt werden [15]. Die folgende Gleichung 1 zeigt die Berechnung des Netzfehlers *error*.

$$error = \sum_k (o_k - t_k) \tag{1}$$

Das gespeicherte Wissen innerhalb eines neuronalen Netzes beläuft sich auf die Gewichte einzelner Verbindungen und den Schwellenwert zur Aktivierung von Neuronen. Genau hier kann das Netz demnach auch trainiert werden. Wenn ein z.B. binäres Ausgabe-Neuron einen unerwünschten Wert hat, so können die Gewichte der Verbindungen zu diesem Neuron entsprechend erhöht oder verringert werden [21]. Die folgende Formel 2 zeigt die Berechnung der Veränderung eines Gewichts einer Verbindung $\Delta w_{ij}$ zu dem betrachteten Neuron.

$$w'_{ij} = w_{ij} + \Delta w_{ij} \text{ wobei } \Delta w_{ij} = \lambda * o_i * \delta_i \tag{2}$$

Durch den *Lernfaktor* $\lambda$ wird der Einfluss von Fehlern auf die Veränderung der Gewichte bestimmt. $\lambda$ wird in der Regel

aus dem Intervall $0,1 \leq \lambda \leq 0,8$ gewählt [15]. $o_i$ beschreibt den Ausgabewert des Vorgänger-Neurons der betrachteten Verbindung zu dem Neuron $j$ und $\delta_j$ ist das Fehlersignal, welches von der verwendeten Aktivierungsfunktion abhängig ist [15]. Wird z.B. eine logistische Aktivierungsfunktion verwendet, so berechnet sich das Fehlersignal entsprechend der Formel 3 mit der Fallunterscheidung für die Betrachtung von Ausgabe-Neuronen oder inneren Neuronen. Bei Betrachtung des Netzes von Eingangschicht zur Ausgangsschicht ist $j$ das betrachtete Neuron, $k$ ein Neuron der nachfolgenden Schicht und $i$ das Neuron der vorangehenden Schicht, dessen Verbindung mit dem Gewicht $w_{ij}$ zu dem aktuell betrachteten Neuron $j$ angepasst werden soll.

$$\delta_j = \begin{cases} o_j * (1 - o_j) * (t_j - o_j) & \text{falls } j \text{ Ausgabe-Neuron} \\ o_j * (1 - o_j) * \sum_k \delta_k * w_{jk} & \text{falls } j \text{ inneres Neuron} \end{cases} \quad (3)$$

Die Anpassung der gewichteten Verbindungen erfolgt rekursiv, ausgehend von den Neuronen der Ausgangsschicht bis hin zu den Gewichten der Verbindungen der Eingangsschicht. Dieser Prozess kann wiederholt werden, bis die Anregung der Ausgangsneuronen dem erwarteten Trainingsergebnis zur Klassifikation entspricht. Je nach Wahl des Lernfaktors $\lambda$ kann die Anzahl der benötigten Wiederholungen für ein zufriedenstellendes Ergebnis stark variieren. Auch muss das Training für jeden Datensatz der Trainingsdaten in hinreichendem Maße erfolgen, damit das neuronale Netz als Modell zur Klassifikation dienen kann.

*3) Beispiel eines ANN:* Zur Simulation von neuronalen Netzen eignet sich z.B. das Tool *JavaNNS*. Das Tool basiert auf dem *Stuttgarter Neuronale Netze Simulator (SNNS)*[24]. Abbildung 4 zeigt dazu ein initiales Netz, dessen Eingangsschicht aus insgesamt 40 Neuronen besteht. Zunächst wurde dazu eine einzelne versteckte Zwischenschicht gewählt, deren Größe die Hälfte der Eingangsschicht umfasst. Damit ließen sich Dokumente Klassifizieren, deren reduzierter Feature-Vektor in seiner Dimension der Anzahl der Neuronen der Eingangschicht entspricht.

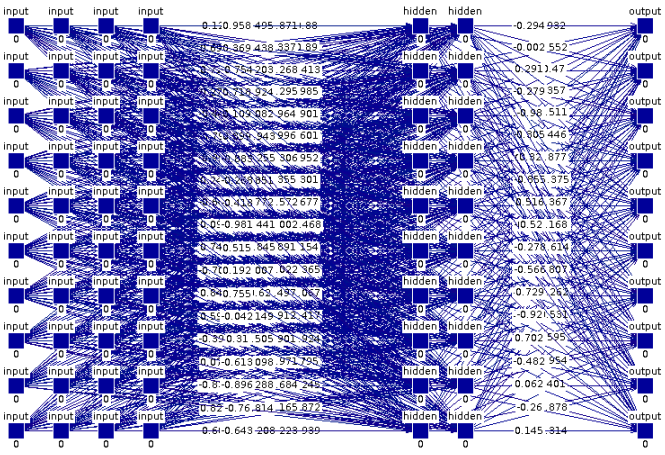


Abbildung 4. initiales ANN mittels JavaNNS

## V. Erste Erfahrungen

Im Zuge der Projekt-Veranstaltung wurden bereits erste Erfahrungen in Richtung der in IV beschriebenen Konzeption gemacht. Entgegen der hier vorgestellten Konzeption wurde allerdings ein Clustering von Textdokumenten eines Dokumentenkorpus realisiert.

Als Dokumentenkorpus wurde dabei zunächst der *Reuters-21578* Dokumentenkorpus[7] verwendet. Dieser ist frei zugänglich und wurde in der Vergangenheit zur Forschung bzgl. der Kategorisierung von Text z.B. im Rahmen der SIGIR angewendet, um die Ergebnisse verschiedener Forschungsgruppen vergleichbar zu machen. Der Vorteil des Korpus liegt dabei in seinem großen Umfang an Dokumenten. Allerdings handelt es sich dabei um relativ kurze Nachrichtentexte, welche zum Teil wenig Relevanz haben. Dies führt in Ergebnissen zu einem Gewissen Anteil an Rauschen bzw. Verfälschungen.

In den ersten Versuchen wurde Apache Solr verwendet. Apache Solr dient zwar in erster Linie der Textsuche, eignet sich aber auch in Verbindung mit z.B. dem in III-B vorgestellten Carrot2 zum Clustern von Text. Dies liegt daran, dass der Anwendungsfall der Textsuche auf grundlegende Mechanismen zur Feature-Extraktion und Feature-Selektion zurückgreift. Solr erzeugt dabei anhand von Apache Lucene zunächst einen Index des Text-Korpus. Dieser Index wurde dazu versuchsweise in Hadoops verteiltem Filesystem HDFS gespeichert, um das Zusammenspiel von Apache Solr und Apache Hadoop zu erproben. Dieser Schritt ist hinsichtlich eines Produktivsystems sinnvoll, allerdings für die hier behandelte Fragestellung zweitrangig. In Abbildung 5 ist das beschriebene Konzept schematisch dargestellt.

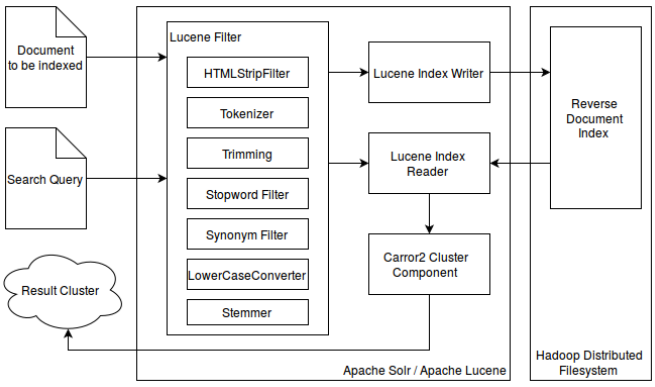


Abbildung 5. Konzept erster Versuche

Anhand von Apache Solr wurden zur Feature-Extraktion die folgenden Methoden angewendet:

- **HTML Filter:** Anhand des *HTMLStripCharFilter* können HTML-Konstrukte in den zu verarbeitenden Texten lokalisiert werden. Durch diesen Filter können demnach HTML-Tags, die sich möglicherweise zur Formatierung im Text befinden, entfernt werden. Damit werden die Ergebnisse der Verarbeitung nicht durch eventuelle HTML-Tags verfälscht. Der hier verwendete Text-Korpus beinhaltet allerdings keine HTML-Konstrukte, da die Dokumente in der heute

[7] http://www.daviddlewis.com/resources/testcollections/reuters21578/

veralteten Metasprache SGML[8] formatiert sind. Dennoch wurde der HTML-Filter angewendet, um seine Funktionalität zu testen.

- **Tokenizer:** Der zusammenhängende Text wird in einzelne Terme bzw. *Token* aufgeteilt. Dies kann z.B. anhand von Leerzeichen passieren. Das Ziel ist dabei die Trennung von zusammenhängenden Texten in atomare Teile, die anhand von Filtern normalisiert und bearbeitet werden können. Über diese atomaren Bestandteile werden Dokumente untereinander vergleichbar. Ein Token besteht dabei aus dem beinhaltenden Term und Attributen, wie z.B. die Position und ggf. ein Gewicht des Terms.
- **Trimming:** Die einzelnen Token zugehörigen Terme werden auf überflüssige voranstehende oder anhängende Leerzeichen untersucht, die daraufhin entfernt werden.
- **Stopword Filter:** Durch den Stopword-Filter werden zuvor festgelegte Stopwords identifiziert und herausgefiltert, um ein aussagekräftiges Ergebnis zu erzielen. Stopwords sind dabei semantisch von keinerlei Bedeutung und können daher problemlos entfernt werden.
- **Synonym Filter:** Synonyme können Ergebnisse verfälschen, da diese als semantisch identische Terme algorithmisch unabhängig betrachtet werden. Aus diesem Grund wird die betroffenen Terme anhand von bekannten Synonymen vereinheitlicht.
- **LowerCase Converter:** Zur Normalisierung der Terme wird Großschreibung vereinheitlicht.
- **Stemming:** Das *Stemming* beschreibt das Normalisieren der Morphologie einzelner Terme. Demnach werden Terme auf ihren grammatikalischen Stamm zurück geführt. Hierzu bietet Lucene unterschiedliche Komponenten, die Stemming für verschiedene Sprachen implementieren. Für die Englische Sprache kommt beispielsweise der Stemming-Algorithmus nach Martin Porter, welcher in [25] vorgestellt wurde.

Diese Filter werden zunächst angewendet, um die Dokumente des verwendeten Korpus in einem Index zu speichern. Wie zuvor im Abschnitt III-D1 beschrieben, werden zu den Dokumenten Felder definiert, die in den Index einfließen sollen. Der Inhalt dieser Felder wird anhand der oben genannten Filter in einen Strom aus Token gewandelt, normalisiert, gefiltert und schließlich in einem invertierten Index gespeichert. Der Lucene-Index steht danach zur Textsuche im Dokumentenkorpus zur Verfügung.

Über Apache Solr kann nun eine Query definiert werden, anhand derer über den Lucene-Index nach Ähnlichkeiten im Dokumentenkorpus gesucht wird. Dies können einzelne Terme sein oder auch ein ganzes Dokument, welches die oben beschriebenen Filter durchläuft. Aus den Termen der Query werden Token erzeugt, welche normalisiert und gefiltert werden. Danach kann über den Index von Lucene nach den aus der Query resultierenden Token gesucht werden. Anschließend werden die gefundenen Dokumente über ein Ähnlichkeitsmaß mit den Token der Query verglichen. Eine benutzerdefinierte Menge an ähnlichen Dokumenten werden durch Lucene an den Anwender als Antwort auf die Query zurück gegeben. Aus dieser Ergebnismenge können nun durch Carrot2 Dokumenten-Cluster gebildet werden. Der Anwender kann einen Leitartikel als Query-Dokument bereitstellen und erhält dazu eine Menge an ähnlichen Dokumenten, welche in Clustern gruppiert sind.

[8]Standard Generalized Markup Language


```
{
  "labels": ["OIL TAX"],
  "score": 18.90509849613169,
  "docs": [
     "EC OILS TAX NO LONGER MAJOR
      ISSUE - ASSOCIATION",

     "ITALY STANCE ON EC OILS TAX
      NOT ENCOURAGING-ASA",

     "LAWSON OIL TAX BREAKS TO HELP
      NEW FIELDS - REVENUE",

     "U.S OIL TAX WOULD NOT AFFECT
      PDVSA-CHAMPLIN DEAL",

     "HERRINGTON SAYS HE MAY RECOMMEND
      TAX BENEFITS FOR U.S. OIL INDUSTRY",

     "HERRINGTON SAYS HE MAY CALL FOR
      OIL TAX BENEFITS",
  ]
}
```


Abbildung 6. Auszug eines Ergebnis-Clusters im Reuters-21578 Korpus

Das beschriebene Verfahren liefert auf den ersten Blick nachvollziehbare Ergebnisse. Gibt der Anwender eine Menge von Termen an, werden Dokumente zurückgeliefert, die unter algorithmisch bestimmten Leitthemen in Cluster aufgeteilt sind, welche sich der Query des Benutzers zuordnen lassen. In Abbildung 6 ist ein Auszug eines beispielhaften Clusters aufgeführt. Zur Veranschaulichung werden darin die Titel der Dokumente im Cluster aufgelistet. In diesem Fall wurde die Cluster-Analyse auf eine Teilmenge von 1000 Dokumenten des Korpus angewendet. Unter dem Label *Oil Tax* sind dazu sechs News-Artikel in dem Cluster zusammengefasst, die sich anhand der jeweiligen Titel nachvollziehbar dem Leitthema des Clusters zuordnen lassen. Gut zu erkennen ist hier, dass der Leitterm des Clusters jeweils in den Titeln der Dokumente zu finden ist. Dies zeigt, wie sich die stärkere Gewichtung von Feldern der Texte (hier der Titel) auf die Ergebnisse der Verarbeitung auswirken kann.

Die tatsächliche Qualität der Ergebnisse lässt sich dagegen schlecht messen. Einerseits bietet der verwendete Dokumentenkorpus viele Artikel ohne Relevanz, die das Ergebnis verfälschen. Auf der anderen Seite fehlt das Domänen-Wissen oder auch eine vorgegebene Ergebnismengen zum Vergleich, anhand derer eine Aussage über die Qualität des angewendeten Verfahrens getroffen werden könnte. In kommenden Arbeiten soll daher auf einen anderen Text-Korpus zurück gegriffen werden, zu dem bereits beispielhafte Dossiers zur Evaluierung der Ergebnisse existieren. Auch wird anstatt einer Cluster-Analyse ein Klassifikator trainiert, wobei auf vorhandene Trainings- und Testdaten zurück gegriffen werden kann. Dadurch lassen sich Ergebnisse erzielen über deren Qualität eine Aussage getroffen werden kann.

## VI. Zusammenfassung

In dieser Ausarbeitung wurde ein Konzept eines Vorschlagsystems zur semi-automatisierten Erstellung journalistischer Dossiers vorgestellt. Anhand eines Klassifikators in Form eines künstlichen neuronalen Netzes soll zu vorgegebenen Focal-Points eine Menge an geeigneten Dokumenten identifiziert und dem Anwender vorgeschlagen werden.

In der Literatur gibt es bereits zahlreiche interessante Projekte, die in einem ähnlichen Umfeld angesiedelt sind. Anhand dieser Projekte lassen sich Erkenntnisse zu optimierten Verfahren zur Reduktion der Feature im Vector-Space-Model gewinnen. Diese Ansätze lassen sich an den Anwendungsfall dieser Arbeit anpassen und können in späteren Arbeiten implementiert werden.

Neben der Reduktion der Feature-Vektoren spielt in dem zu entwickelnden Vorschlagsystem die Implementierung und das Training eines geeigneten Klassifikators in Form eines vorwärts gerichteten neuronalen Netzes eine bedeutende Rolle. Durch das neuronale Netz wird eine deutliche Verbesserung der Ergebnisse gegenüber anderen Methoden zur Klassifikation erwartet.

Anhand erster Experimente wurde bereits die Funktionsweise von Apache Lucene erarbeitet und erste Ergebnisse erzielt. Dabei wurde ein Verfahren zur Cluster-Analyse von Dokumenten des verwendeten Korpus angewendet. Anhand des Verfahrens ließen sich nachvollziehbare, allerdings schlecht evaluierbare Ergebnisse generiert. Die so gewonnenen Erfahrungen lassen sich dennoch gut in kommenden Arbeiten nutzen.

## Literatur


[1] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and S. Trani, "Dexter 2.0-an open source tool for semantically enriching data." in International Semantic Web Conference (Posters & Demos), 2014, pp. 417–420.

[2] R. Feldman and J. Sanger, The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press, 2007. [Online]. Available: https://books.google.de/books?id=U3EA_zX3ZwEC

[3] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and S. Trani, "Dexter: an open source framework for entity linking," in Proceedings of the sixth international workshop on Exploiting semantic annotations in information retrieval. ACM, 2013, pp. 17–20.

[4] S. Osinski, J. Stefanowski, and D. Weiss, "Lingo: Search results clustering algorithm based on singular value decomposition," in Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'04 Conference held in Zakopane, Poland, May 17-20, 2004, 2004, pp. 359–368.

[5] S. Osinski and D. Weiss, "Carrot$^2$: Design of a flexible and efficient web information retrieval framework," in Advances in Web Intelligence Third International Atlantic Web IntelligenceConference, AWIC 2005, Lodz, Poland, June 6-9, 2005, Proceedings, 2005, pp. 439–444. [Online]. Available: http://dx.doi.org/10.1007/11495772_68

[6] N. Hälker, "Teilautomatisierte Erstellung von Dossiers auf Basis von Textmining-Verfahren," Arbeitspapier, HAW Hamburg, 2015.

[7] M. Brehmer, S. Ingram, J. Stray, and T. Munzner, "Overview: The design, adoption, and analysis of a visual document mining tool for investigative journalists," Visualization and Computer Graphics, IEEE Transactions on, vol. 20, no. 12, 2014, pp. 2271–2280. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2014.2346431

[8] A. Białecki, R. Muir, and G. Ingersoll, "Apache lucene 4," in SIGIR 2012 workshop on open source information retrieval, 2012, pp. 17–24.

[9] C. van Rijsbergen, Information Retrieval. 1979. Butterworth, 1979.

[10] D. Cutting and J. Pedersen, "Optimization for dynamic inverted index maintenance," in Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1989, pp. 405–411.

[11] J. Zitting, "Text and metadata extraction with apache tika," Apache Lucene, Eurocone, 2010, pp. 3–12.

[12] M. Schöneberg, "Konzepte zur semi-automatisierten Erstellung von Pressedossiers," Masterthesis, HAW Hamburg, 2015. [Online]. Available: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/schoeneberg.pdf

[13] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," Commun. ACM, vol. 18, no. 11, Nov. 1975, pp. 613–620. [Online]. Available: http://doi.acm.org/10.1145/361219.361220

[14] G. S. Ingersoll, T. S. Morton, and A. L. Farris, Taming Text: How to Find, Organize, and Manipulate It. Greenwich, CT, USA: Manning Publications Co., 2013.

[15] J. Cleve and U. Lämmel, Data Mining. De Gruyter, Oldenbourg, 2014. [Online]. Available: http://books.google.de/books?id=4i2nngEACAAJ

[16] F. Sebastiani, "Machine learning in automated text categorization," ACM computing surveys (CSUR), vol. 34, no. 1, 2002, pp. 1–47.

[17] A. Vellino and D. Zeber, "A hybrid, multi-dimensional recommender for journal articles in a scientific digital library," in Proceedings of the 2007 IEEE/WIC/ACM international conference on web intelligence and international conference on intelligent agent technology, 2007, pp. 111–114.

[18] C. D. Manning, P. Raghavan, H. Schütze et al., Introduction to information retrieval. Cambridge university press Cambridge, 2008, vol. 1.

[19] P. D. Turney, P. Pantel et al., "From frequency to meaning: Vector space models of semantics," Journal of artificial intelligence research, vol. 37, no. 1, 2010, pp. 141–188.

[20] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to wordnet: An on-line lexical database*," International journal of lexicography, vol. 3, no. 4, 1990, pp. 235–244.

[21] A. Zell, Simulation neuronaler Netze. Oldenbourg, 1997. [Online]. Available: http://books.google.de/books?id=bACTSgAACAAJ

[22] R. Navigli and P. Velardi, "Learning domain ontologies from document warehouses and dedicated web sites," Comput. Linguist., vol. 30, no. 2, Jun. 2004, pp. 151–179. [Online]. Available: http://www.dsi.uniroma1.it/~{}velardi/CL.pdf

[23] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," in Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies. Amsterdam, The Netherlands: IOS Press, 2007, pp. 3–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1566770.1566773

[24] I. Fischer, F. Hennecke, C. Bannes, and A. Zell, "Javanns–java neural network simulator: user manual–version 1.1," University of Tübingen, Wilhelm-Schickard-Institute for Computer Science, Department of Computer Architecture, 2001. [Online]. Available: http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/manual/JavaNNS-manual.pdf

[25] M. F. Porter, "An algorithm for suffix stripping," Program, vol. 14, no. 3, 1980, pp. 130–137.