HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

iSYS SOFTWARE
Individualsoftware und IT-Beratung

MASTERARBEIT
zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

Konzeption und prototypische Implementierung eines BigData-gestützten Empfehlungssystems zur Objektidentifizierung im Kontext des Forschungsprojekts MPI (Massively Adhoc Processing in the Internet)

Conceptional design and prototypic implementation of an BigData-based recommendation system for object identification in the context of the research project MPI (Massively Adhoc Processing in the Internet)

vorgelegt von

Tobias Lindner
www.tobiaslindner.eu
Matrikelnummer: 47962414

München, der 31. März 2016

Hochschule für angewandte Wissenschaften München
Fakultät für Informatik und Mathematik
Masterstudiengang Wirtschaftsinformatik

Erstprüfer: Prof. Dr. Peter Mandl
Betreuer: Dr. Nikolai Bauer

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, 31. März 2016

_____

Tobias Lindner

Inhaltsverzeichnis

EHRENWÖRTLICHE ERKLÄRUNG III

INHALTSVERZEICHNIS V

TABELLENVERZEICHNIS VIII

ABBILDUNGSVERZEICHNIS VIII

1 EINLEITUNG 1

1.1 MOTIVATION 1
1.2 ZIELSETZUNG 3
1.3 AUFBAU DER ARBEIT 4

2 STAND DER TECHNIK 5

2.1 DUPLIKATERKENNUNG 5
2.1.1 DUPLIKAT 5
2.1.2 DUPLIKATERKENNUNG BZW. OBJEKTIDENTIFIZIERUNG 6
2.1.3 STANDARDISIERUNG UND BEREINIGUNG 7
2.1.3.1 Standardisierung 7
2.1.3.2 Bereinigung 8
2.1.3.3 Anwendung 9
2.1.3.4 Verfahren für nicht zu bereinigende Daten 9
2.1.3.5 Empfehlung 9
2.1.4 ERKENNUNG UND KONSOLIDIERUNG 10
2.1.4.1 Vorverarbeitung der Daten 10
2.1.4.2 Partitionierung der Datensätze 11
2.1.4.3 Erkennung von Duplikaten 11
2.1.4.4 Konsolidierung zu einem Datensatz 12
2.2 VORVERARBEITUNG 13
2.2.1 PART-OF-SPEECH TAGGING 14
2.2.2 STEMMING 15
2.2.3 STOPPWÖRTER 15
2.2.4 TOKENIZER 15

2.3 ÄHNLICHKEITSBESTIMMUNG 16
2.3.1 ZEICHEN-BASIERT 17
2.3.1.1 Hamming-Abstand 18
2.3.1.2 Jaro und Jaro-Winkler-Distanz 18
2.3.1.3 Levenshtein und Damerau-Levenshtein-Distanz 20
2.3.1.4 Longest Common Subsequence 23
2.3.1.5 Smith-Waterman und Smith-Waterman-Gotoh-Distanz 24
2.3.2 TOKEN-BASIERT 26
2.3.2.1 Jaccard-Koeffizient 26
2.3.2.2 Kosinus-Ähnlichkeit 27
2.3.2.3 Sørensen-Dice-Koeffizient 28
2.3.2.4 n-Gramme 29
2.3.2.5 TF-IDF-Maß 30
2.3.3 HYBRID 32
2.3.3.1 Monge-Elkan-Ähnlichkeit 32
2.3.3.2 Soft-TF-IDF-Maß 32
2.3.4 PHONETISCH 33
2.3.4.1 Soundex 34
2.3.4.2 Kölner Phonetik 34
2.3.4.3 Metaphone, Double-Metaphone und Metaphone 3 36
2.3.5 DOMÄNENSPEZIFISCH 36
2.3.5.1 Datumswerte 36
2.3.5.2 Muster 37
2.3.5.3 Numerische Werte 37
2.3.5.4 Ontologien 37
2.4 EVALUATION UND QUALITÄTSMESSUNG 38

3 KONZEPTIONELLE BETRACHTUNG 40

3.1 PROBLEMANALYSE 40
3.2 ANFORDERUNGSANALYSE UND -DEFINITION 41
3.3 LÖSUNGSANSATZ 43
3.3.1 ALLGEMEINE KONZEPTION 43
3.3.2 LÖSUNGS-/UMSETZUNGSANSATZ 1 44
3.3.3 LÖSUNGS-/UMSETZUNGSANSATZ 2 45
3.3.4 LÖSUNGS-/UMSETZUNGSANSATZ 3 45
3.3.5 LÖSUNGS-/UMSETZUNGSANSATZ 4 46

4 PROTOTYPISCHE IMPLEMENTIERUNG 47

4.1 EINGESETZTE TECHNOLOGIEN 47
4.1.1 APACHE HADOOP 47
4.1.2 APACHE HBASE 50
4.1.3 SIMMETRICS 50
4.2 AUFBAU DER INFRASTRUKTUR UND CLUSTERKONFIGURATION 51
4.2.1 INFRASTRUKTUR 51
4.2.2 CLUSTERKONFIGURATION 53
4.3 ANALYSE DER AUSGANGSDATEN 54
4.4 DATENMODELL 56
4.5 IMPLEMENTIERUNG 62

5 EVALUATION UND VALIDATION 68

5.1 TESTKONZEPT 68
5.2 ERZEUGUNG DER TESTDATEN 69
5.3 ERGEBNISSE DER QUALITÄTSMESSUNG 72
5.4 ANALYSE UND BEURTEILUNG DER ERGEBNISSE 79

6 ZUSAMMENFASSUNG UND AUSBLICK 82

6.1 FAZIT 82
6.2 AUSBLICK 83

7 LITERATURVERZEICHNIS 84

8 ANHANG 91

8.1 AUFBAU UND INHALT DER DISCOGS.COM-DATEIEN 91
8.1.1 AUFBAU UND INHALT DER "ARTISTS"-DATEI VON DISCOGS.COM 91
8.1.2 AUFBAU UND INHALT DER "RELEASES"-DATEI VON DISCOGS.COM 93
8.2 QUELLCODE 96
8.2.1 QUELLCODE DER XML-ELEMENT-KLASSEN 96
8.2.2 QUELLCODE FÜR DIE DATENAUSTAUSCH-KLASSE 98
8.3 WEITERE ERGEBNISSE DER QUALITÄTSMESSUNG 100
8.3.1 F-SCORE (SORTIERT) 100
8.3.2 AUSFÜHRUNGSZEIT (SORTIERT) 102
8.3.3 ANZAHL DER BERECHNUNGEN (SORTIERT) 104

Tabellenverzeichnis

TABELLE 1: BEISPIEL FÜR IDENTISCHE UND NICHTIDENTISCHE DUPLIKATE 5
TABELLE 2: DIE BUCHSTABENCODES DES SOUNDEX-ALGORITHMUS 34
TABELLE 3: DIE BUCHSTABENCODES DER KÖLNER PHONETIK 35
TABELLE 4: KLASSIFIKATIONSMATRIX DER DUPLIKATERKENNUNG 38
TABELLE 5: DIE LEISTUNGSMERKMALE DER ZUGRUNDELIEGENDEN PHYSIKALISCHEN MASCHINEN 51
TABELLE 6: DIE LEISTUNGSMERKMALE DER VIRTUELLEN MASCHINEN 51
TABELLE 7: DIE EINGESETZTEN SOFTWARE-KOMPONENTEN 52
TABELLE 8: APACHE HADOOP KONFIGURATION 53
TABELLE 9: APACHE HBASE KONFIGURATION 53
TABELLE 10: INFORMATIONEN ZU DISCOGS 55
TABELLE 11: INFORMATIONEN ZU MUSICBRAINZ 56
TABELLE 12: PRECISION, RECALL UND F-SCORE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 72
TABELLE 13: TRUE-POSITIVES, FALSE-POSITIVES UND FALSE-NEGATIVES DER GETESTETEN ÄHNLICHKEITSMETRIKEN 72
TABELLE 14: AUSFÜHRUNGSZEITEN DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 76
TABELLE 15: ANZAHL DER BERECHNUNGEN PRO SEKUNDE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 76
TABELLE 16: PRECISION, RECALL UND F-SCORE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 100
TABELLE 17: AUSFÜHRUNGSZEITEN DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 102
TABELLE 18: ANZAHL DER BERECHNUNGEN PRO SEKUNDE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 104

Abbildungsverzeichnis

ABBILDUNG 1: VERSCHIEDENE ALGORITHMEN ZUR ÄHNLICHKEITSBESTIMMUNG VON ZEICHENKETTEN 16
ABBILDUNG 2: BEISPIEL FÜR EINE MÖGLICHE ONTOLOGIE 37
ABBILDUNG 3: ZUORDNUNG VON MUSIKNUTZUNGEN (AKTUELLES VORGEHEN) 41
ABBILDUNG 4: ZUORDNUNG VON MUSIKNUTZUNGEN (MÖGLICHES NEUES VORGEHEN) 43
ABBILDUNG 5: DIE WICHTIGSTEN KOMPONENTEN DES HADOOP-ÖKOSYSTEMS 47
ABBILDUNG 6: DIE HDFS-ARCHITEKTUR IM ÜBERBLICK 48
ABBILDUNG 7: DIE ARBEITSWEISE DES MAPREDUCE-VERFAHRENS 49
ABBILDUNG 8: EREIGNISBESCHREIBUNG DER MIGRATION EINER VIRTUELLEN MASCHINE 52
ABBILDUNG 9: DARSTELLUNG DER CLUSTER-INFRASTRUKTUR 54
ABBILDUNG 10: MEHRDIMENSIONALE DATENSTRUKTUR EINER PARTIELL SPALTENORIENTIERTEN DATENBANK 57
ABBILDUNG 11: DAS HBASE-DATENMODELL ZUR SPEICHERUNG DER RELEASES 58
ABBILDUNG 12: DAS HBASE-DATENMODELL ZUR SPEICHERUNG DER ARTISTEN 60
ABBILDUNG 13: DAS HBASE-DATENMODELL ZUR SPEICHERUNG DER BENCHMARK-DATEN 61
ABBILDUNG 14: MANIPULATION DER TESTDATEN 70
ABBILDUNG 15: F-SCORE-WERTE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 73
ABBILDUNG 16: TRUE-POSITIVES, FALSE-POSITIVES UND FALSE-NEGATIVES DER GETESTETEN ÄHNLICHKEITSMETRIKEN (1) 74
ABBILDUNG 17: TRUE-POSITIVES, FALSE-POSITIVES UND FALSE-NEGATIVES DER GETESTETEN ÄHNLICHKEITSMETRIKEN (2) 75
ABBILDUNG 18: AUSFÜHRUNGSZEITEN DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 77
ABBILDUNG 19: ANZAHL DER BERECHNUNGEN PRO SEKUNDE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (GRUPPIERT) 78
ABBILDUNG 20: F-SCORE-WERTE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 101
ABBILDUNG 21: AUSFÜHRUNGSZEITEN DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 103
ABBILDUNG 22: ANZAHL DER BERECHNUNGEN PRO SEKUNDE DER GETESTETEN ÄHNLICHKEITSMETRIKEN (SORTIERT) 105

1 Einleitung

1.1 Motivation

Die Menge an Daten und Informationen, die sowohl im wirtschaftlichen und wissenschaftlichen als auch im privaten Umfeld täglich neu entsteht und meist langfristig gespeichert wird, nimmt stetig zu. Das Marktforschungsunternehmen IDC schätzt die Menge der im Jahr 2012 weltweit neu erzeugten Daten auf über 2,8 Zettabyte (entspricht ca. drei Milliarden Terabyte) und prognostiziert für das Jahr 2020 neu erzeugte Datenmengen in einer Größenordnung von mehr als 40 Zettabyte pro Jahr [IDC].

Dies führt immer häufiger dazu, dass nicht nur die Speicherung, Verarbeitung und Analyse, sondern auch das Zusammenführen von Daten aus verschiedenen Quellen und die Sicherung der Datenqualität (also das Bereinigen von redundanten, inkonsistenten und falsch formatierten Datensätzen) zu einem wichtigen Bestandteil vieler heutiger IT-Projekte wird [Bet08], [SBR09], [Sch10]. Die Aufgaben sind hierbei so umfangreich wie vielfältig und reichen vom Korrigieren von Eingabefehlern über das automatische Erkennen und Umformatieren von verschiedenen Schreibweisen und Abkürzungen bis hin zum eindeutigen Auffinden von Suchobjekten in einer vorhandenen Objektmenge oder dem Abgleichen und Zusammenführen von mehreren bestehenden Datenbeständen.

Die dabei angewandten Verfahren, die meist automatisch ausgeführt werden, werden als Objektidentifizierung oder Duplikaterkennung bezeichnet und arbeiten meist nach dem Prinzip des paarweisen Vergleichs von Datensätzen. Häufig müssen beim Vergleich mehrere Attribute der Datensätze mit einbezogen werden, um aussagekräftige bzw. sinnvolle Ergebnisse zu erzielen. Das Ergebnis des Vergleichs beschreibt die Ähnlichkeit des Datensatzpaars anhand einer (reellen) Zahl. Mit Hilfe von verschiedenen, individuell festzulegenden Entscheidungsverfahren wird letztendlich ermittelt, ob die Ähnlichkeit ausreichend groß ist, um das Datensatzpaar als Duplikat zu bezeichnen. Dabei bestehen zwei große Herausforderungen in der Entwicklung von Methoden zur automatischen Duplikaterkennung. Zum einen muss eine möglichst hohe Durchsatzrate erzielt werden, um die meist große Menge an zu vergleichenden Datensätzen möglichst schnell abarbeiten zu können. Zum anderen müssen die Ergebnisse der Duplikaterkennung qualitativ hochwertig sein, um möglichst viele der vorhandenen Duplikate als solche zu erkennen bzw. die Anzahl der fälschlich erkannten Duplikate möglichst gering zu halten.

Auch die Musikbranche steht, vor allem durch die heutzutage stark steigende digitale Musiknutzung, immer häufiger vor der Herausforderung, Suchobjekte in einer großen Objektmenge eindeutig aufzufinden sowie getrennt geführte Datenbestände zusammenzuführen. Ein Beispiel ist die Durchsetzung des Urheberrechts in der Musikindustrie: Musikstücke sind persönliche Schöpfungen eines Künstlers und deren Nutzung durch Dritte soll gemäß dem Urheberrecht angemessen vergütet werden. Die eindeutige Erkennung von Duplikaten ist in diesem Bereich vor allem deshalb relevant, da der Prozess der Rechteermittlung aller Urheber sehr aufwändig ist und dieser Aufwand durch die korrekte Ermittlung bereits bekannter Werke stark reduziert wird.

Die Wahrung der eben beschriebenen Nutzungsrechte - durch die Erhebung und anschließende Verteilung von Nutzungsgebühren an die Urheber - erfolgt durch die zuständigen Verwertungsgesellschaften. In Deutschland ist dies beispielsweise die *Gesellschaft für musikalische Aufführungs- und mechanische Vervielfältigungsrechte (GEMA)*. Die hierbei anfallende Datenmenge, die regelmäßig an die GEMA übermittelt und von dieser anschließend verarbeitet wird, beläuft sich bereits auf mehrere 100 Gigabyte bin hin zu vielen Terabyte - pro Monat [MPI].

Daher befasst sich das Forschungsprojekt *Massively Adhoc Processing in the Internet (MPI)* des *Competence Center Wirtschaftsinformatik* der *Hochschule München* in Zusammenarbeit mit der Firma *iSYS Software GmbH* mit der Erforschung und Entwicklung von effizienten Prozessen und Softwarearchitekturen zur Verarbeitung dieser an die GEMA gelieferten Massendaten. Diese Abschlussarbeit ist, neben weiteren Bachelor- und Masterarbeiten, in das Forschungsprojekt integriert und befasst sich konkret mit der Konzeption und anschließenden prototypischen Implementierung eines BigData-gestützten Empfehlungssystems zur urheberrechtlichen Zuordnung von Musiknutzungen.

1.2 Zielsetzung

Das Ziel dieser Abschlussarbeit ist die Konzeption und prototypische Implementierung eines BigData-gestützten Empfehlungssystems zur urheberrechtlichen Zuordnung von Musiknutzungen. Hierbei wird in der Konzeptionsphase die derzeit in der Entwicklung befindliche Produkt-Datenbank des Master Data Management (MDM) der GEMA zugrunde gelegt. Zudem werden für die theoretische Skizzierung von verschiedenen möglichen Lösungsansätzen sowie für die Konzeption und anschließende prototypische Implementierung des Empfehlungssystems die allgemeinen Regeln zur Bestimmung von Produktdatendubletten für Basisprodukte aus der Dokumentation des MDM als Orientierung verwendet. Aufgrund des Umfangs dieser Regeln und der begrenzten Bearbeitungszeit der Abschlussarbeit können jedoch nicht alle Regeln vollständig umgesetzt bzw. implementiert werden. Für die Qualitätsmessung des Systems bzw. der einzelnen implementierten Algorithmen zur Ähnlichkeitsbestimmung wird zunächst ein Testdatenbestand erzeugt. Dieser besteht aus einer zufälligen Auswahl an Datensätzen aus dem Referenzdatenbestand, der anschließend zufällig, mit zuvor festgelegten Eigenschaften, verändert wird. Für die Ermittlung der Qualität werden die zwei, im Bereich der Duplikaterkennung populären Metriken Precision und Recall sowie die Kombination aus beiden, das sogenannte F-Maß, verwendet. Darüber hinaus wird die Ausführungsgeschwindigkeit der Algorithmen gemessen und verglichen. Als Datenquelle für die prototypische Implementierung sowie die abschließende Qualitätsmessung werden die Daten der öffentlich zugänglichen und gemeinfreien Online-Datenbanken für Musik-Metadaten *Discogs* [Dis15] und/oder *MusicBrainz* [MuB15] verwendet.

Neben Potenzialen und Herausforderungen bei der Implementierung eines BigData-gestützten Empfehlungssystems zur urheberrechtlichen Zuordnung von Musiknutzungen sollen auch die Leistungsfähigkeit (sowohl die Ausführungsgeschwindigkeiten als auch die Treffergenauigkeiten) sowie mögliche Grenzen von ausgewählten Algorithmen zur Ähnlichkeitsbestimmung aufgezeigt werden. Das Empfehlungssystem soll dabei möglichst offen gestaltet werden, um spätere Erweiterungen (z.B. weitere Duplikaterkennungs-Algorithmen) einfach zu halten.

1.3 Aufbau der Arbeit

Im Grundlagenteil wird zunächst eine Literaturrecherche durchgeführt. Hierzu wird eine kurze Einführung zum Thema Objektidentifizierung bzw. Duplikaterkennung gegeben und anschließend der aktuelle Stand der Technik in diesem Bereich erarbeitet. Hierbei wird vor allem auf die verschiedenen Möglichkeiten bzw. Metriken der Ähnlichkeitsbestimmung sowie auf mögliche Varianten der Erfolgs- und Qualitätsmessung eingegangen. Zudem werden einige Schritte bei der Vorverarbeitung von Daten - beispielsweise Stemming, Stoppwörter und Tokenizer - näher erläutert.

Die Literaturrecherche dient als Grundlage für die Findung und anschließende Auswahl von geeigneten Techniken und Methoden für die Lösung der konkreten Problemstellung.

In der konzeptionellen Betrachtung wird zunächst die konkrete Problemstellung analysiert. Anschließend werden die Anforderungen definiert sowie mögliche Lösungsansätze skizziert.

Nach einer kurzen Beschreibung der bei der prototypischen Implementierung zum Einsatz kommenden Technologien erfolgt eine detaillierte Darstellung der Infrastruktur und Konfiguration des vom *Forschungsprojekt MPI* der *Hochschule München* bereitgestellten Hadoop-Clusters. Anschließend werden die Ausgangsdaten untersucht und die Entwicklung eines geeigneten Datenmodells sowie die prototypische Implementierung des Systems ausführlich erörtert. Zudem wird das System bzw. die einzelnen implementierten Algorithmen zur Ähnlichkeitsbestimmung validiert. Hierzu wird ein Testkonzept entwickelt. Dieses umfasst vor allem die Generierung von sinnvollen Testdaten. Die bei der Messung erzielten Ergebnisse (Ausführungsgeschwindigkeiten und Treffergenauigkeiten der einzelnen Algorithmen) werden übersichtlich dargestellt sowie analysiert und beurteilt.

Abschließend werden die Ergebnisse der Masterarbeit kurz zusammengefasst und es wird ein Ausblick gegeben.

2 Stand der Technik

2.1 Duplikaterkennung

2.1.1 Duplikat

Als Duplikate oder Dubletten werden unterschiedliche Datensätze bezeichnet, die in der realen Welt dasselbe, eindeutige Objekt repräsentieren [NaH10]. Die doppelte Aufnahme von Adressen aufgrund von unterschiedlichen Schreibweisen bei der Eingabe, mehrfach geführte Lieferanten oder Kunden in einem Managementsystem oder verschiedene Repräsentationen eines Produkts oder Materials in einem ERP-System sind einige, ausgewählte Beispiele hierfür.

ID	Vorname	Nachname	E-Mail Adresse	Telefonnummer
1	John	Smith	mail@johnsmith.com	089/123456
2	John	Smith	mail@johnsmith.com	089/123456
3	Jon	Smith	mail@johnsmith.com	+49 (89) 123456

Tabelle 1: Beispiel für identische und nichtidentische Duplikate

Wichtig ist die Unterscheidung zwischen zwei Arten von Duplikaten [ABEM10], [NaH10]:

- ***Identische Duplikate:*** Stimmen bei zwei oder mehreren Datensätzen die Werte aller Attribute (außer ggf. dem Primärschlüssel) überein, so werden diese Datensätze als identische Duplikate bezeichnet. In der obigen Tabelle sind dies die Datensätze mit der ID 1 und der ID 2. Das Erkennen von identischen Duplikaten ist trivial und lässt sich beispielsweise durch einen direkten Vergleich aller Attribute leicht umsetzen. Eine aufwändige Bereinigung und Konsolidierung der betroffenen Datensätze ist nicht notwendig, da die überzähligen Datensätze ohne Informationsverlust einfach gelöscht werden können.
- ***Nichtidentische Duplikate:*** Bei nichtidentischen (unscharfen) Duplikaten handelt es sich um Datensätze, bei denen mindestens ein, jedoch auch mehrere Attributwerte voneinander abweichen. Dies wird in der obigen Tabelle beispielhaft mit dem Datensatz mit der ID 3 visualisiert. Dieser repräsentiert in der Real-Welt offenbar dasselbe Objekt wie die Datensätze mit der ID 1 und der ID 2, unterscheidet sich jedoch in den Attribut "Vorname" und "Telefonnummer" und ist somit nicht identisch mit diesen. Der Erkennungs- und Bereinigungsprozess ist in diesen Fällen weitaus komplexer, da kein direkter Vergleich über alle Attribute möglich ist. Darüber hinaus ist nach dem Identifizieren der betroffenen Datensätze eine aufwändige Konsolidierung notwendig, da ein einfaches Löschen der betroffenen Datensätze einen Informationsverlust zur Folge haben könnte.

Das Erkennen von nichtidentischen Duplikaten bildet hierbei den Schwerpunkt dieser Arbeit.

2.1.2 Duplikaterkennung bzw. Objektidentifizierung

Die Duplikaterkennung bzw. Objektidentifizierung kann in den (Forschungs-) Bereich Informations- und Datenqualität eingegliedert werden und enthält bzw. beschreibt Maßnahmen zur Verbesserung dieser. Die ersten wissenschaftlichen Publikationen hierzu wurden bereits 1959 [NKAJ59] und 1969 [FeS69] veröffentlicht. [Jur51] beschreibt Datenqualität als "fitness-for-use", also "für einen bestimmten Gebrauch geeignet" und [WaS96] hat die Betrachtung der Qualität von Informationen in nachfolgende vier Kategorien und 15 Dimensionen untergliedert:

- Accessibility: Accessibility, Access security
- Contextual: Appropriate amount of data, Completeness, Relevancy, Timeliness, Value-added
- Intrinsic: Accuracy, Believability, Objectivity, Reputation
- Representational: Concise representation, Ease of understanding, Interpretability, Representational consistency

Auf Basis der oben genannten Kategorien und Dimensionen hat die *Deutsche Gesellschaft für Informations- und Datenqualität (DGIQ)* im Jahr 2007 eine Übersetzung vorgeschlagen und empfiehlt, diese im deutschsprachigen Raum einheitlich zu verwenden [DGI07]:

- System (systemunterstützt): Bearbeitbarkeit, Zugänglichkeit
- Nutzung (zweckabhängig): Angemessener Umfang, Vollständigkeit, Relevanz, Aktualität, Wertschöpfung
- Inhalt (inhärent): Fehlerfreiheit, Glaubwürdigkeit, Objektivität, hohes Ansehen
- Darstellung (darstellungsbezogen): einheitliche Darstellung, Verständlichkeit, Übersichtlichkeit, eindeutige Auslegbarkeit

Die Maßnahmen und Methoden der Duplikaterkennung verbessert dabei zwei Dimensionen direkt: die *Fehlerfreiheit* und die *Vollständigkeit* [NaH10]. Als Fehlerfreiheit wird der "Umfang, in dem Daten korrekt, zuverlässig und fehlerfrei sind" [NaH10] beschrieben und gibt das Verhältnis von fehlerhaften Datensätzen zur Gesamtzahl der Datensätze an. Durch die Aufdeckung und Eliminierung von redundanten Datensätzen während der Duplikaterkennung wird die Fehlerfreiheit verbessert. Die Vollständigkeit gibt an, ob die Attributwerte eines Datensatzes das repräsentierende Objekt in der realen Welt vollständig abbilden. Da die gefundenen Duplikate vor der Eliminierung konsolidiert werden, werden ggf. unterschiedliche Informationen in den Attributwerten zu einem Datensatz zusammengefasst und dadurch die Vollständigkeit verbessert. Darüber hinaus werden die meisten anderen Dimensionen durch die Anwendung der Duplikaterkennung indirekt verbessert.

2.1.3 Standardisierung und Bereinigung

Um die Daten- und Informationsqualität in Datenbanken und anderen Informationssystemen zu verbessern, müssen die enthaltenen Datenfehler (inkonsistente, redundante, veraltete oder falsch formatierte Daten) entfernt oder korrigiert werden. Dies wird als Datenbereinigung bezeichnet und kann sowohl automatisiert als auch manuell durchgeführt werden. Der Prozess zur Bereinigung der Daten besteht nach [ABEM10, S.159ff] aus zwei aufeinanderfolgenden Schritten sowie verschiedenen Verfahren, Methoden und Hilfsmitteln. Diese sind:

1. Standardisierung der Daten
    - Strukturieren
    - Normieren
2. Bereinigung der Daten
    - Entfernen
    - Ersetzen
    - Ableiten
    - Defaults verwenden
    - Duplikate entfernen
    - Auftrennen

2.1.3.1 Standardisierung

Um Daten besser bereinigen zu können, sollten sie zuvor standardisiert werden. Um dies zu realisieren, erfolgt zunächst eine Strukturierung und anschließend eine Normierung [ABEM10].

Strukturieren. Das Ziel der Strukturierung ist eine einheitliche Formatierung der Daten. So werden beispielsweise alle Datumsangaben in das Formatformat TT.MM.JJJJ (z.B. 01.01.2000) und alle Uhrzeitangaben in das Format HH24:MM:SS (13:10:54) umstrukturiert. Gegebenenfalls werden zusammengesetzte Daten in einzelne Bestandteile aufgeteilt. Zum Beispiel werden alle Telefon- und Faxnummern in die Ländervorwahl, Ortsvorwahl und Rufnummer oder alle Kundennamen in die Bestandteile Anrede, Titel, Vorname und Nachname zerlegt. Da die Daten häufig in einer Vielzahl von unterschiedlichen Formaten vorliegen, ist dies jedoch meist nicht trivial. Daher werden häufig spezifische Referenzdatenbanken oder komplexe Softwarelösungen von Drittanbietern verwendet.

Normieren. Das Ziel der Normierung ist die Abbildung von verschiedenen Werten der einzelnen Attribute auf vorgegebene Wertelisten. Dies findet u.a. häufig bei Firmenzusätzen sowie der Anrede oder dem akademischen Titel Anwendung. Zum Einsatz kommen externe Referenzdatenbanken (wie z.B. Länderkürzel nach ISO 3166 oder Sprachcodes nach ISO 639) sowie selbst erstellte Wertelisten.

2.1.3.2 Bereinigung

Nach der Standardisierung erfolgt die Bereinigung der fehlerhaften Daten. Hierfür stehen laut [ABEM10, S.162f] sechs Methoden zur Auswahl, die sowohl einzeln als auch kombiniert angewendet werden können.

Entfernen. Bei der Entfernung werden fehlerhafte Daten herausgefiltert und nicht weiterverarbeitet. Dies wird meist angewendet, wenn einzelne fehlerhafte Datensätze nicht korrigiert werden können, die übrigen, fehlerfreien Datensätze allerdings weiter verarbeitet werden müssen. Ein Beispiel ist das Entfernen von Artikel-Stammsätzen, die keine Artikelnummer und Artikelbezeichnung besitzen.

Ersetzen. Bei der Ersetzung werden fehlerhafte Daten durch Daten aus anderen Quellen, wie beispielsweise internen oder externen Referenzdatenbeständen, ersetzt. Ein Beispiel aus der Praxis ist das Ermitteln und anschließende Einfügen einer fehlenden Kundennummer auf einer Bestellung; die Ermittlung der Kundennummer erfolgt dabei z.B. durch eine Suche nach dem Namen und der Adresse des Kunden in der Kunden-Stammdatentabelle.

Ableiten. Fehlerhafte oder fehlende Daten können durch Ableitung aus anderen Daten korrigiert bzw. ergänzt werden. Zum Beispiel lässt sich der Gesamtpreis einer Bestellung mit Hilfe der Einzelpreise und Bestellmengen der einzelnen Positionen berechnen oder die fehlende Anrede einer Person aus dem Vornamen ableiten.

Defaults verwenden. In manchen Fällen ist es möglich, fehlende Werte durch im Vorfeld sinnvoll definierte Standardwerte zu ersetzen.

Duplikate entfernen. Die Entfernung von Duplikaten ist eine weitere Methode der Datenbereinigung. Zu beachten ist jedoch, dass die redundanten Daten vor dem Entfernen zu einem einzigen Datensatz konsolidiert werden, da andernfalls ein meist unerwünschter Daten- und Informationsverlust nicht auszuschließen ist.

Auftrennen. Dies ist das Gegenteil der Methode "Duplikate entfernen". Hier werden fehlerhaft zusammengefasste Daten wieder getrennt.

Um den Bereinigungsprozess nachvollziehbar und revisionssicher zu gestalten, sollten die originären, fehlerhaften Daten auf keinen Fall einfach mit den neuen, korrigierten Daten überschrieben bzw. gelöscht werden. [ABEM10, S.163ff] nennt folgende Alternativen: Müssen nur wenige Spalten bzw. Datensätze korrigiert werden, bietet es sich an, die korrigierten Werte in zusätzliche Spalten bzw. Zeilen zu speichern. Muss hingegen eine große Anzahl an Spalten bzw. Datensätzen korrigiert werden, empfiehlt sich die Speicherung in einer separaten Tabelle.

2.1.3.3 Anwendung

[ABEM10, S.183] nennt vier Stellen innerhalb des ETL-Prozesses, an denen die Daten grundsätzlich standardisiert und bereinigt werden können. Diese werden nachfolgend kurz erörtert:

An der Quelle: Das Standardisieren und Bereinigen an der Quelle reduziert die Gefahr, fehlerhafte Daten in ein Data Warehouse (DWH) einzuspielen. Eine separate Validierung von Daten innerhalb des DWH wird allerdings dennoch empfohlen, um Änderungen an den Daten und somit an der Datenqualität frühzeitig zu erkennen und ggf. rechtzeitig Gegenmaßnahmen einleiten zu können.

Bei der Transformation: Dies ist der Normalfall. Da bei der Transformation die Daten homogenisiert und integriert werden, sind für die anschließende Bereinigung im Allgemeinen alle notwendigen Daten vorhanden.

Beim Laden: Stehen an der Quelle oder bei der Transformation nicht alle benötigten Daten für die Standardisierung und Bereinigung zur Verfügung, können diese Vorgänge erst beim Laden in das Ziel erfolgen.

In den Zielen: Vereinzelt müssen Daten direkt in den Zielen standardisiert und bereinigt werden. Zum Beispiel wenn fehlerhafte Daten eingespielt wurden oder wenn, aufgrund von fehlenden Daten, eine Bereinigung in den vorhergehenden Schritten nicht möglich war. Dies wird in der Praxis jedoch selten durchgeführt, da es beim Reporting und bei Analysen zu Fehlern und Widersprüchen kommen kann, falls der Zugriff auf die fehlerhaften Daten nicht gesperrt wurde.

[ABEM10, S.183] empfiehlt folgenden bewährten Grundsatz für die Praxis: "Die Bereinigung sollte direkt im Anschluss an die Validierung durchgeführt werden. [...] Dadurch existiert eine starke Verbindung und Nähe zwischen diesen beiden Teilen. Bei Änderungen in der Validierung [...] ist es damit wahrscheinlicher, dass auch die Bereinigung entsprechend geändert wird."

2.1.3.4 Verfahren für nicht zu bereinigende Daten

Trotz meist guter - sowohl automatisierter als ggf. auch anschließender manueller - Validierung, Standardisierung und Bereinigung verbleiben in der Praxis Fehler im Datenbestand. Das Ziel muss es sein, diesen Anteil an Fehlern "so gering wie ***nötig*** zu halten - ***nicht*** so gering wie ***möglich***" [ABEM10], da der Aufwand ab einem gewissen Punkt extrem hoch wird.

2.1.3.5 Empfehlung

Da die Standardisierung und Bereinigung von Daten sehr komplex ist, sollten die hierfür notwendigen Programme nicht selbst entwickelt, sondern vorhandene Werkzeuge eingesetzt werden. Darüber hinaus sollte die Bereinigung so früh wie möglich, am besten an der Quelle, erfolgen [ABEM10].

2.1.4 Erkennung und Konsolidierung

Alle Methoden, die zur Erkennung von Duplikaten eingesetzt werden, sollten laut [ABEM10, S.166] nachfolgende zwei Ziele verfolgen: *Effizienz* und *Effektivität*. Da zum einen häufig sehr große Datenmenge verarbeitet werden müssen, ist es notwendig, die Anzahl der verwendeten Vergleichsoperationen möglichst klein und dadurch die Laufzeit möglichst niedrig zu halten. Zum anderen müssen gleichzeitig hoch qualitative Ergebnisse erzielt werden, um die Menge an richtig gefundenen Duplikaten möglichst hoch bzw. die Anzahl an fälschlich erkannten Duplikaten möglichst niedrig zu halten.

Um den beiden geforderten Ziele *Effizient* und *Effektivität* gerecht zu werden, sollte der Prozess zur Erkennung und anschließenden Konsolidierung von Duplikaten laut [ABEM10, S.167ff] nachfolgende vier Schritt umfassen:

1. Vorverarbeitung der Daten
2. Partitionierung der Datensätze
3. Erkennung von Duplikaten
4. Konsolidierung zu einem Datensatz

2.1.4.1 Vorverarbeitung der Daten

Der erste Schritt des Erkennungs- und Konsolidierungsprozesses, die Vorverarbeitung der Daten, ist die Basis des gesamten Prozesses. Die Ausführung bzw. die Ergebnisse dieses ersten Schrittes entscheiden über den Erfolg der nächsten Schritte. Nachfolgend wird, basierend auf [ABEM10, S.168], ein mögliches Vorgehen zur Vorverarbeitung exemplarisch aufgezeigt. Wichtig ist, dass die vorverarbeiteten Werte in zusätzlichen Attributen gespeichert und die Originalwerte nicht überschrieben werden.

Zunächst werden die Werte der Attribute, in denen textuelle Inhalte hinterlegt sind, in eine einheitliche Groß- oder Kleinschreibung umgewandelt. Darüber hinaus werden Sonderzeichen, Leerzeichen zwischen den einzelnen Wörtern sowie mit Hilfe von sogenannten Stoppwortlisten die Füllwörter entfernt. Zudem werden sprachspezifische Zeichen, wie beispielsweise Umlaute der deutschen Sprache oder diakritische Zeichen (z.B. Akzente, usw.), ersetzt. Außerdem werden alle oder einige der im Kapitel 2.1.3 erläuterten Verfahren und Methoden zur Standardisierung und Bereinigung angewendet. Aus *Hochschule für angewandte Wissenschaften München* wird beispielsweise *hochschuleangewandtewissenschaftenmuenchen*.

Einige Bestandteile der Vorverarbeitung, zum Beispiel die Stoppwortlisten, werden im Kapitel 2.2 nochmals ausführlicher beschrieben.

2.1.4.2 Partitionierung der Datensätze

Um Duplikate zu finden, ist es notwendig, innerhalb eines Datenbestands jeden Datensatz (n) mit jedem anderen Datensatz ($n-1$) zu vergleichen. Unter der Annahme, dass das Ähnlichkeitsmaß symmetrisch ist - also $sim(c, c') = sim(c', c)$ gilt - führt dies zu $\frac{\mathrm{n} \times (\mathrm{n}-1)}{2}$ Vergleichsoperationen [NaH10, S.16]. Bei einer Million Datensätzen innerhalb eines Datenbestands sind dies beispielsweise fast eine Billion durchzuführende Vergleichsoperationen. Da es bei der Durchführung einer Duplikaterkennung meist nicht ausreichend ist, nur ein Attribut jedes Datensatzes zu vergleichen, steigt die Anzahl der notwendigen Operationen proportional zur Anzahl der zu vergleichenden Attribute. Um die Forderung nach Effizienz zu erfüllen, ist es sinnvoll, die Anzahl der Vergleichsoperationen und somit auch die Laufzeit bzw. Rechenleistung des Erkennungs- und Konsolidierungsprozesses zu minimieren. Dies ist durch eine Partitionierung der Datensätze, also die Einschränkung des Vergleichsraums, möglich. Die Partitionierung erfolgt dabei nach einem oder mehreren Attributen (auch Partitionierungsschlüssel genannt), wobei der Partitionierungsschlüssel so gewählt wird, dass sich alle potentiellen Duplikate innerhalb dieser Partition befinden. Denkbare Partitionen sind bei einer (internationalen) Adressdatenbank beispielsweise Kontinente, Länder, Regionen oder die ersten ein bis zwei Ziffern der Postleitzahl. Da dieses Vorgehen jedoch nicht sehr fehlertolerant ist, ist die optimale Wahl der Attribute erfolgsentscheidend. Laut [ABEM10, S.169] ist es in der Praxis "[...] oft schwierig bis unmöglich, die Schlüssel und deren Einteilung festzulegen". Zudem kann es vorkommen, dass aufgrund der Partitionierung Duplikate nicht entdeckt werden.

2.1.4.3 Erkennung von Duplikaten

Im dritten Schritt, der Erkennung von Duplikaten, erfolgt zunächst ein paarweiser Vergleich der Datensätze und anschließend eine Entscheidung, ob es sich um Duplikate handelt. Hierbei geht der Prozess auf ein im Jahr 1969 von Fellegi und Sunter entworfenes abstraktes Modell [FeS69] zurück:

Es existieren zwei Datenmengen A und B. Diese können zu einer gemeinsamen Menge aus geordneten Paaren

$$A \times B \ = \ \{(a, b); a \in A, b \in B\}$$

zusammengefasst werden. Diese enthält die Teilmenge der Duplikate

$$M \ = \ \{(a, b); a = b, a \in A, b \in B\}$$

und der Teilmenge der Nicht-Duplikate

$$U \ = \ \{(a, b); a \neq b, a \in A, b \in B\}$$

Für die Erkennung von Duplikaten wird zunächst ein Vergleichsraum mit im Vorfeld festgelegten Kriterien definiert. Anschließend werden für jedes Tupel (a, b) diese Kriterien mit einer im Voraus gewählten Vergleichsfunktion $sim(a, b)$ bewertet. Die Ähnlichkeit zweier Datensätze ergibt sich aus der Ähnlichkeit der einzelnen Attribute, wobei diese individuell gewichtet werden können. Eine ebenfalls im Voraus gewählte Entscheidungsfunktion $dec(sim)$ wertet die Ergebnisse der Vergleichsfunktion aus und klassifiziert die Tupel anhand eines Schwellwerts t als Duplikat M oder als Nicht-Duplikat U:

$$dec(sim) = \begin{cases} M & falls\ sim(a,b) > t \\ U & ansonsten \end{cases}$$

Um Datensätze, die sich in einer Grauzone zwischen Duplikaten und Nicht-Duplikaten befinden, eindeutig identifizieren und in einer Nachverarbeitung genauer untersuchen zu können, wird hierfür in der Praxis häufig eine dritte Menge MU sowie ein zweiter Schwellenwert eingeführt [ABEM10, S.173]:

$$dec(sim) = \begin{cases} M & falls\ sim(a,b) > t_1 \\ U & falls\ sim(a,b) < t_2 \\ MU & falls\ t_2 \geq sim(a,b) \leq t_1 \end{cases}$$

Um die nachfolgenden Prozesse spezifisch steuern zu können, wird neben der qualitativen Zuordnung zu den Mengen *M*, *U* oder *MU* auch das quantitative, normierte Ergebnis der Vergleichsfunktion (0 für Nicht-Duplikate bis 1 für Duplikate) gespeichert. Hierdurch wird es u.a. möglich, die Datensätze in einer auf- oder absteigend Sortierung auszugeben und so beispielsweise eine manuelle Prüfung der Ergebnisse zu erleichtern.

Die Wahl des Schwellenwerts beeinflusst dabei die Genauigkeit der Duplikaterkennung. Um diese zu testen und zu bewerten, können verschiedene Kennzahlen wie beispielsweise die Sensitivität oder die Spezifität ermittelt werden. Weitere Informationen hierzu sind im Kapitel 2.4 zu finden.

2.1.4.4 Konsolidierung zu einem Datensatz

Nachdem die Duplikate erfolgreich erkannt wurden, müssen diese konsolidiert werden. Eine einfache Eliminierung aller überschüssigen Datensätze ist nur bei vollständig identischen Duplikaten möglich. Andernfalls müssen die Informationen der Duplikate zu einem Datensatz zusammengeführt werden, um den Verlust von Informationen zu vermeiden. Hierbei wird nach [ABEM10] zwischen drei verschiedenen Fällen unterschieden: Die Duplikate besitzen die gleichen Attribute und Attributwerte; Duplikate besitzen unterschiedliche Anzahl von Attributen und mind. ein Attribut unterscheidet sich; mind. ein Attribut der Duplikate ist gleich, aber dessen Attributwerte unterscheiden sich.

Kommt es aufgrund unterschiedlicher Attributwerte bei der Zusammenführung zu einem Konflikt, sind laut [ABEM10] zwei Verfahrenswege möglich:

Konfliktvermeidende Verfahren. Bei diesen Verfahren wird ein Konflikt vermieden, indem die Lösung des Konflikts auf einen nachfolgenden Auswertungsprozess verschoben wird. Dies steigert jedoch die Komplexität sowie Fehleranfälligkeit. Die Konflikt-verursachenden Attributwerte werden hierbei mit einem Kennzeichen markiert (Maskierung) und mengenbasiert zusammengeführt, d.h. die Werte werden nicht mehr als Einzelwerte, sondern als Wertemenge im Attribut gespeichert.

Konfliktlösende Verfahren. Bei diesen Verfahren wird ein Konflikt sofort gelöst. Zum Beispiel kann aus der Menge an Konflikt-verursachenden Werten ein einziger Wert (z.B. der am häufigsten auftretende) als der korrekte ausgewählt werden (Selektionsverfahren). Auch die Berechnung eines neuen Wertes (z.B. Durchschnitt, Minimum, Maximum, etc.) ist möglich (Aggregatverfahren). Zudem kann für die Auswahl des korrekten Wertes auch die Vertrauenswürdigkeit des Wertes oder Datensatzes herangezogen werden (konfidenzbasierte Verfahren). Auch die Kombination von verschiedenen Verfahren ist möglich. Welche Verfahren letztendlich eingesetzt werden hängt dabei sehr stark von den konkreten Anforderungen ab.

Da eine detaillierte Beschreibung der einzelnen Verfahren den Rahmen dieser Abschlussarbeit bei weitem sprengen würde, wird an dieser Stelle auf die einschlägige Literatur zu diesem Thema verwiesen.

2.2 Vorverarbeitung

Die Vorverarbeitung der Daten als Bestandteil eines effektiven und effizienten Erkennungs- und Konsolidierungsprozesses wurde bereits im Kapitel 2.1.4.1 kurz dargestellt. Da die Vorverarbeitung der Daten die Basis des Prozesses bildet und maßgebend für den Erfolg der Erkennung und Konsolidierung ist, wird an dieser Stelle erneut und wesentlich detaillierter darauf eingegangen.

Im Bereich der allgemeinen Verarbeitung von Informationen (*Information Retrieval, IR*) kommen u.a. auch spezielle Methoden, Verfahren und Technologien zur maschinellen Verarbeitung natürlicher Sprache (*Natural Language Processing, NLP*) zum Einsatz. Die Entwicklung geht bis in die 1950er-Jahre zurück [Bat95] und reicht von der Spracherkennung bis zur syntaktischen, semantischen und pragmatischen Analyse. Das Ziel ist die Verbesserung der Kommunikation zwischen Mensch und Computer. Hierfür werden Erkenntnisse aus der Linguistik mit denen aus der Computerwissenschaft, der Künstlichen Intelligenz sowie des Maschinellen Lernens verknüpft.

Die Aufgaben des NLP sind sehr umfangreich. Daher wird, um den Umfang dieser Abschlussarbeit nicht zu sprengen, nachfolgend eine Auswahl der wichtigsten Aufgaben stichpunktartig aufgelistet sowie anschließend einige Verfahren näher erläutert. Für weitere Informationen sei auf [Bat95], [JuM08] und [MRS08] sowie auf die weitere, umfangreiche Literatur in diesem Bereich (Google Scholar, eine Suchmaschine zur allgemeinen Literaturrecherche wissenschaftlicher Dokumente, listet für die Suchbegriffe *Natural Language Processing* mehr als 2,7 Millionen Ergebnisse [GoS]) verwiesen. Darüber hinaus betreiben viele Universitäten und Forschungseinrichtungen, wie beispielsweise Stanford [NLPG1] und Berkeley [NLPG2] sowie Microsoft Research [NLPG3] oder die Universität Leipzig [NLPG4], eigene NLP-Forschungsgruppen.

Zu den Aufgaben zählen u.a.:

- Zerlegung von Wörtern in Einzelbestandteile (morphologische Segmentierung)
- Entfernung von nicht relevanten Informationen
- Erkennung von Wort- und Satzteilbeziehungen
- Erkennung von Eigennamen (Named Entity Recognition, NER)
- Erkennung von Gefühlen, Stilformen (z.B. Ironie) und Wortbedeutungen
- Erkennung der Textsorte (Diskursanalyse)
- Erkennung von Fragen und Folgefragen
- Plausibilitätsprüfungen von Antworten
- Maschinelle Übersetzung
- Zusammenfassung von Text
- Vereinfachung von Text
- Umwandlung von Text in gesprochene Sprache
- Umwandlung von gesprochener Sprache in Text
- Umwandlung von Bildern in Text (Optical Character Recognition, OCR)

2.2.1 Part-of-Speech Tagging

Als Part-of-Speech Tagging werden Verfahren bezeichnet, die die Wörter und Satzzeichen eines gegebenen Textes den Wortarten zuordnen. Hierbei ist es wichtig, neben der Definition des Wortes auch den Kontext (sowohl des ganzen Satzes/Textes, als auch der angrenzenden Wörter) zu berücksichtigen. Die Einteilung der Wörter erfolgt insbesondere nach drei Funktionen: Die Bedeutung des Wortes (semantische Funktion), die Form des Wortes (morphologische Funktion) und die Verwendung des Wortes im Satz (syntaktische Funktion). Wurde die Kennzeichnung der Wortart ursprünglich manuell durchgeführt, existieren heute viele automatisierte Verfahren, die in überwachtes maschinelles Lernen (vordefiniertes Tagset mit allen Wortart-Tags) und unüberwachtes

maschinelles Lernen (Tagsets entstehen durch stochastische Verfahren) unterteilt werden. Bekannte überwachte Verfahren sind beispielsweise das Eric Brill Verfahren [Bri92] sowie Entscheidungsbäume nach Helmut Schmid [ScH94]. Ein in der deutschen Sprache häufig verwendetes, vordefiniertes Tagset ist das Stuttgart-Tübingen-Tagset (STTS) [STTS95].

2.2.2 Stemming

Unter dem Begriff Stemming (dt. Stammformreduktion) werden Verfahren verstanden, die verschiedene Varianten eines Wortes auf den gemeinsamen Wortstamm zurückführen. Ziel ist es, die Anzahl an unterschiedlichen Wörtern im Text zu reduzieren. Beispielsweise werden die verschiedenen Wortvarianten sehe, *gesehen*, *sah* zu *sehen* reduziert. Auch die Zerlegung und Zusammensetzung von Wörtern ist in einigen Sprachen zu berücksichtigen (z.B. *lief weg* wird zu *weglaufen*). Der erste bekannte Stemming-Algorithmus wurde 1968 von Julie Beth Lovins [Lov68] veröffentlicht. Weitere nennenswerte Stemmer sind der Porter-Stemmer-Algorithmus [Por80] aus dem Jahr 1980 (mittlerweile der De-facto-Standard für englische Texte) sowie KSTEM [Kro93] von 1993. Die entwickelten Stemming-Algorithmen können nicht verifiziert werden, sondern müssen an Textkorpora getestet werden (sogenannte experimentelle Wissenschaft). Weitere Informationen mit ausführlicheren Beschreibungen und Beispielen sind in [MRS08, S.32ff] nachzulesen.

2.2.3 Stoppwörter

Als Stoppwörter (auch Stoppwortliste genannt) werden Wörter bezeichnet, die in einem Text sehr häufig auftreten sowie meist nur eine grammatikalische bzw. syntaktische Funktion besitzen und daher gewöhnlich keinen oder nur einen sehr geringen Rückschluss auf den Inhalt des Textes ermöglichen [MRS08, S.27]. In der deutschen Sprache werden vor allem die bestimmten und unbestimmten Artikel (*der*, *die*, *das*, *einer*, *eine*, *ein*) sowie Konjunktionen (z.B. *und*, *oder*) und Präpositionen (z.B. *an*, *in*, *von*) in eine Stoppwortliste aufgenommen. Die Stoppwörter werden sowohl bei der Erfassung von Texten und Dokumenten als auch bei Suchanfragen nicht beachtet, da andernfalls nahezu jeder Text bzw. jedes Dokument im Suchergebnis erscheinen würde und dieses somit für den Anwender nutzlos wäre.

2.2.4 Tokenizer

Ein Tokenizer zerlegt eine textuelle Eingabe (z.B. ein Textdokument oder einen Quellcode) zur besseren Weiterverarbeitung in logisch zusammengehörige Einheiten, die Token genannt werden [MRS08, S.22ff]. Dieser Vorgang wird auch als lexikalische Analyse bezeichnet. Die Zerlegung kann je nach Anwendungsfall nach verschiedenen Regeln erfolgen. Die textuelle Eingabe kann beispielsweise in die einzelnen Wörter (Verwendung des Leerzeichens als Trennzeichen) oder Sätze (Verwendung von Satzzeichen als Trennzeichen) zerlegt werden.

2.3 Ähnlichkeitsbestimmung

Wie bereits im Kapitel 2.1.4.3 beschrieben, wird zur Erkennung von Duplikaten die Ähnlichkeit zweier Datensätze ermittelt, indem zunächst mit Hilfe einer Vergleichsfunktion die Ähnlichkeit der einzelnen Attributwerte bestimmt wird. Da es sich bei diesen Attributwerten im Kontext dieser Abschlussarbeit hauptsächlich um Zeichenketten handelt, werden im nachfolgenden Kapitel nur die wichtigsten Vergleichsfunktionen zur Ähnlichkeitsbestimmung von Zeichenketten näher untersucht. Darüber hinaus werden weitere Vergleichsfunktionen (z.B. zur Ähnlichkeitsbestimmung von Datumswerten oder Zahlen) kurz angesprochen. Die Auswahl der in dieser Arbeit betrachteten Algorithmen erfolgt auf Basis einer umfangreichen Literaturrecherche. Die einzelnen Quellen hierfür werden bei den entsprechenden Algorithmen genannt. An dieser Stelle soll insbesondere auf [ABEM10], [NaH10] sowie [Nav99] verwiesen werden.

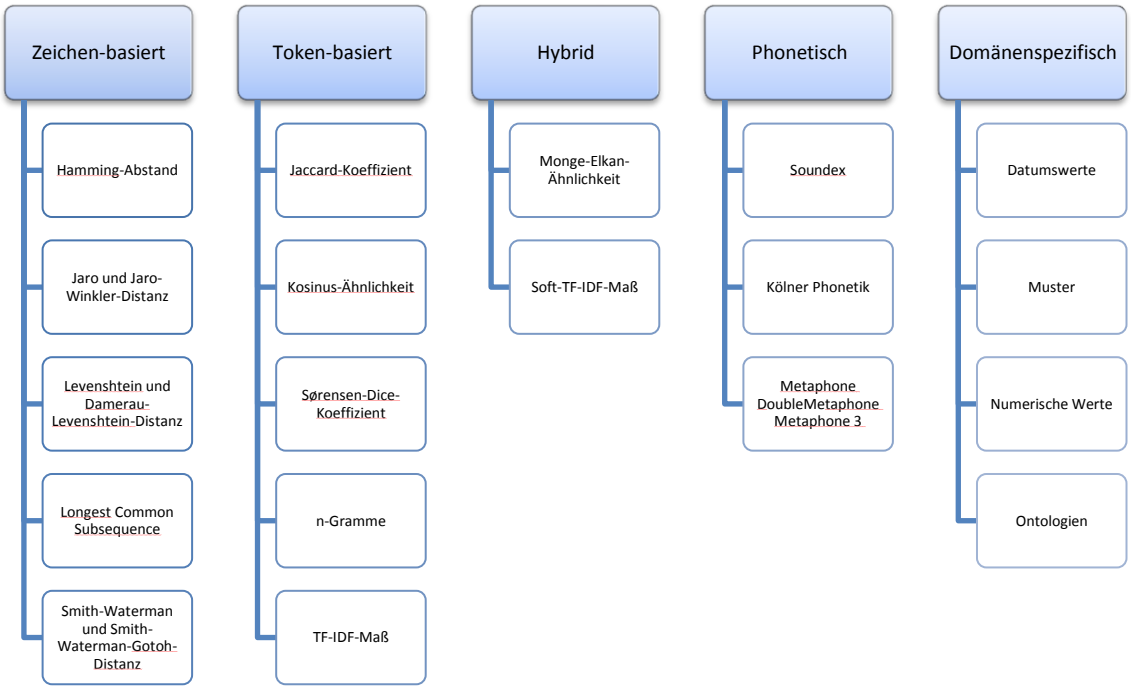

Abbildung 1: Verschiedene Algorithmen zur Ähnlichkeitsbestimmung von Zeichenketten

Neben der Untersuchung von Zeichenketten hinsichtlich ihrer Ähnlichkeit (*Approximate String Matching* [Nav99]) können Zeichenketten auch auf eine exakte Übereinstimmung (*Exact String Matching* [ChL04]) sowie mit Hilfe von Platzhaltern (*Wildcard String Matching* [ScU01]) untersucht werden. In dieser Arbeit wird jedoch nur das *Approximate String Matching* näher betrachtet.

Die Algorithmen zur Ähnlichkeitsbestimmung werden auch als Metrik bzw. Ähnlichkeitsmetrik bezeichnet [ABEM10, S.174] und ermöglichen es, die typografische Abweichung zweier Zeichenketten numerisch zu bewerten (Ähnlichkeitsmaß). Hierfür liefern sie als Ergebnis eine reelle Zahl zwischen 0 (für keine Ähnlichkeit) bis 1 (für völlige Gleichheit).

Zudem existieren einige Algorithmen, die statt dem Ähnlichkeitsmaß den Abstand zwischen zwei Zeichenketten messen und diesen als Ergebnis in Form einer Distanz bzw. eines Distanzmaßes liefern. Im Gegensatz zum Ähnlichkeitsmaß, das ein normiertes Ergebnis zwischen 0 und 1 liefert (wobei die Zahl umso höher ist, je ähnlicher die Zeichenketten sind), ist das nicht normierte Distanzmaß umso kleiner, je ähnlicher die Zeichenketten sind. Das Distanzmaß kann dabei mit nachfolgender Formel problemlos in eine Ähnlichkeitsmaß umgerechnet werden.

$$\text{Ähnlichkeitsmaß}(\text{s1}, \text{s2}) = 1 - \frac{\text{Distanzmaß}(\text{s1}, \text{s2})}{\text{Maximum}(\text{Länge}(\text{s1}), \text{Länge}(\text{s2}))}$$

Die Algorithmen zur Ähnlichkeitsbestimmung von Zeichenketten können gemäß ihrer grundlegenden Funktionsweise in die in Abbildung 1 dargestellten Klassen (Zeichen-basiert, Token-basiert, Hybrid, Phonetisch und Sonstige/Domänenspezifisch) eingeteilt werden. Die Klassen sowie die einzelnen Algorithmen werden in den nachfolgenden Unterabschnitten detaillierter beschrieben.

2.3.1 Zeichen-basiert

Bei den Zeichen-basierten Algorithmen wird die Ähnlichkeit zweier Zeichenketten ermittelt, indem diese buchstabenweise verglichen werden. Da die Algorithmen den Abstand (also die Distanz) in Form von notwendigen Operationen messen, werden diese auch als Distanz-basierte oder Edit-basierte Algorithmen bezeichnet. Zu den möglichen Operationen zählen [Nav99]:

- Einfügen eines Buchstaben
- Löschen eines Buchstaben
- Ändern eines Buchstaben
- Vertauschen von zwei aufeinanderfolgenden Buchstaben

Da eine unterschiedliche Wortreihenfolge bei zwei zu vergleichenden Zeichenketten schnell zu einer hohen Anzahl an notwendigen Operationen und somit zu einem großen Distanzmaß führt, eignen sich diese Algorithmen vor allem zum Vergleich von einzelnen Wörtern.

In dieser Abschlussarbeit werden folgende Zeichen-basierte Algorithmen näher betrachtet:

- Hamming-Abstand
- Jaro und Jaro-Winkler-Distanz
- Levenshtein und Damerau-Levenshtein-Distanz
- Longest Common Subsequence
- Smith-Waterman und Smith-Waterman-Gotoh-Distanz

2.3.1.1 Hamming-Abstand

Funktionsweise. Der Hamming-Abstand ist ein Maß für die Unterschiedlichkeit von Zeichenketten, welches 1950 von dem US-amerikanischen Mathematiker Richard Wesley Hamming im Bell System Technical Journal unter dem Titel *Error-detecting and error-correcting codes* [Ham50] veröffentlicht wurde. Der Abstand gibt dabei die Anzahl der unterschiedlichen Zeichen innerhalb zweier Zeichenketten mit fester Länge an und wird vor allem zur Fehlererkennung und -korrektur bei Datenübertragungen genutzt. Um Zahlen zu vergleichen, werden diese häufig binär kodiert. Anschließend wird der Abstand durch eine XOR-Operation und das Zählen der resultierenden Einsen ermittelt. Der Abstand zum Nullvektor (Zeichenkette gleicher Länge, die mit Nullzeichen gefüllt ist) wird als Hamming-Gewicht bezeichnet.

Definition. Sind ein endliches Alphabet α sowie zwei gleich lange Wörter $x = (x_1, \cdots, x_n)$ und $y = (y_1, \cdots, y_n)$ aus α^n gegeben, dann definiert man den Hamming-Abstand zwischen den beiden Wörtern x und y als:

$$\Delta(x, y) = \left|\{j \in \{1, \dots, n\}, x_j \neq y_j\}\right|$$

Einfaches Beispiel. Die Hamming-Distanz zwischen den Wörtern *Karolin* und *Kerstin* bzw. *Karolin* und *Kathrin* ist jeweils drei. Die Hamming-Distanz zwischen den beiden Zahlen *93* (binär: 1011101) und *73* (binär: 1001001) ist entweder eins (wenn die Zahlen als Zeichenkette interpretiert werden) oder zwei (wenn die Zahlen zuvor binär kodiert und anschließend mit dem XOR-Operator verknüpft (Ergebnis: 0010100) und die resultierenden Einsen gezählt werden).

2.3.1.2 Jaro und Jaro-Winkler-Distanz

Funktionsweise. Die Jaro-Distanz [Jar89], [Jar95] und die Jaro-Winkler-Distanz [Win90], [WiT91] messen den Abstand zweier Zeichenketten und liefern als Ergebnis die Ähnlichkeit auf einer Skala von Null bis Eins. Hierbei gilt, je höher der Werte, desto ähnlicher sind die Zeichenketten. Um die beiden gegebenen Zeichenketten zu vergleichen, werden zunächst die gemeinsamen Zeichen identifiziert und anschließend die Anzahl der notwendigen Zeichenvertauschungen ermittelt. Zusätzlich wird bei der Jaro-Winkler-Distanz ein gemeinsamer Präfix in den Zeichenketten berücksichtigt sowie ein Skalierungsfaktor für eine gewichtete Bewertung eingeführt. Die Algorithmen eigenen sich vor allem für den Vergleich von kurzen Zeichenketten, insbesondere für den Vergleich von Namen (von Personen, Städten, Ländern, etc.). Im Gegensatz zu anderen Zeichen-basierten Algorithmen, wie beispielsweise Levenshtein, Damerau-Levenshtein, Smith-Waterman und Smith-Waterman-Gotoh, wird die Jaro- und Jaro-Winkler-Distanz nicht mit Hilfe der Methode der dynamischen Programmierung [Bel57] implementiert (Erläuterung der Methode im Kapitel 2.3.1.3).

Definition. Die Jaro-Distanz d_{jaro} von zwei gegebenen Zeichenketten x und y ist wie folgt definiert:

$$d_{jaro} = \begin{cases} 0 & falls\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m - 0.5t}{m}\right) & ansonsten \end{cases}$$

Hierbei ist m die Anzahl der gemeinsamen Zeichen und t die Anzahl der notwendigen Zeichenvertauschungen. Als gemeinsame Zeichen werden Zeichen bezeichnet, die gleich sind und deren Positionen innerhalb der beiden Zeichenketten nicht mehr als

$$\left\lfloor \frac{max(|x|, |y|)}{2} \right\rfloor - 1$$

voneinander abweichen.

Die Jaro-Winkler-Distanz d_{jw} basiert auf einer gewichteten Bewertung der Jaro-Distanz d_{jaro} und wird um ein gemeinsames Präfix p, welches auf maximal vier beschränkt ist, sowie einen Skalierungs- bzw. Korrekturfaktor f im Bereich $\{0{,}1; \dots; 0{,}25\}$ erweitert. Nach [Win90] ist der Standardwert für f auf 0,1 festgesetzt und der Präfix-Bonus wird nur angewandt, wenn die Jaro-Distanz größer als ein Schwellwert b_t ist. Dieser hat nach Winkler's Implementierung den Wert 0,7.

$$d_{jw} = \begin{cases} d_{jaro} & falls\ d_{jaro} < b_t \\ d_{jaro} + \left(p \times f \times \left(1 - d_{jaro}\right)\right) & ansonsten \end{cases}$$

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *Mar<u>tha</u>* und *Mar<u>hta</u>*. Die Anzahl der gemeinsamen Zeichen beträgt $m = 6$, die Anzahl der Zeichenvertauschungen ist $t = 2$ und die Länge der Zeichenketten ist $|x| = 6$ und $|y| = 6$. Hieraus erfolgt eine Jaro-Distanz von

$$d_{jaro}(\text{"}Martha\text{"}, \text{"}Marhta\text{"}) = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6 - 0.5 \times 2}{6}\right) \approx 0{,}944$$

Die Länge des gemeinsamen Präfixes ist $p = 3$ (für "Mar"). Für den Skalierungsfaktor wird der von Winkler vorgeschlagene Wert $f = 0{,}1$ verwendet. Hieraus ergibt sich eine Jaro-Winkler-Distanz von

$$d_{jw}(\text{"}Martha\text{"}, \text{"}Marhta\text{"}) = 0{,}944 + (3 \times 0{,}1 \times (1 - 0{,}944)) \approx 0{,}961$$

2.3.1.3 Levenshtein und Damerau-Levenshtein-Distanz

Funktionsweise. Der Levenshtein-Algorithmus [Lev66], der 1965 von dem russischen Mathematiker und Wissenschaftler Wladimir Lewenstein (wiss. Transliteration: Vladimir Levenštejn) entwickelt wurde, ermittelt die minimale Anzahl an Einfüge-, Lösch- und Ersetz-Operationen, um eine gegebene Zeichenkette in eine zweite, gegebene Zeichenkette zu überführen. Diese Mindestanzahl an notwendigen Editieroperationen wird Levenshtein-Distanz oder Editierdistanz genannt. Der Algorithmus wird meist nach dem Prinzip der dynamischen Programmierung [Bel57] implementiert. Die dynamische Programmierung ist eine in den 1940er Jahren von dem amerikanischen Mathematiker Richard Bellman eingeführte Methode zum algorithmischen Lösen von Optimierungsproblemen. Diese Methode kann vor allem dann erfolgreich eingesetzt werden, wenn das zu lösende Optimierungsproblem in viele gleichartige Teilprobleme zerlegt werden kann. Die optimale Lösung für das Problem ergibt sich schließlich aus dem Zusammensetzen der optimalen Lösungen der Teilprobleme. Der große Vorteil dieser Methode ist, dass kostspielige Rekursionen vermieden werden, da einmal berechnete Teilergebnisse in einer Tabelle zwischengespeichert und wiederverwendet werden.

Um die Levenshtein-Distanz zu errechnen, wird eine zweidimensionale Matrix erstellt. Anschließend wird von der oberen linken zur unteren rechten Ecke für jede Zelle die Levenshtein-Distanz zwischen dem i-ten Buchstaben des ersten Wortes und dem j-ten Buchstaben des zweiten Wortes ermittelt. Jeder Sprung (sowohl horizontal und vertikal als auch diagonal) wird mit virtuellen Kosten für die Editierung versehen. Wird horizontal oder vertikal gesprungen, entspricht dies einer Einfüge- oder Löschoperation. Hierfür werden meist virtuelle Kosten von "1" angesetzt. Für eine Ersetz-Operation wird diagonal gesprungen. Stimmen dabei der i-te und j-te Buchstabe überein, werden virtuelle Kosten von "0" angesetzt. Bei unterschiedlichen Buchstaben wird der Sprung mit virtuelle Kosten von "1" bewertet. Da der lokale Aufwand bei jedem Sprung minimiert wird, enthält die Zelle in der rechten unteren Ecke die Mindestanzahl an Editieroperationen und somit die Levenshtein-Distanz.

Neben der oben beschriebenen, einfachen Levenshtein-Distanz wird der Algorithmus in weiteren, leicht abgewandelten Formen implementiert. Die bekanntesten Varianten werden nachfolgend kurz dargestellt. Zudem gibt es einige verwandte, in dieser Arbeit nicht näher betrachtete Algorithmen wie z.B. der Hirschberg-Algorithmus [Hir75] und der Needleman-Wunsch-Algorithmus [NeW70].

Bei der *Gewichteten Levenshtein-Distanz* können zum einen die verschiedenen, durchgeführten Editieroperationen und zum anderen die einzelnen Zeichen und Buchstaben unterschiedlich gewichtet werden. Werden beispielsweise höhere virtuelle Kosten für die Operationen Einfügen und Löschen angesetzt, können Zeichenketten mit gleicher Länge gegenüber Zeichenketten mit

unterschiedlicher Länge besser bewertet werden. Die unterschiedliche Gewichtung von einzelnen Zeichen und Buchstaben findet u.a. bei der sogenannten Schreibmaschinendistanz Anwendung. Bei diesem Berechnungs-Verfahren wird bei der Gewichtung die Entfernung der Zeichen und Buchstaben auf der Tastatur berücksichtigt.

Die *Damerau-Levenshtein-Distanz* [Dam64] erweitert die einfache Levenshtein-Distanz um die Eigenschaft, zwei vertauschte Zeichen zu identifizieren. Wird zum Beispiel das falsch geschriebene Wort *Levensthein* mit dem richtigen Wort *Levenshtein* verglichen, sind bei der einfachen Levenshtein-Distanz zwei Editieroperationen notwendig, bei der Damerau-Levenshtein-Distanz hingegen nur eine.

Definition. Die Implementierung des Algorithmus basiert meist auf dem Prinzip der dynamischen Programmierung nach [Bel57]. Soll die Levenshtein-Distanz zwischen den beiden Wörtern x und y ermittelt werden, wird hierfür zunächst eine Matrix der Größe $(|x|+1) \times (|y|+1)$ initialisiert. Die Werte der einzelnen Zellen $M_{i,j}$ (mit Spalte i und Zeile j) der Matrix werden dabei folgendermaßen berechnet:

$$
\begin{aligned}
&M_{0,0} = 0 \\
&M_{i,0} = i, 1 \leq i \leq |x| \\
&M_{0,j} = j, 1 \leq j \leq |y| \\
&M_{i,j} = min \begin{cases} M_{i-1,j-1} & falls\ x_i = y_j \\ M_{i-1,j-1} + 1 & (Ersetzen) \\ M_{i,j-1} + 1 & (Einfügen) \\ M_{i-1,j} + 1 & (Löschen) \end{cases} \\
&1 \leq i \leq |x|, 1 \leq j \leq |y|
\end{aligned}
$$

Die Damerau-Levenshtein-Distanz erweitert die Levenshtein-Distanz um die Fähigkeit, zwei vertauschte Zeichen zu identifizieren. Hierzu wird die oben abgedruckte Definition der Levenshtein-Distanz um nachfolgende mathematische Definition erweitert (c bezeichnet hierbei die Kosten für die Vertauschung zweier Zeichen):

$$
\begin{aligned}
&M_{i,j} = min \begin{cases} M_{i-1,j-1} & falls\ x_i = y_j \\ M_{i-1,j-1} + 1 & (Ersetzen) \\ M_{i,j-1} + 1 & (Einfügen) \\ M_{i-1,j} + 1 & (Löschen) \\ M_{i-2,j-2} + c & (Vertauschen, falls\ x_i = y_{j-1} \wedge x_{i-1} = y_j) \end{cases} \\
&2 \leq i \leq |x|, 2 \leq j \leq |y|
\end{aligned}
$$

Einfaches Beispiel. In der nachfolgenden Matrix wird die Berechnung der Levenshtein-Distanz zwischen den beiden Wörtern *Meilenstein* und *Levenshtein* beispielhaft aufgezeigt.

		M	**e**	**i**	**l**	**e**	**n**	**s**	**t**	**e**	**i**	**n**
	0	1	2	3	4	5	6	7	8	9	10	11
L	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

Die Levenshtein-Distanz zwischen den beiden Wörtern beträgt 4 und kann in der rechten unteren Ecke der Matrix abgelesen werden. Um zu diesem Ergebnis zu gelangen, existieren zwei mögliche Wege durch die obige Matrix, die den niedrigsten Aufwand produzieren. Diese sind:

```
M e i l e n s   t e i n          M e i l e n s   t e i n
L e   v e n s h t e i n          L e v   e n s h t e i n
~ = + ~ = = = - = = = =          ~ = ~ + = = = - = = = =
```

= Übereinstimmung; ~ Ersetzen; + Einfügen; - Löschen

Die nachfolgenden Tabellen zeigen die Levenshtein-Distanz (links) sowie die Damerau-Levenshtein-Distanz (rechts) zwischen den beiden Zeichenketten *an act* und *a cat*. Während bei der Levenshtein-Distanz zwei Änderungs-Operationen für die Buchstaben *ac* und *ca* notwendig sind, erfolgt dies bei der Damerau-Levenshtein-Distanz mit nur einer Vertausch-Operation. Zudem wird bei beiden eine Einfüge-Operation für den fehlenden Buchstaben *n* benötigt. Die relevanten Zellen sind nachfolgend farblich hervorgehoben.

		a	**n**		**a**	**c**	**t**
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
c	3	2	2	2	2	2	3
a	4	3	3	3	2	3	3
t	5	4	4	4	3	3	3

		a	**n**		**a**	**c**	**t**
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
c	3	2	2	2	2	2	3
a	4	3	3	3	2	2	3
t	5	4	4	4	3	3	2

2.3.1.4 Longest Common Subsequence

Funktionsweise. Das Longest Common Subsequence (LCS) Problem beschreibt die Herausforderung, die längste gemeinsame Teilsequenz zu finden, die in einer Menge von Sequenzen enthalten ist. Es ist in die klassische Informatik eingebettet, bildet die Basis für Programme zum Vergleich von Daten (z.B. bei Systemen zur Versionskontrolle) und findet ebenso in der Bioinformatik, beispielsweise bei der Verarbeitung von DNA-Sequenzen, Anwendung. Wichtig ist hierbei die Unterscheidung zwischen Longest Common Subsequence und Longest Common Substring. Während es beim Auffinden von Substrings notwendig ist, dass die Teilzeichenfolgen immer zusammenhängend in der Originalsequenz vorhanden sind, ist beim Auffinden von Subsequences nur die Reihenfolge wichtig. *aba* ist beispielsweise sowohl eine Subsequence als auch ein Substring von *cabad*. *cbd* ist hingegen nur eine Subsequence von *cabad*, allerdings kein Substring. Wie bereits eingangs erwähnt, ist es das Ziel von LCS, die längste gemeinsame Teilsequenz zu finden. Ein möglicher, wenn auch naiver und zeitintensiver Ansatz wäre es, alle möglichen Teilsequenzen von A zu betrachten und anschließend zu untersuchen, welche hiervon auch eine Teilsequenz von B sind sowie sich kontinuierlich die längste gefundene Teilsequenz zu merken. Das LCS Problem kann allerdings auch als Rekursionsgleichung oder dem Prinzip der dynamischen Programmierung gelöst sowie die Algorithmen hinsichtlich der Ausführungszeit und des Speicherbedarfs optimiert werden [HuS77], [MaP80]. Da es zum Thema Teilsequenzen sehr viele Veröffentlichungen gibt, sei an dieser Stelle für einen ersten Überblick sowie für den Vergleich von verschiedenen Algorithmen auf [BHR00] verwiesen.

Definition. Sind zwei Sequenzen $X = (x_1, x_2, \dots, x_m)$ und $Y = (y_1, y_2, \dots, y_n)$ mit $i = |X|$ und $j = |Y|$ gegeben, dann ist die Länge der gemeinsamen Teilsequenz definiert durch:

$$LCS(X_i, Y_j) = \begin{cases} 0 & falls\ i = 0\ oder\ j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & falls\ x_i = y_i \\ Maximum\left(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\right) & falls\ x_i \neq y_j \end{cases}$$

Einfaches Beispiel. Es sind folgende zwei Sequenzen gegeben:

Professor Doktor Max-Emanuel Müller
Professor Doktor Maximilian Müller

Dann hat die Longest Common Subsequence eine Länge von 27 und ist:

Professor Doktor Max Müller

2.3.1.5 Smith-Waterman und Smith-Waterman-Gotoh-Distanz

Funktionsweise. Die in Kapitel 2.3.1.3 beschriebene Levenshtein-Distanz sowie die verschiedenen Implementierungs-Varianten davon sind vor allem für den Vergleich von kurzen Zeichenketten mit möglichst wenigen Abweichungen (z.B. Tippfehlern, Rechtschreibfehlern oder Zeichendrehern) geeignet. Werden jedoch Abkürzungen (z.B. *Professor Doktor* und *Prof. Dr.* oder *Hans Joachim Hof* und *Hans J. Hof*) oder Präfixe bzw. Suffixe (z.B. *Prof. Dr. Max Mustermann* und *Max Mustermann*) verwendet, sind diese Algorithmen weniger geeignet, da viele Einfüge- bzw. Lösch-Operationen notwendig sind. Die Smith-Waterman [SmW81] und die Smith-Waterman-Gotoh-Distanz [Got82] funktionieren ähnlich wie die Levenshtein-Distanz, erweitern die Vergleichs-Funktionalität jedoch um die Erkennung und effiziente Verarbeitung von Abkürzungen und Präfixen bzw. Suffixen.

Die Smith-Waterman-Distanz fokussiert sich auf das unterschiedliche Bewerten von vorhandenen Präfixen bzw. Suffixen. Hierzu wird zunächst die längste gemeinsame Teilsequenz beider Zeichenketten ermittelt und anschließend die Zeichenketten in Präfix, Teilsequenz und Suffix zerlegt. Da beim Einfügen bzw. Löschen eines kompletten Präfix bzw. Suffix geringere virtuelle Kosten entstehen als beim Einfügen bzw. Löschen jedes einzelnen Zeichens des Präfixes bzw. Suffixes ist die ermittelte Smith-Waterman-Distanz geringer als die Levenshtein-Distanz.

Um mit Abkürzungen und fehlenden Fragmenten besser umgehen zu können, werden neben den bekannten Editier-Operationen der Levenshtein-Distanz (Einfügen, Ändern und Löschen von einzelnen Zeichen) zwei neue Operationen für das Einfügen und Löschen von vollständigen Zeichenblöcken eingeführt. Die virtuellen Kosten für diese neuen Operationen sind dabei wesentlich niedriger als die Summe der Kosten für das Einfügen bzw. Löschen jedes einzelnen Zeichens innerhalb eines Zeichenblocks und werden auch als affine Gapkosten bezeichnet [WSB76]. Die Smith-Waterman-Gotoh-Distanz ist eine Erweiterung der Smith-Waterman-Distanz und berücksichtigt neben dem Präfix und Suffix auch die eben erwähnten affinen Gapkosten.

Beide Algorithmen werden nach dem Prinzip der dynamischen Programmierung nach [Bel57] implementiert.

Definition. Um die Smith-Waterman-Distanz zwischen den beiden Wörtern x und y zu ermittelt, wird hierfür zunächst - konstant zur Ermittlung der Levenshtein-Distanz - eine Matrix der Größe $(|x|+1) \times (|y|+1)$ initialisiert. Die Werte der einzelnen Zellen $M_{i,j}$ der Matrix werden anschließend mit einer Alignment-Score-Funktion $W(a,b)$ folgendermaßen berechnet:

$$M_{i,0} = 0, 0 \leq i \leq |x|$$
$$M_{0,j} = 0, 0 \leq j \leq |y|$$
$$M_{i,j} = max \begin{cases} 0 & leeres\ Suffix \\ M_{i-1,j-1} + W(a_i, b_j) & Match\ bzw. Mismatch \\ M_{i,j-1} + W(-, b_j) & Einfügen \\ M_{i-1,j} + W(a_i, -) & Löschen \end{cases}$$
$$1 \leq i \leq |x|, 1 \leq j \leq |y|$$

Das Zeichen $-$ steht dabei für einen GAP-Charakter und bezeichnet beim Sequenzalignment (einem Teilgebiet des Pattern Matching, welches häufig in der Bioinformatik zum Einsatz kommt) eine Lücke bzw. Leerstelle. Dies bedeutet, dass an der entsprechenden Stelle der zu vergleichenden Zeichenkette/Sequenz ein weiteres Element steht.

Die Alignment-Score-Funktion $W(a,b)$ könnte beispielsweise wie folgt definiert sein:

$$W(a,b) = \begin{cases} +2 & wenn\ x = y\ (Match) \\ -1 & wenn\ x = -\ oder\ y = -\ oder\ x \neq y\ (Mismatch) \end{cases}$$

Die Erweiterungen der Smith-Waterman-Gotoh-Distanz um affine Gapkosten können in [Got82] und [WSB76] sowie der einschlägigen Literatur nachgelesen werden.

Einfaches Beispiel. Gegeben sind die beiden Sequenzen *BFKB* und *HFFK*. Die Zellen der Matrix (links) werden nach der obigen Definition gefüllt (ähnlich zur Levenshtein-Distanz). Als Alignment-Score-Funktion wird ebenfalls die oben definierte Beispielfunktion verwendet. Anschließend wird, ausgehend von der Zelle mit der größten Distanz, diagonal zurückgewandert; bis eine Zelle mit dem Wert 0 erreicht wird (rechte Matrix). Das Ergebnis ist das optimale lokale Alignment (längste gemeinsame Teilsequenz). Im Beispiel ist dies *FK*.

	–	**B**	**F**	**K**	**B**
–	0	0	0	0	0
H	0	0	0	0	0
F	0	0	2	1	0
F	0	0	2	1	0
K	0	0	1	4	3

	–	**B**	**F**	**K**	**B**
–	0	0	0	0	0
H	0	0	0	0	0
F	0	0	2	1	0
F	0	0	2	1	0
K	0	0	1	4	3

2.3.2 Token-basiert

Token-basierte Ähnlichkeitsalgorithmen ermitteln die Ähnlichkeit zweier Zeichenketten, indem diese zunächst in einzelne Token zerlegt werden und anschließend verglichen wird, wie viele Token die Zeichenketten gemeinsam haben [NaH10]. Die Zerlegung in einzelne Token kann sowohl nach vorgegebenen Trennzeichen (z.B. Leer- oder Satzzeichen) als auch nach festgelegten Intervallen (z.B. Zerlegung in Buchstabenpärchen; u.a. die nachfolgend näher betrachteten n-Gramme in Kapitel 2.3.2.4) erfolgen. Die Zeichenkette *Max Mustermann* würde bei Verwendung des Leerzeichens als Trennzeichen beispielsweise in die Token *{Max, Mustermann}* aufgeteilt werden. Durch die Zerlegung in Token sind die Token-basierten Ähnlichkeitsalgorithmen hinsichtlich vertauschter Wörter in den zu vergleichenden Zeichenketten weniger empfindlich als die Zeichen-basierten Algorithmen und eignen sich dadurch sehr gut für den Vergleich längerer Zeichenketten mit mehreren Wörtern.

In dieser Abschlussarbeit werden folgende Token-basierte Algorithmen näher betrachtet:

- Jaccard-Koeffizient
- Kosinus-Ähnlichkeit
- Sørensen-Dice-Koeffizient
- n-Gramme
- TF-IDF-Maß

2.3.2.1 Jaccard-Koeffizient

Funktionsweise. Der Jaccard-Koeffizient [Jac01] nach dem Schweizer Botaniker Paul Jaccard dient zur Ermittlung der Ähnlichkeit zweier Mengen und gibt das Verhältnis der Anzahl der gemeinsamen Elemente, also der Schnittmenge, zur Größe der Vereinigungsmenge an. Ein Nachteil des Jaccard-Koeffizienten ist, dass bereits eine kleine Abweichung der gemeinsamen Token (z.B. nur ein einziges Token unterscheidet sich) zu einem großen Unterschied bei der Ähnlichkeit führt (siehe Beispiel).

Definition. Gegeben sind die zwei Mengen X und Y als Mengen von Token zweier Zeichenketten. Der Jaccard-Koeffizient ist definiert als das Verhältnis der Schnittmenge von X und Y zur Vereinigungsmenge von X und Y:

$$Jaccard(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

Die Jaccard-Distanz ermittelt den Abstand zwischen zwei gegebenen Mengen und ist definiert durch:

$$d_{Jaccard}(X,Y) = 1 - Jaccard(X,Y) = \frac{|X \cup Y| - |X \cap Y|}{|X \cup Y|}$$

Da die Jaccard-Distanz die Eigenschaften der Dreiecksungleichung $d(X,Z) \leq d(X,Y) + d(Y,Z)$ erfüllt, handelt es sich um eine ordnungsgemäße Distanz-Metrik [Lip99], [LeW71].

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *Herr Alex Muster* und *Frau Alex Muster* bzw. die daraus resultierenden Token-Mengen *{Herr, Alex, Muster}* und *{Frau, Alex, Muster}*. Daraus ergibt sich ein Jaccard-Koeffizient bzw. eine Jaccard-Distanz von:

$$Jaccard(\{Herr, Alex, Muster\}, \{Frau, Alex, Muster\}) = 0{,}5$$

$$d_{Jaccard}(\{Herr, Alex, Muster\}, \{Frau, Alex, Muster\}) = 0{,}5$$

2.3.2.2 Kosinus-Ähnlichkeit

Funktionsweise. Die Kosinus-Ähnlichkeit kommt vor allem im Bereich des Information Retrieval und des Text- bzw. Data-Mining zum Einsatz [Sin01]. Hierbei wird die Ähnlichkeit von zwei Vektoren ermittelt, indem der Kosinus des Winkels zwischen den beiden Vektoren bestimmt wird. Die Werte der Kosinus-Ähnlichkeit reichen dabei von -1 (genau entgegengesetzt gerichtete Vektoren) bis 1 (exakt gleichgerichtete Vektoren). Im Bereich des textuellen Vergleichs (z.B. beim Suchen von Wörtern in Dokumenten oder beim Vergleich von Textdateien) kommen häufig sogenannte Häufigkeits-Vektoren (z.B. TF-IDF, Kapitel 2.3.2.5) zum Einsatz, da deren Gewicht nie negativ ist und somit die Kosinus-Ähnlichkeit immer zwischen 0 und 1 liegt [Coh98].

Definition. Die Kosinus-Ähnlichkeit zweier Vektoren X und Y ist definiert als der Kosinus des eingeschlossenen Winkels θ:

$$Kosinus(X,Y) = cos\,\theta = \frac{X \times Y}{||X|| \times ||Y||} = \frac{\sum_{i=1}^{n} X_i \times Y_i}{\sqrt{\sum_{i=1}^{n} (X_i)^2} \times \sqrt{\sum_{i=1}^{n} (Y_i)^2}}$$

Der Kosinus-Abstand ermittelt den Abstand zwischen zwei gegebenen Vektoren und ist definiert durch:

$$d_{Kosinus}(X,Y) = 1 - Kosinus(X,Y)$$

Im Gegensatz zum Jaccard-Abstand ist der Kosinus-Abstand keine ordnungsgemäße Distanz-Metrik, da die Eigenschaften der Dreiecksungleichung $d(X,Z) \leq d(X,Y) + d(Y,Z)$ nicht erfüllt werden [Gal99].

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *Herr Alex Muster* und *Frau Alex Muster*. Zunächst wird eine Liste aller Wörter erstellt und anschließend gezählt, wie oft die Wörter in den Zeichenketten vorkommen. Dies sind auch gleichzeitig die beiden Vektoren X und Y.

	Herr	**Frau**	**Alex**	**Muster**
X	1	0	1	1
Y	0	1	1	1

Aus den beiden Vektoren ergibt sich ein Kosinus-Koeffizient von:

$$Kosinus(\{1,0,1,1\},\{0,1,1,1\}) = \frac{2}{3}$$

2.3.2.3 Sørensen-Dice-Koeffizient

Funktionsweise. Der Sørensen-Dice-Koeffizient [Sor48], [Dic45] (häufig auch Dice's Koeffizient oder Sørensen-Index genannt) wurde von den beiden Botanikern Thorvald Sørensen und Lee Raymond Dice entwickelt, um die Ähnlichkeit zweier Mengen bzw. Stichproben zu bestimmen und diese zu vergleichen. Der Sørensen-Dice-Koeffizient ist sehr ähnlich zum Jaccard-Koeffizienten, gibt jedoch, im Gegensatz zu diesem, das Verhältnis der zweifachen Anzahl der gemeinsamen Elemente zur Summe aller Elemente an. Wird der Sørensen-Dice-Koeffizient im Bereich des Information Retrieval eingesetzt um die Ähnlichkeit von Zeichenketten zu ermitteln, werden zur Bildung der Mengen häufig n-Gramme (siehe Kapitel 2.3.2.4) verwendet [KMK03].

Definition. Gegeben sind die zwei Mengen X und Y als Mengen von Token zweier Zeichenketten. Der Sørensen-Dice-Koeffizient ist definiert als das Verhältnis der zweifachen Schnittmenge von X und Y zur Summe aller Elemente von X und Y:

$$SørensenDice(X,Y) = \frac{2 \times |X \cap Y|}{|X| + |Y|}$$

Der Sørensen-Dice-Abstand ermittelt den Abstand zwischen zwei gegebenen Mengen bzw. Stichproben und ist definiert durch:

$$d_{SørensenDice}(X,Y) = 1 - SørensenDice(X,Y)$$

Obwohl der Sørensen-Dice-Koeffizient sehr ähnlich zum Jaccard-Koeffizienten ist, ist der Sørensen-Dice-Abstand im Gegensatz zum Jaccard-Abstand keine ordnungsgemäße Distanz-Metrik, da die Eigenschaften der Dreiecksungleichung $d(X,Z) \leq d(X,Y) + d(Y,Z)$ nicht erfüllt werden [Gal99].

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *night* und *nacht*. Diese werden unter Verwendung von Bi-Grammen (weitere Informationen dazu in nachfolgenden Kapitel 2.3.2.4) in die beiden Mengen *{#n,ni,ig,gh,ht,t#}* und *{#n,na,ac,ch,ht,t#}* zerlegt. Da die beiden Mengen drei gemeinsame Elemente besitzen, ergibt sich ein Sørensen-Dice-Koeffizient von:

$$SørensenDice(\{\#n, ni, ig, gh, ht, t\#\}, \{\#n, na, ac, ch, ht, t\#\}) = 0{,}5$$

2.3.2.4 n-Gramme

Funktionsweise. Wie bereits eingangs erwähnt, werden bei Token-basierten Ähnlichkeitsalgorithmen die Zeichenketten zunächst in einzelne Token zerlegt. Diese Zerlegung der Zeichenketten kann nach vorgegebenen Trennzeichen (z.B. Leerzeichen) oder in aufeinanderfolgende Fragmente der Länge n erfolgen. Die sogenannten n-Gramme (teilweise auch als q-Gramme bezeichnet) sind schließlich das Ergebnis der Zerlegung einer Zeichenkette in aufeinanderfolgende Fragmente und werden, abhängig von n, entsprechend bezeichnet (Monogramm [n=1], Bigramm [n=2], Trigramm [n=3], usw.). Hierbei kann auch eine Überlappung der Token erzielt werden, indem für jedes zu bildende Fragment das Fenster um beispielsweise 1 statt n Zeichen verschoben wird. Wird die Überlappung verwendet, sollten $n-1$ Platzhalterzeichen am Anfang und Ende der zu vergleichenden Zeichenketten hinzugefügt werden, damit auch für die ersten und letzten Zeichen entsprechende n-Gramme gebildet werden können. Dies ist jedoch nicht vorgeschrieben und wird daher von diversen Implementierungen unterschiedlich gehandhabt. n-Gramme kommen vor allem in der Computerlinguistik (u.a. im Natural Language Processing, siehe Kapitel 2.2), in der Biologie (z.B. bei der Analyse von DNA- oder Protein-Sequenzen) oder in der Kryptologie zum Einsatz. Wolfgang Schönpflug hat beispielsweise 1969 die n-Gramm-Häufigkeiten in der deutschen Sprache analysiert [Sch69] und auch die Firma Google stellt eine Auswertung von n-Grammen der bei Google Books vorhandenen Bücher zur Verfügung [GoB].

Definition. Die erzeugten n-Gramme können als Grundlage für verschiedene Token-basierte Ähnlichkeitsalgorithmen, wie beispielsweise Jaccard-Koeffizient, Kosinus-Ähnlichkeit oder Sørensen-Dice-Koeffizient, verwendet werden. Die Qualität der Ergebnisse ist dabei sowohl von dem gewählten Algorithmus und dem gewählten n als auch von verschiedenen Faktoren bzgl. der Zeichenketten (u.a. Datenqualität allgemein, Sprache, verwendete Sonderzeichen, etc.) abhängig.

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten $X = "Herr\ Alex\ Muster"$ und $Y = "Frau\ Alex\ Muster"$ mit folgenden Mono-, Bi- und Trigrammen (Leerzeichen werden zur besseren Lesbarkeit beispielhaft mit einem Paragraf-Zeichen (§) kodiert; Die Raute (#) ist beispielhaft das verwendete Platzhalterzeichen):

$$X_1 = \{\mathrm{H, e, r, r, §, A, l, e, x, §, M, u, s, t, e, r}\}$$

$$X_2 = \{\mathrm{\#H, He, er, rr, r§, §A, Al, le, ex, x§, §M, Mu, us, st, te, er, r\#}\}$$

$$X_3 = \{\mathrm{\#\#H, \#He, Her, err, rr§, r§A, §Al, Ale, lex, el§, l§M, §Mu, Mus, ust, ste, ter, er\#, r\#\#}\}$$

$$Y_1 = \{\mathrm{F, r, a, u, §, A, l, e, x, §, M, u, s, t, e, r}\}$$

$$Y_2 = \{\mathrm{\#F, Fr, ra, au, u§, §A, Al, le, ex, x§, §M, Mu, us, st, te, er, r\#}\}$$

$$Y_3 = \{\mathrm{\#\#F, \#Fr, Fra, rau, au§, u§A, §Al, Ale, lex, el§, l§M, §Mu, Mus, ust, ste, ter, er\#, r\#\#}\}$$

Hieraus ergeben sich folgende Jaccard- und Sørensen-Dice-Koeffizienten:

$$Jaccard(X_1, Y_1) \approx 0{,}77$$

$$Jaccard(X_2, Y_2) \approx 0{,}57$$

$$Jaccard(X_3, Y_3) \approx 0{,}5$$

$$SørensenDice(X_1, Y_1) \approx 0{,}87$$

$$SørensenDice(X_2, Y_2) \approx 0{,}73$$

$$SørensenDice(X_3, Y_3) \approx 0{,}67$$

2.3.2.5 TF-IDF-Maß

Funktionsweise. Das TF-IDF-Maß [BaR99] wird im Bereich des Information Retrieval eingesetzt, um die Relevanz von Ausdrücken in Dokumenten einer Dokumentensammlung zu beurteilen und kommt häufig bei Suchmaschinen zum Einsatz. Die Abkürzung TF-IDF steht hierbei für *Term-Frequency-Inverse-Document-Frequency*. Das TF-IDF-Maß wächst proportional zur Vorkommenshäufigkeit des Ausdrucks innerhalb eines Dokuments (Term-Frequency, auch Suchwortdichte bezeichnet), wird dabei jedoch mit der Vorkommenshäufigkeit des Ausdrucks über alle Dokumente hinweg (die sogenannte Inverse Dokumenthäufigkeit) gegengerechnet. Dadurch kann berücksichtigt werden, ob ein Ausdruck in wenigen Dokumenten sehr häufig vorkommt oder in vielen Dokumenten enthalten ist, dafür aber in den einzelnen Dokumenten nur sehr selten. Es handelt sich dabei nicht um ein normiertes Ergebnis, d.h. es befindet sich nicht innerhalb eines gegebenen Intervalls.

Definition. Das TF-IDF-Maß mit dem gesuchten Ausdruck t sowie dem gegebenen Dokument d aus der ebenfalls gegebenen Dokumentensammlung D ist folgendermaßen definiert:

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Die Vorkommenshäufigkeit $TF(t,d)$ liefert als Ergebnis, wie häufig der Ausdruck t im Dokument d vorkommt. Damit dieses Ergebnis in langen Dokumenten nicht verzerrt wird, kann die Vorkommenshäufigkeit normalisiert werden. Dies kann beispielsweise dadurch erreicht werden, dass durch die maximale Vorkommenshäufigkeit aller enthaltenen Ausdrücke geteilt wird oder die Logarithmus-Funktion angewandt wird [MRS08]:

$$TF(t,d) = \frac{f(t,d)}{max\{f(w,d): w \in d\}}$$

$$TF(t,d) = \begin{cases} 0 & falls\ f(t,d) = 0 \\ 1 + \log\big(f(t,d)\big) & ansonsten \end{cases}$$

Die Inverse Dokumenthäufigkeit lässt sich mit Hilfe der Anzahl der Dokumente in der Dokumentensammlung sowie der Anzahl der Dokumente, in denen der Ausdruck t enthalten ist ermitteln:

$$IDF(t,D) = \log\left(\frac{|D|}{|\{d \in D\ :\ t \in d\}|}\right)$$

Einfaches Beispiel. Gegeben sind zwei Dokumente mit folgenden Inhalten:

$$d_1 = Das\ ist\ ein\ Beispieltext$$
$$d_2 = Das\ ist\ ein\ weiteres\ Beispiel\ eines\ Beispiel\ Textes$$

$$TFIDF(\text{"Das"}, d_1, \{d_1, d_2\}) = 1 \times log\left(\frac{2}{2}\right) = 0$$

$$TFIDF(\text{"Beispieltext"}, d_1, \{d_1, d_2\}) = 1 \times log\left(\frac{2}{1}\right) \approx 0{,}301$$

$$TFIDF(\text{"Textes"}, d_2, \{d_1, d_2\}) = 1 \times log\left(\frac{2}{1}\right) \approx 0{,}301$$

$$TFIDF(\text{"Beispiel"}, d_1, \{d_1, d_2\}) = 0 \times log\left(\frac{2}{1}\right) = 0$$

$$TFIDF(\text{"Beispiel"}, d_2, \{d_1, d_2\}) = 2 \times log\left(\frac{2}{1}\right) \approx 0{,}602$$

2.3.3 Hybrid

In diesem Abschnitt werden Ähnlichkeitsalgorithmen vorgestellt, die für die Ermittlung der Ähnlichkeit zweier Zeichenketten die Funktionsweise von Zeichen-basierten und Token-basierten Algorithmen kombinieren.

In dieser Abschlussarbeit werden folgende hybride Algorithmen näher betrachtet:

- Monge-Elkan-Ähnlichkeit
- Soft-TF-IDF-Maß

2.3.3.1 Monge-Elkan-Ähnlichkeit

Funktionsweise. Die Ähnlichkeit nach Monge-Elkan [MoE96] wird ermittelt, indem zunächst zwei zu vergleichende Zeichenketten in einzelne Token zerlegt werden. Anschließend wird zu jedem Token aus der ersten Zeichenkette ein Token aus der zweiten Zeichenkette ermittelt, welches die höchste Ähnlichkeit aufweist. Zur Ermittlung der Ähnlichkeit der Token kann eine Zeichen-basierte Ähnlichkeitsfunktion (z.B. eine in Kapitel 2.3.1 vorgestellte Funktion) verwendet werden. Abschließend werden die maximalen Ähnlichkeiten der einzelnen Token aufsummiert und mit der Anzahl der Token aus der ersten Zeichenkette normalisiert.

Definition. Sind die beiden zu vergleichenden Zeichenketten X und Y sowie die daraus resultierenden Token-Mengen t_X und t_Y gegeben, ist die Monge-Elkan-Ähnlichkeit definiert durch:

$$MongeElkan(t_X, t_Y) = \frac{1}{|t_X|} \sum_{i=1}^{|t_X|} \max_{1 \leq j \leq |t_Y|} TokenSimilarity(t_{X_i}, t_{Y_j})$$

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *Herr Alex Mustermann* und *Frau Alex Musterfrau* bzw. die daraus resultierenden Token-Mengen *{Herr, Alex, Mustermann}* und *{Frau, Alex, Musterfrau}*. Zur Ermittlung der Ähnlichkeit der einzelnen Token wird die Levenshtein-Distanz verwendet und das resultierende Distanzmaß, wie in Kapitel 2.3 beschrieben, in ein Ähnlichkeitsmaß umgerechnet:

$$MongeElkan(\{Herr, Alex, Mustermann\}, \{Frau, Alex, Musterfrau\}) = \frac{0{,}3 + 1 + 0{,}6}{3} \approx 0{,}63$$

2.3.3.2 Soft-TF-IDF-Maß

Funktionsweise. Identisch zur Monge-Elkan-Ähnlichkeit wird beim Soft-TF-IDF-Maß ebenfalls eine Zeichen-basierte Ähnlichkeitsfunktion zur Ermittlung der Ähnlichkeit der Token definiert. Allerdings wird beim Soft-TF-IDF-Maß zusätzlich ein Schwellenwert eingeführt, sodass nur Token berücksichtigt

werden, deren Ähnlichkeit diesen Schwellenwert übersteigen. Darüber hinaus werden nur Token der ersten Zeichenkette berücksichtigt, die auch in der zweiten Zeichenkette vorhanden sind [NaH10].

Definition. Sind die beiden zu vergleichenden Zeichenketten X und Y, die daraus resultierenden Token-Mengen t_X und t_Y sowie ein Schwellenwert α gegeben, ist das Soft-TF-IDF-Maß definiert durch:

$$Close(\alpha, t_X, t_Y) = \{t_i | t_i \in t_X \ \wedge \ \exists t_j \in t_Y : TokenSimilarity(t_i, t_j) > \propto\}$$

$$SoftTFIDF(t_X, t_Y) = \sum_{t_i \in Close(\propto, t_X, t_Y)} \left(\frac{TFIDF_{t_i}}{\|V_X\|} \times \frac{TFIDF_{t_j}}{\|V_Y\|} \times \max_{1 \leq j \leq |t_Y|} TokenSimilarity(t_{X_i}, t_{Y_j}) \right)$$

V_X und V_Y bezeichnen hierbei die Vektoren, die die TF-IDF-Gewichtung der Ausdrücke aus X und Y enthalten.

Einfaches Beispiel. Gegeben sind die beiden Zeichenketten *Herr Alex Mustermann* und *Frau Alex Musterfrau* sowie die beiden Annahmen, dass der gegebene Schwellenwert zur Menge $Close(\alpha, t_X, t_Y) = \{Alex\}$ führt und die beiden oben genannten Zeichenketten die beiden Vektoren $V_X = \{0.05, 0.6, 0.5, 0, 0, 0\}$ und $V_Y = \{0, 0, 0, 0.05, 0.6, 0.5\}$ ergeben.

$$SoftTFIDF(t_X, t_Y) = \frac{0{,}6}{\sqrt{0{,}05^2 + 0{,}6^2 + 0{,}5^2 + 0^2 + 0^2 + 0^2}} \times 1 \ \approx 0{,}77$$

2.3.4 Phonetisch

Bei der phonetischen Suche handelt es sich um ein Suchverfahren zur Ermittlung von ähnlich klingenden Wörtern in der menschlichen Sprache. Dabei werden die einzelnen Wörter in so genannte Phoneme zerlegt und dadurch für jedes Wort eine phonetische, also klangliche Repräsentation erstellt. Anschließend werden diese phonetischen Repräsentationen miteinander verglichen. Ähnlich klingende Wörter (z.B. Maier und Meier) weisen dabei die gleiche phonetische Repräsentation auf.

In dieser Abschlussarbeit werden folgende phonetische Algorithmen näher betrachtet:

- Soundex
- Kölner Phonetik
- Metaphone und Double-Metaphone

2.3.4.1 Soundex

Funktionsweise. Der Soundex-Algorithmus wurde von Robert Russell und Margaret Odell entwickelt und im Jahr 1918 patentiert (US1261167 und US1435663) [Ode56]. Ziel war es, die Familiennamen bei der Volkszählung in den USA zu indizieren. Daher ist Soundex vor allem auf die englische Aussprache optimiert, leistet jedoch auch häufig gute Ergebnisse bei der deutschen Sprache.

Definition. Der Soundex-Code eines Wortes setzt sich zusammen aus dem Anfangsbuchstaben des Wortes gefolgt von einem Bindestrich und drei Ziffern. Diese Ziffern repräsentieren die Konsonanten des Wortes (Kodierung siehe nachfolgende Tabelle). Können - aufgrund der Anzahl der Buchstaben - mehr als drei Ziffern erzeugt werden, wird nach der dritten Ziffer abgebrochen. Hat das Wort zu wenig Buchstaben, werden die fehlenden Ziffern mit Nullen aufgefüllt.

Ziffer	Repräsentierte Buchstaben
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Tabelle 2: Die Buchstabencodes des Soundex-Algorithmus

Die Vokale (A, E, I, O, U) und die drei Konsonanten H, W und Y werden ignoriert, außer es handelt sich um den ersten Buchstaben. In der deutschen Sprache werden zudem noch folgende Regeln festgelegt: die Umlaute (Ä, Ö, Ü) werden ignoriert, dass ß wird zum normalen S.

Zudem werden aufeinanderfolgende gleiche Ziffern zu einer Ziffer reduziert und ggf. fehlende Ziffern mit Nullen aufgefüllt. (*abfx* würde beispielsweise den Code *A-112* ergeben und wird zu *A-120*).

Einfaches Beispiel. Das Wort *Wikipedia* wird mit dem Soundex-Code *W-213* kodiert. Der erste Buchstabe ist das W gefolgt von einem Bindestrich und den Ziffern 2 (für K), 1 (für P) und 3 (für D). Die Vokale I, E und A werden ignoriert. Der asiatische Name *Lee* wird mit dem Soundex-Code *L-000* kodiert, da der Vokal E ignoriert wird und die fehlenden Ziffern mit Nullen aufgefüllt werden.

2.3.4.2 Kölner Phonetik

Funktionsweise. Die Kölner Phonetik [Pos69] (häufig auch Kölner Verfahren genannt) wurde 1969 von dem deutschen Verfassungsschutzmitarbeiter Hans Joachim Postel veröffentlicht. Ziel ist es, Wörtern mit ähnlichem Sprachklang den gleichen phonetischen Code (Ziffernfolge) zuzuordnen. Die Kölner Phonetik ist vergleichbar mit dem Soundex-Algorithmus, wurde jedoch wesentlich besser auf die deutsche Sprache abgestimmt.

Definition. Um den phonetischen Code eines Wortes zu ermitteln, wird jeder Buchstaben mit Hilfe der unten stehenden Tabelle auf eine Ziffer abgebildet. Dabei wird jeder Buchstabe entweder separat oder in einem Kontext (benachbarter Buchstabe oder Wortanfang (Anlaut)) betrachtet.

Ziffer	Repräsentierte Buchstaben	Kontext
0	A, E, I, J, O, U, Y	
-	H	
1	B	
	P	nicht vor H
2	D, T	nicht vor C, S, Z
3	F, V, W	
	P	vor H
4	G, K, Q	
	C	im Anlaut vor A, H, K, L, O, Q, R, U, X
		vor A, H, K, O, Q, U, X außer nach S, Z
48	X	nicht nach C, K, Q
5	L	
6	M, N	
7	R	
8	S, Z	
	C	nach S, Z
		im Anlaut außer vor A, H, K, L, O, Q, R, U, X
		nicht vor A, H, K, O, Q, U, X
	D, T	vor C, S, Z
	X	nach C, K, Q

Tabelle 3: Die Buchstabencodes der Kölner Phonetik

Umlaute (Ä, Ö, Ü) werden meist mit der Ziffer Null, dass ß mit der Ziffer Acht kodiert. Alle anderen Zeichen (z.B. Bindestriche, Sonderzeichen, usw.) werden ignoriert. Groß- und Kleinbuchstaben werden analog betrachtet. Im Gegensatz zum Soundex ist die Länge des Codes nicht beschränkt.

Die Umwandlung eines Wortes in den phonetischen Code erfolgt in drei Schritten:

1. Das Wort wird - entsprechend der Tabelle - buchstabenweise von links nach rechts kodiert
2. Alle mehrfach nebeneinander vorkommenden Ziffern werden entfernt
3. Alle Nullen (außer am Anfang) werden entfernt

Einfaches Beispiel. Das Wort *Wikipedia* wird in der Kölner Phonetik mit dem Code *3412* kodiert.

1. Kodierung nach der Umwandlungstabelle: *304010200*
2. Mehrfach nebeneinander vorkommende Ziffern entfernen: *304010200*
3. Alle Nullen (außer am Anfang) entfernen: *3412*

2.3.4.3 Metaphone, Double-Metaphone und Metaphone 3

Funktionsweise. Der phonetische Algorithmus Metaphone [Phi90] wurde im Jahr 1990 von Lawrence Philips veröffentlicht. Ziel war es, die sehr grobe Unterscheidbarkeit des Soundex-Algorithmus massiv zu verbessern. Dies wird vor allem dadurch erreicht, dass die Kodierung der Wörter nicht zeichenweise erfolgt, sondern ganze Zeichenketten kodiert werden. Darüber hinaus hängt die Kodierung auch von der Position der Zeichenketten innerhalb des Wortes ab. Ähnlich zur Kölner Phonetik haben die vom Metaphone-Algorithmus erzeugten phonetischen Codes keine feste Länge. Metaphone unterstützt dabei nur die englische Sprache. Im Jahr 2000 veröffentlichte Philips in der Juni-Ausgabe des *C/C++ Users Journal* eine verbesserte Version des Algorithmus unter dem Namen Double-Metaphone. Dieser enthält eine Vielzahl von fundamentalen Design-Verbesserungen und unterstützt darüber hinaus neben Englisch viele weitere Sprachen, wie beispielsweise Deutsch, Französisch, Italienisch, Spanisch oder Chinesisch. Im Oktober 2009 wurde eine weitere verbesserte Version unter dem Namen Metaphone 3 von Philips veröffentlicht.

Definition. Die Beschreibung der Regeln der ursprünglichen Metaphone-Implementierung zur Generierung eines phonetischen Codes würden den Rahmen dieser Masterarbeit sprengen. Deshalb wird an dieser Stelle auf die einschlägige Literatur zu diesem Thema verwiesen. Der C++ SourceCode einer leicht modifizierten Version des Double-Metaphone ist unter [ASP] abrufbar. Der SourceCode von Metaphone 3 kann für diverse Programmiersprachen unter [AMO] kostenpflichtig erworben werden.

Einfaches Beispiel. Die Zeichenketten *Frau Alex Mustermann* und *Herr Alex Mustermann* ergeben die Metaphone-Codes *FRLKSMSTRMN* und *HRLKSMSTRMN*. Es ist zu erkennen, dass nur die ersten beiden Buchstaben (für Frau, bzw. Herr) voneinander abweichen.

2.3.5 Domänenspezifisch

2.3.5.1 Datumswerte

Beim Vergleich von Datumswerten wird häufig die Differenz zwischen zwei gegebenen Werten ermittelt. Anschließend wird diese Differenz in eine für den Kontext sinnvolle Einheit (z.B. Minuten, Stunden, Tage oder Jahre) umgewandelt und als numerischer Wert betrachtet bzw. als solcher weiterverarbeitet. Um beispielsweise die älteste Person in einer (Kunden-)Datenbank zu bestimmen, wird zunächst das Alter aller Personen in dieser Datenbank ermittelt. Hierfür wird die Differenz des Geburtsdatums dieser Personen zum heutigen Datum errechnet, in Jahren angegeben und beispielsweise absteigend sortiert ausgegeben.

2.3.5.2 Muster

Um Muster in Zeichenketten festzustellen, kommen häufig reguläre Ausdrücke zum Einsatz. Neben der Implementierung in diversen Programmiersprachen werden reguläre Ausdrücke auch als Filterkriterien in Textsuchen eingesetzt. So können beispielsweise aus einem Text alle Wörter herausgesucht werden, die mit einem bestimmten Buchstaben beginnen und/oder enden ohne dabei dazwischenliegende Buchstaben explizit angeben zu müssen. Auch zum Überprüfen von Daten eignen sich reguläre Ausdrücke. So können beispielsweise fünfstellige Postleitzahlen mit einem optionalen "D-"-Präfix für Deutschland mit Hilfe des Ausdrucks "^(D-)?[0-9]{5}" validiert werden. Für weitere Informationen sei auf die einschlägige Literatur zu diesem Thema, z.B. [Fri07], verwiesen.

2.3.5.3 Numerische Werte

Bei numerischen Werten muss zwischen kardinal skalierten Werten (stetige Werte nach objektiver, geordneter Werteskala mit festem Nullpunkt; z.B. Preise) und nominal skalierten, nummerischen Werten (diskrete Werte nach ungeordneter Werteskala ohne festem Nullpunkt; z.B. Umsatzsteuernummern oder Telefon- und Faxnummern) unterschieden werden. Bei ersterem lässt sich die Ähnlichkeit zwischen zwei Werten einfach ermitteln: Je geringer der Abstand, desto ähnlicher. Bei letzterem eigenen sich beispielsweise ontologie-basierte Ähnlichkeitsmetriken. Da die Auswahl von Ähnlichkeitsmetriken für numerische Werte sehr eingeschränkt ist, werden diese häufig in Text-Werte umgewandelt und die bereits beschriebenen Ähnlichkeitsalgorithmen (z.B. Hamming-Abstand, Levenshtein-Distanz, usw.) angewandt [ABEM10].

2.3.5.4 Ontologien

Da nominal skalierte Werte (z.B. Farben, Tierarten, usw.) keine lexikalische oder syntaktische Ähnlichkeit aufweisen, lässt sich deren Ähnlichkeit nicht mit den bereits vorgestellten Metriken ermitteln. Daher kommen hierzu beispielsweise ontologie-basierte Ähnlichkeitsmetriken [KeM02] zum Einsatz. Diese erstellen ein domänenspezifisches Begriffsmodell mit logischen Beziehungen.

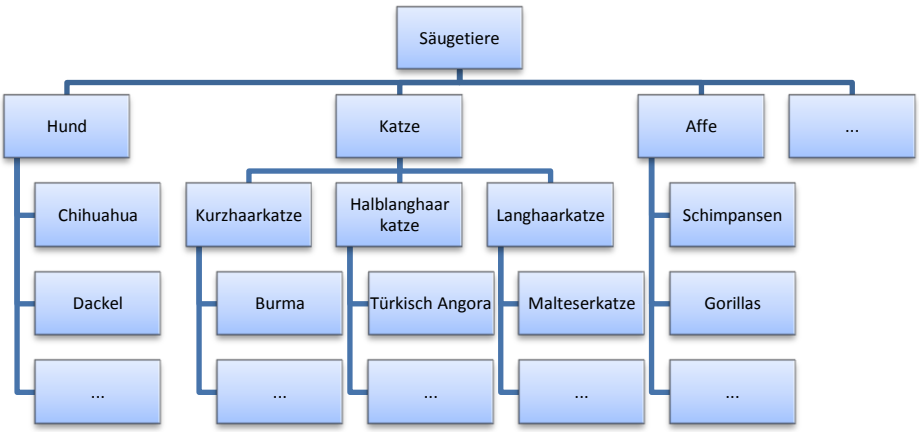

Abbildung 2: Beispiel für eine mögliche Ontologie

2.4 Evaluation und Qualitätsmessung

Um die Genauigkeit der verschiedenen Algorithmen zur Ähnlichkeitsbestimmung untereinander vergleichen und die Qualität der Duplikaterkennung bewerten zu können, werden Kennzahlen benötigt. Im Forschungsbereich des Information Retrieval kommen hierfür vor allem drei Kennzahlen zum Einsatz [NaH10]: *Precision (dt. Spezifität)*, *Recall (dt. Sensitivität)* und das harmonische Mittel aus beiden, der so genannte *F-Score (dt. F-Maß)*.

Bei der Duplikaterkennung wird von zwei Gruppen ausgegangen:

- ***Duplikate***: Datensatzpaare, die zusammengehörten
- ***Nicht-Duplikate***: Datensatzpaare, die nicht zusammengehörten

Das Ergebnis der Duplikaterkennung, also die Zuordnung eines Datensatzes zu einem anderen, lässt sich wie folgt klassifizieren:

- ***True-Positives (TP):*** Datensatzpaar wurde von der Duplikaterkennung als Duplikat ausgewiesen und ist tatsächlich ein Duplikat
- ***False-Positives (FP):*** Datensatzpaar wurde von der Duplikaterkennung fälschlicherweise als Duplikat ausgewiesen und ist tatsächlich ein Nicht-Duplikat
- ***False-Negatives (FN):*** Datensatzpaar wurde von der Duplikaterkennung fälschlicherweise als Nicht-Duplikat ausgewiesen und ist tatsächlich ein Duplikat
- ***True-Negatives (TN):*** Datensatzpaar wurde von der Duplikaterkennung als Nicht-Duplikat ausgewiesen und ist tatsächlich ein Nicht-Duplikat

Die oben erläuterten Klassifikationen werden nachfolgend nochmals grafisch dargestellt.

		Realität / Expertenanalyse	
		Duplikat	**Nicht-Duplikat**
Duplikaterkennung	**Duplikat**	True-Positives (TP)	False-Positives (FP)
	Nicht-Duplikat	False-Negatives (FN)	True-Negatives (TN)

Tabelle 4: Klassifikationsmatrix der Duplikaterkennung

Die Kennzahl *Precision (dt. Spezifität)* gibt an, wie hoch der Anteil der korrekt erkannten Duplikate im Verhältnis zu allen mit dem Verfahren erkannten Duplikate ist:

$$Precision\ (dt. Spezifität) = \frac{True - Positives\ (TP)}{True - Positives\ (TP)\ +\ False - Positves\ (FP)}$$

Die Kennzahl *Recall (dt. Sensitivität)* gibt an, wie hoch der Anteil der korrekt erkannten Duplikate im Verhältnis zu allen korrekten Duplikaten ist:

$$Recall\ (dt. Sensitivität) = \frac{True - Positives\ (TP)}{True - Positives\ (TP)\ +\ False - Negatives\ (FN)}$$

"Eine hohe Spezifität bedeutet, dass zwei Datensätze nur bei sehr großer Ähnlichkeit als Duplikate ausgewiesen werden. Dies kann zur Folge haben, dass echte Duplikate nicht als solche erkannt werden, da sie nicht „ähnlich“ genug sind. Bei einer hohen Sensitivität werden relativ viele Datensätze als Duplikate erkannt, auch solche, die möglicherweise gar keine sind." [ABEM10]

Um eine zusammenfassende Kennzahl zu erhalten, können Precision und Recall zu einem harmonischen Mittel kombiniert werden. Diese Kennzahl wird als *F-Score (dt. F-Maß)* bezeichnet und ist wie folgt definiert:

$$F - Score\ (dt. F - Maß) = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Nachfolgend wird nun ein Beispiele aufgezeigt, um die Qualitätsmessung der Duplikaterkennung und den Einsatz der Kennzahlen exemplarisch zu verdeutlichen:

Gegeben ist ein Testdatenbestand mit 1.000 Datensätzen. Von Experten wurden 200 Datensätze als Duplikate klassifiziert. Die Duplikaterkennung hat 180 Duplikate richtig erkannt, 30 fälschlicherweise als Duplikate gekennzeichnet und 20 Duplikate nicht gefunden. Hieraus ergibt sich:

$$Precision\ (dt. Spezifität) = \frac{180}{180 + 30} \approx 85\%$$

$$Recall\ (dt. Sensitivität) = \frac{180}{180 + 20} = 90\%$$

$$F - Score\ (dt. F - Maß) = \frac{2 \times 85\% \times 90\%}{85\% + 90\%} \approx 87\%$$

Bewertung:

- 85% der mit dem eingesetzten Verfahren gefundenen Duplikate waren korrekte Duplikate
- 90% aller korrekten, von den Experten klassifizierten Duplikate wurden gefunden

3 Konzeptionelle Betrachtung

3.1 Problemanalyse

Die Verwertungsgesellschaft GEMA erhält Informationen über die Nutzung eines Musikstücks in elektronischer Form - meist in vereinheitlichten XML- oder CSV-Dateien. Um diese gemeldeten Musiknutzungen abrechnen zu können, müssen die verwendeten Musikstücke eindeutig einem Urheber zugeordnet werden. Diese Zuordnung erfolgt über den Abgleich der gemeldeten Musikstücke mit einer internen Datenbank der GEMA. Für die Ermittlung des eindeutigen Urhebers werden mehrere Attribute eines Musikstückes (u.a. beispielsweise der Titel des Songs sowie die Namen der Artisten und des Albums) verwendet.

Zunächst wird versucht, die gemeldeten Musiknutzungen in einem automatisierten Prozess einem eindeutigen Urheber zuzuordnen. Da der Prozess sehr umfangreich und vielschichtig ist, wird an dieser Stelle nicht näher auf die einzelnen Teilschritte eingegangen. Die Details dieser Teilschritte sind für das allgemeine Verständnis der Zuordnung von Musiknutzungen sowie für die Problemanalyse ohnehin nicht notwendig. Wichtig ist an dieser Stelle nur, dass für den Abgleich der Attribute die Levenshtein-Distanz als Ähnlichkeitsmetrik verwendet wird. Ursprünglich wurde die Vorverarbeitung sowie die Ähnlichkeitsbestimmung mit Hilfe von selbst entwickelten Algorithmen durchgeführt. Mittlerweile kommt hierfür das in Java implementierte OpenSource Such-Framework Apache Lucene [Lucene] zum Einsatz. Dieses verwendet jedoch intern - abhängig von der individuellen Konfiguration - ebenfalls die Levenshtein-Distanz. Im Idealfall kann ein Musikstück während dem automatischen Matching-Prozess identifiziert und einem eindeutigen Urheber zugeordnet werden. Kann das Musikstück jedoch nicht eindeutig identifiziert werden (z.B. weil mehrere gleichwertige Zuordnungs-Möglichkeiten gefunden wurden, die Attribute zu stark voneinander abweichen, das Musikstück noch nicht in der internen Datenbank eingepflegt wurde, o.ä.), folgt in einem darauffolgenden Prozess eine manuelle Zuordnung. Hierbei erhält ein Sachbearbeiter neben den Informationen des nicht zuordenbaren Musikstückes eine Liste mit mehreren Zuordnungsvorschlägen. Diese Vorschläge werden anschließend einzeln per Hand geprüft und ggf. eine manuelle Zuordnung durchgeführt. Eignet sich keiner der Vorschläge, z.B. weil das Musikstück noch nicht in der internen Datenbank eingepflegt ist, wird durch den Sachbearbeiter eine Internetrecherche durchgeführt und die gefundenen Informationen ggf. in die interne Datenbank eingetragen - ebenfalls von Hand. Um die nicht automatisch zuordenbaren Musikstücke abzuarbeiten, sind laut GEMA derzeit mehrere Sachbearbeiter in Vollzeit angestellt. Die nachfolgende Abbildung zeigt den beschriebenen Ablauf der Zuordnung von Musiknutzungen nochmals schematisch (gestrichelte Linien: Datenabruf; durchgezogene Linien: Zuordnungs-Ablauf).

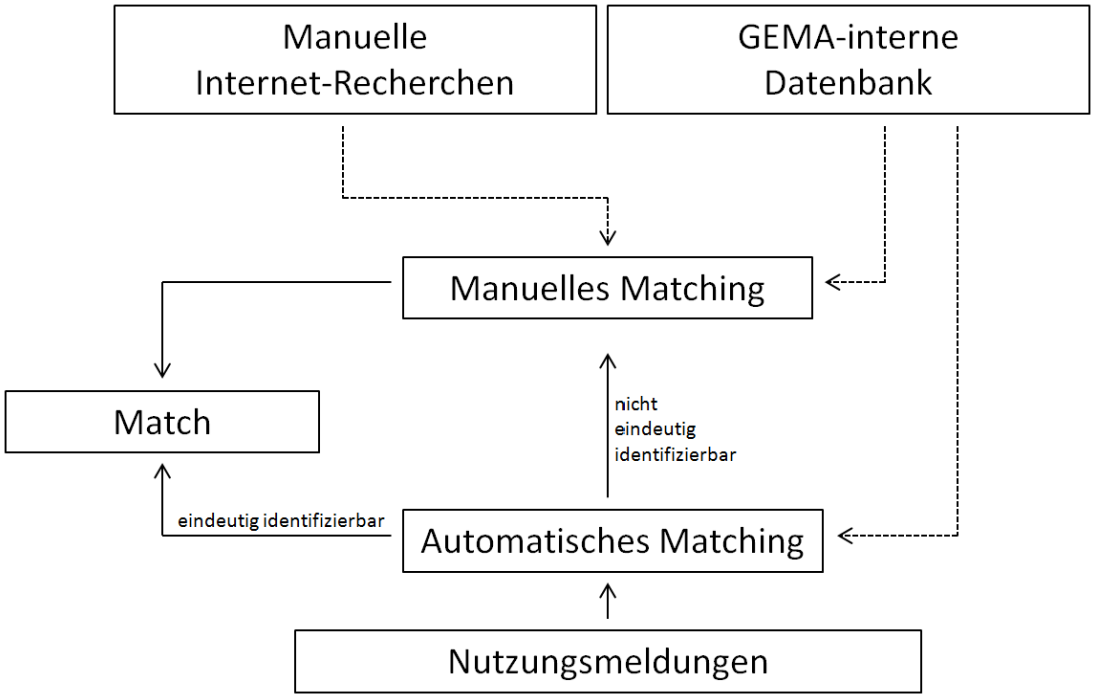

Abbildung 3: Zuordnung von Musiknutzungen (aktuelles Vorgehen)

Da aufgrund der aktuell stark steigenden Musiknutzung die Menge der zu verarbeitenden Nutzungsmeldungen in Zukunft zunehmen wird und darüber hinaus die aus dem Online-Bereich (z.B. YouTube) gemeldeten Daten wesentlich heterogen (häufig verschiedene Schreibweisen im Titel oder Album) sind, ist davon auszugehen, dass auch die Menge der nicht automatisch zuordenbaren Musikstücke ansteigen wird. Dies könnte zu einem nicht tragbaren personellen Aufwand führen. Zudem werden sich die notwendigen Rechenkapazitäten für die aufwändigen Ähnlichkeits-berechnungen entsprechend erhöhen.

Aus diesem Grund soll ein neues Vorgehen zur automatischen und manuellen Zuordnung von Musiknutzungen erarbeitet werden. In den nachfolgenden beiden Kapiteln werden hierfür die Anforderungen analysiert und definiert sowie mögliche Lösungsansätze entwickelt.

3.2 Anforderungsanalyse und -definition

Nachfolgend werden die Anforderungen an das zu konzipierende und prototypisch zu entwickelnde Empfehlungssystem zur urheberrechtlichen Zuordnung von Musiknutzungen dargestellt:

- **BigData-Umfeld / horizontal skalierbar**

  Da die Anzahl der zu verarbeitenden Nutzungsmeldungen kontinuierlich steigt, soll das Empfehlungssystem mit Hilfe von BigData-Technologien umgesetzt werden. Auch eine horizontale Skalierung des Systems soll möglich sein.

- **Java- und Linux-basiert**
  Um eine leichte Integration in die bestehende Systemlandschaft zu ermöglichen sowie die Kosten möglichst gering zu halten, sollte das Empfehlungssystem in Java implementiert sowie unter Linux betrieben werden können.
- **OpenSource**
  Sowohl bei der Auswahl der Technologien als auch bei der Nutzung von externen Datenquellen soll soweit möglich auf eine freie Lizenz geachtet bzw. diese bevorzugt werden.
- **MDM-Vorgaben berücksichtigen**
  Die in der Dokumentation des Master Data Management (MDM) der GEMA beschriebenen allgemeinen Regeln zur Bestimmung von Produktdatendubletten sollen bei der Implementierung des Prototypen berücksichtigt werden.
- **Leichte Erweiterbarkeit sowie Anpassbarkeit**
  Bei der Konzeption des Empfehlungssystems sowie bei der Implementierung des Prototypen soll darauf geachtet werden, dass das System möglichst offen gestaltet wird, um eine spätere Erweiterung (beispielsweise um zusätzliche Ähnlichkeitsalgorithmen) einfach zu halten. Auch die Anpassbarkeit von gegebenenfalls eingesetzter Drittanbieter-Software an individuelle Anforderungen und Gegebenheiten soll möglich sein.

Ziele des zu konzipierenden und prototypisch zu entwickelnden Empfehlungssystems sind:

- **Optimierung hinsichtlich der Zuordnungs-Qualität**
  Bei der Konzeption des Systems sowie bei der Auswahl und Implementierung von Ähnlichkeitsalgorithmen bzw. Such-Frameworks ist eine hohe Zuordnungs-Qualität anzustreben. Eine hohe Qualität bei den automatischen Zuordnungen führt zwangsläufig zu einer geringeren Anzahl an manuell zu bearbeitenden Nutzungsmeldungen und somit zu einem geringeren personellen Aufwand. Darüber hinaus soll die notwendige Bearbeitungszeit für die manuelle Zuordnung von Nutzungsmeldungen durch wenige aber dafür qualitativ hochwertige Zuordnungsvorschläge reduziert werden.
- **Optimierung hinsichtlich der Performance**
  Da die Anzahl der zu verarbeitenden Nutzungsmeldungen in Zukunft steigen wird, ist bei der Konzeption sowie bei der Auswahl und Implementierung von Ähnlichkeitsalgorithmen bzw. Such-Frameworks auf eine kurze Ausführungszeit zu achten.

3.3 Lösungsansatz

Im nachfolgenden Abschnitt wird das Konzept des zu entwickelnden Empfehlungssystems skizziert sowie mögliche Lösungs- bzw. Umsetzungsansätze näher betrachtet.

3.3.1 Allgemeine Konzeption

Wie bereits in Kapitel 3.1 kurz erwähnt, ist das Ziel dieser Abschlussarbeit ein neues Vorgehen zur automatischen und manuellen Zuordnung von Musiknutzungen zu erarbeiten. Dies soll in Form einer Konzeption eines BigData-gestützten Empfehlungssystems zur urheberrechtlichen Zuordnung von Musiknutzungen mit Betrachtung möglicher Lösungs- bzw. Umsetzungsansätze erfolgen. Durch eine anschließende prototypische Implementierung soll darüber hinaus die Umsetzbarkeit und Leistungsfähigkeit des erarbeiteten Konzepts erprobt werden.

Die nachfolgende Abbildung visualisiert das erarbeitete Konzept für ein mögliches neues Vorgehen zur automatischen und manuellen Zuordnung von Musiknutzungen schematisch (gestrichelte Linien: Datenabruf; durchgezogene Linien: Zuordnungs-Ablauf). Die erarbeiteten Bestandteile des neuen Vorgehens sind in rot hervorgehoben.

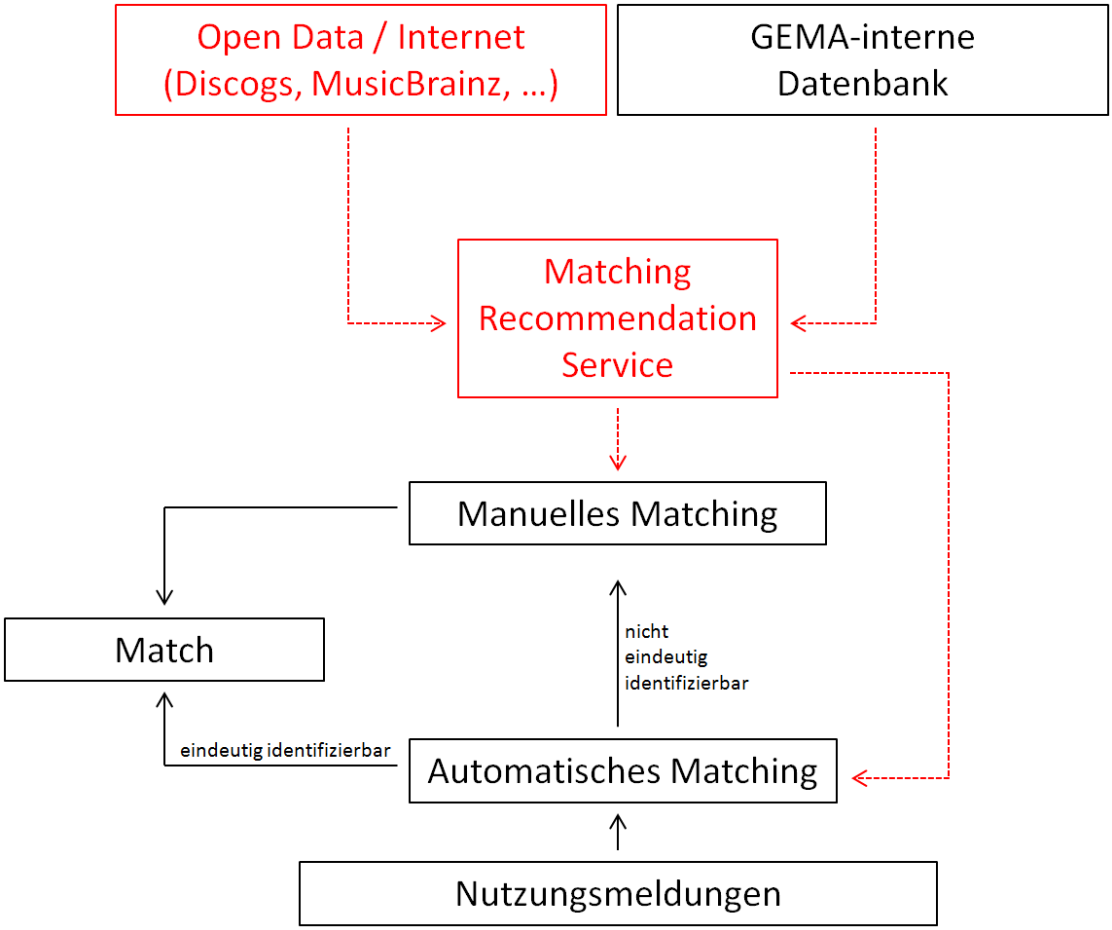

Abbildung 4: Zuordnung von Musiknutzungen (mögliches neues Vorgehen)

Ziel ist es, durch die Einführung eines BigData-gestützten Empfehlungssystems sowohl das automatische als auch das manuelle Matching in den beiden Bereichen *Zuordnungs-Qualität* und *Performance* zu optimieren. Die Datenbasis des Empfehlungssystems kann dabei aus GEMA-internen Datenbeständen sowie zusätzlich oder alternativ aus öffentlich zugänglichen und gemeinfreien Datenquellen (z.B. Discogs oder MusicBrainz, Auswahl der Datenquellen siehe Kapitel 4.3) aufgebaut werden. Auch die Nutzung von weiteren Informationsquellen, wie beispielsweise Datenbanken mit Songtexten oder Konzertinformationen, können zu einem späteren Zeitpunkt als sinnvoll oder notwendig erachtet werden und an das Empfehlungssystem angebunden werden. Durch den Einsatz einer möglichst großen und breiten Datenbasis könnte zum einen die Zuordnungs-Qualität deutlich verbessert sowie zum anderen der notwendige Zeitaufwand für manuelle Internet-Recherchen erheblich minimiert werden. Darüber hinaus können die Informationen aus den diversen Quellen zusammengefasst über eine zentrale Benutzeroberfläche abgerufen und dargestellt werden. Dadurch ist ein zeitintensiver Wechsel zu anderen Anwendungen - z.B. einem Browser für die manuelle Internet-Recherche - nicht mehr notwendig. Um die zusätzlichen Daten effizient verarbeiten zu können ist ein BigData-gestützter, horizontal skalierbarer Ansatz zu präferieren.

In einem ersten Schritt könnte die konzipierte Lösung zunächst als Empfehlungssystem implementiert werden. Dies bedeutet, dass sowohl für das automatische als auch für das manuelle Matching nur Vorschläge geliefert werden, jedoch keine Zuordnungen stattfinden. Die Zuordnung von Musiknutzungen erfolgt weiterhin durch die bestehenden Systeme. Dadurch ist zum Beispiel ein Vergleich der beiden Systeme sowie individuelle Anpassung am neuen System möglich. In einem zweiten Schritt könnten die Ergebnisse des Empfehlungssystems als Grundlage für die bestehenden Systeme genutzt werden. So könnte beispielsweise die automatische Zuordnung die Ergebnisse des Empfehlungssystems als Vorauswahl nutzen. Bei der manuellen Zuordnung könnten die Vorschläge aus beiden Systemen als Basis verwendet werden. In einem dritten Schritt könnten letztendlich die bestehenden Systeme vollständig durch das neu konzipierte System ersetzt werden.

In den nachfolgenden Abschnitten werden mehrere mögliche Lösungs- bzw. Umsetzungsansätze für das oben beschriebene Konzept kurz erläutert.

3.3.2 Lösungs-/Umsetzungsansatz 1

Ein möglicher Lösungs-/Umsetzungsansatz des BigData-gestützten Empfehlungssystems ist die Implementierung von verschiedenen Ähnlichkeitsmetriken auf Basis von MapReduce-Algorithmen innerhalb von Apache Hadoop. Einer der größten Vorteile dieses Ansatzes ist die sehr hohe Flexibilität: Eine individuelle, auf die jeweiligen Anforderungen zugeschnittene Entwicklung des Empfehlungssystems sind ebenso möglich wie spätere Anpassungen und spezifische Erweiterungen.

Auch die Berücksichtigung bzw. Umsetzung der einzelnen MDM-Vorgaben bei der Implementierung sollte dadurch ohne größere Probleme möglich sein. Die Speicherung der Referenzdaten kann sowohl in einer HBase-Datenbank als auch in Flat-Files (z.B. CSV-Dateien), die direkt im HDFS abgelegt werden, erfolgen. Die weiteren Anforderungen, u.a. OpenSource, Java- und Linux-basiert sowie die horizontale Skalierbarkeit des Systems können durch Apache Hadoop und Apache HBase erfüllt werden. Nachteilig ist an diesem Umsetzungsansatz lediglich die Tatsache, dass sowohl das Empfehlungssystem als auch die einzelnen Ähnlichkeitsmetriken auf Basis von MapReduce vollständig selbst entwickelt werden müssen. Während der Recherche konnte jedoch schon eine vielversprechende, Java-basierte OpenSource-Bibiothek mit diversen, fertig implementierten Ähnlichkeitsmetriken gefunden werden (siehe Kapitel 4.1.3). Diese lässt sich ohne weiteres innerhalb von MapReduce-Algorithmen nutzen. Dies sowie die sehr hohe Flexibilität favorisieren diesen Ansatz, sowohl für die Implementierung eines Prototypen im Rahmen dieser Abschlussarbeit als auch für eine spätere produktive Nutzung.

3.3.3 Lösungs-/Umsetzungsansatz 2

Ein weiterer möglicher Umsetzungsansatz ist die Nutzung der OpenSource Such-Engine Apache Blur [Blur], welche auf Apache Hadoop aufsetzt und somit viele gestellte Anforderungen, wie horizontale Skalierbarkeit sowie Java- und Linux-basiert bereits erfüllt. Blur ist aktuell noch ein Inkubator-Projekt der Apache Foundation. Einer der größten Vorteile dieses Ansatzes ist die interne Nutzung des weit verbreiteten, freien Such-Frameworks Lucene. Dadurch kann auf eine erprobte Indizierungs- und Such-Technologie aufgebaut werden und es ist nicht notwendig, eigene Ähnlichkeitsmetriken vollständig selbst zu implementieren. Dies ist allerdings - zumindest im Vergleich zum Ansatz 1 - auch ein Nachteil, da dadurch nur die vorgegebenen Metriken genutzt werden können. Dies schränkt die Flexibilität sowie die Anforderung nach leichter Erweiterbar- und Anpassbarkeit ein. Apache Blur wurde zudem im Kontext des Forschungsprojekts MPI bereits im Jahr 2015 durch eine Masterarbeit von Herrn Angermeier mit dem Titel "Patternmatching in der Musikindustrie" ausführlich betrachtet. Daher wird Ansatz 2 in dieser Abschlussarbeit nicht näher untersucht und für weiterführende Informationen auf die erwähnte Masterarbeit verwiesen.

3.3.4 Lösungs-/Umsetzungsansatz 3

Ein dritter Umsetzungsansatz ist der Einsatz der freien, in Java geschriebenen Suchmaschine Elasticsearch [Elastic]. Elasticsearch zählt zu den am weitesten verbreiteten Suchservern und baut ebenfalls auf dem freien Such-Framework Lucene auf. Dadurch ergeben sich die gleichen Vor- bzw. Nachteile wie bei Ansatz 2: Die zugrundeliegende Indizierungs- und Such-Technologie ist bewährt; die Entwicklung eigener Ähnlichkeitsmetriken somit nicht notwendig, allerdings auch nicht möglich.

Dies führt zu Einschränkungen hinsichtlich der Flexibilität bezüglich individueller Erweiterungen und Anpassungen. Elasticsearch ermöglicht den Betrieb der Suchmaschine sowohl auf einem einzelnen Server als auch in einem Computerverbund. Dadurch ist eine horizontale Skalierung und somit die Umsetzung von Hochverfügbarkeit und Lastverteilung jederzeit möglich. Kann auf die individuelle Erweiterung um zusätzliche Ähnlichkeitsmetriken verzichtet werden, ist Elasticsearch ein alternativer Lösungs-/Umsetzungsansatz für das in Kapitel 3.3.1 beschriebene Konzept. Im Gegensatz zu Ansatz 2 (mit Apache Blur) kann beim Einsatz von Elasticsearch auf die Einrichtung und den Betrieb eines Hadoop-Clusters verzichtet werden. Da in dieser Abschlussarbeit neben der Konzeption und prototyptischen Implementierung eines Empfehlungssystems zur urheberrechtlichen Zuordnung von Musiknutzungen auch verschiedene Ähnlichkeitsmetriken implementiert und gegenüber gestellt werden sollen, wird der Lösungsansatz mit Elasticsearch an dieser Stelle nicht weiter verfolgt. Die Umsetzung des beschriebenen Konzepts auf Basis von Elasticsearch sowie ein anschließender Vergleich mit den Ergebnissen dieser Abschlussarbeit sind jedoch denkbare Themen für eine Studien- oder Seminararbeit.

3.3.5 Lösungs-/Umsetzungsansatz 4

Weitere Verfahren des Information Retrieval, wie beispielsweise Latent Semantic Analysis, oder der Einsatz von maschinellem Lernen und künstlicher Intelligenz auf Basis von Apache Mahout [Mahout] (Stichwort: Collaborative Filtering, Classification, Clustering, usw.) sind weitere, denkbare Lösungs- bzw. Umsetzungsansätze. Die Untersuchung dieser Verfahren und Techniken würden den Umfang dieser Abschlussarbeit jedoch bei weitem sprengen. Daher wird an dieser Stelle nicht weiter darauf eingegangen. Weitere Studien-, Seminar- oder Abschlussarbeiten in diesem Bereich sowie der Vergleich mit den Ergebnissen dieser Arbeit sind allerdings vorstellbar.

4 Prototypische Implementierung

4.1 Eingesetzte Technologien

Für ein besseres Verständnis werden im nachfolgenden Kapitel die bei der Implementierung des Prototypen zum Einsatz kommenden Technologien kurz näher betrachtet.

4.1.1 Apache Hadoop

Apache Hadoop ist ein in Java geschriebenes OpenSource Framework für verteilt und horizontal skalierbar arbeitende Anwendungen [War12]. Es ermöglicht die verteilte und parallele Verarbeitung von riesigen Datenmengen im Tera- und Petabytebereich auf einer großen Anzahl von Computern in einem Rechnerverbund. Hadoop wurde vom Lucene-Erfinder Doug Cutting ins Leben gerufen und ist seit 2008 ein Top-Level-Projekt der Apache Software Foundation. Die entscheidenden Inspirationsquellen für die Entwicklung von Hadoop sind der im Dezember 2004 von den Google-Mitarbeitern Jeffrey Dean und Sanjay Ghemawat veröffentlichte MapReduce-Ansatz [DeG04] sowie die Vorschläge des Google-Dateisystems GFS [GhGL03].

Hadoop besteht aus den beiden Kernbestandteilen - HDFS und MapReduce-Algorithmus - sowie einer Vielzahl von Erweiterungen. Die nachfolgende Abbildung (Quelle: [CC13]) zeigt eine Übersicht über die wichtigsten Komponenten des Hadoop-Ökosystems.

Abbildung 5: Die wichtigsten Komponenten des Hadoop-Ökosystems

Wie bereits erwähnt, ist eines der Kernkomponenten von Hadoop das Java-basierte, hochverfügbare und verteilte Dateisystem HDFS (Hadoop Distributed File System). Es ist auf die zuverlässige Speicherung von großen Dateien sowie auf den schnellen Lesezugriff auf große Datenvolumina ausgelegt. Hierzu werden Blockgrößen von mindestens 64 MB verwendet. Außerdem wird der so genannte "Write Once, Read Many"-Ansatz verfolgt, d.h. Daten werden idealerweise nur einmal geschrieben aber dafür sehr häufig gelesen. Um die Auslastung des Netzwerks zu minimieren und gleichzeitig die Verarbeitung zu beschleunigen, werden die MapReduce-Jobs auf den Knoten ausgeführt, auf denen die zu verarbeitenden Daten gespeichert sind. Dies wird als "Cluster-lokale Verarbeitung" bezeichnet [War12]. Um die Ausfallsicherheit zu garantieren, werden die einzelnen Datenblöcke mehrfach auf verschiedenen Computern im Cluster abgelegt.

Für die Verwaltung der Metadaten sowie für die Bearbeitung von Client-Anfragen ist der NameNode und der Secondary NameNode zuständig. Diese sind die Master-Komponenten in der klassischen Master-Slave-Architektur. Die Speicherung der Daten übernehmen die Slave-Komponenten, die so genannten DataNodes. Die nachfolgende Abbildung (Quelle: [CC13]) gibt einen Überblick über die Architektur des HDFS.

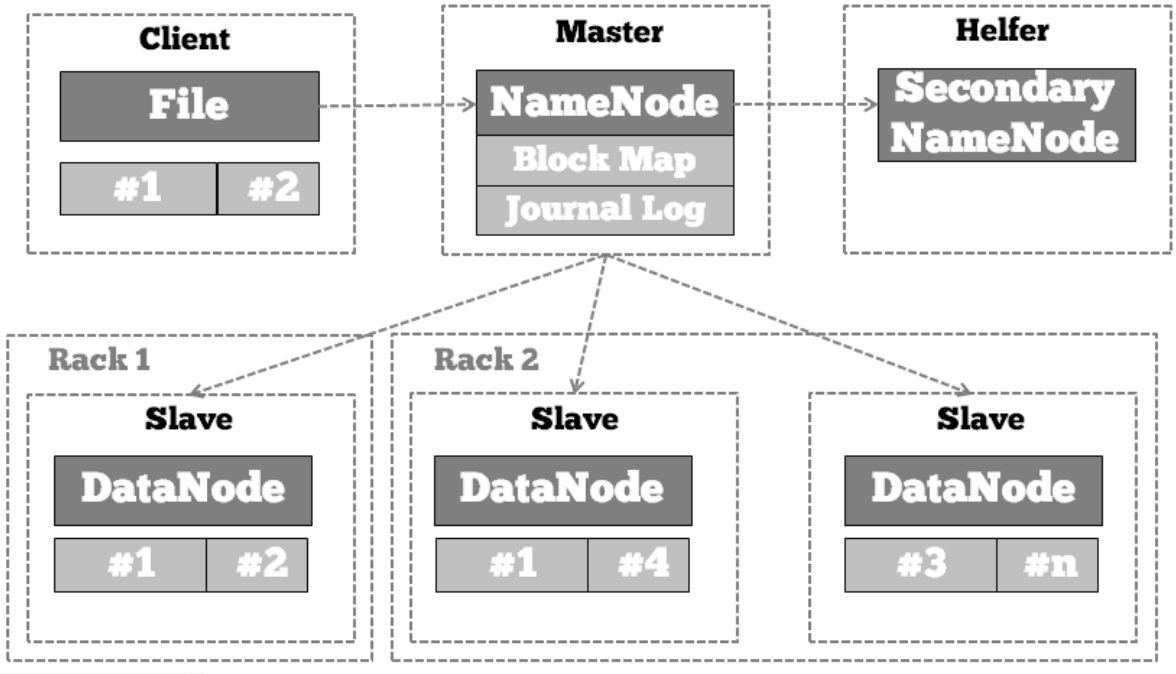

Abbildung 6: Die HDFS-Architektur im Überblick

Die zweite Kernkomponente von Hadoop ist der MapReduce-Algorithmus. Dieser ermöglicht eine effiziente Ausführung von rechenintensiven und nebenläufigen Prozessen auf einem großen Computercluster. Der MapReduce-Ansatz wurde ursprünglich von Google-Mitarbeitern entwickelt und im Jahr 2004 veröffentlicht [DeG04]. Das Ziel war es, durch einen "Teile-und-Herrsche"-Ansatz die Verarbeitung der großen, verteilt gespeicherten Datenmengen so nahe wie möglich am Speicherort durchzuführen. Hierdurch wird die Datenmenge, die über das Netzwerk übertragen werden muss deutlich reduziert und somit die Verarbeitung erheblich beschleunigt. Der MapReduce-Ansatz basiert im allgemeinen auf der Verarbeitung von Schlüssel-Wert-Paaren. Diese findet dabei in einer Map- und einer darauffolgenden Reduce-Phase statt. Zwischen diesen beiden Phasen kann zudem eine Combine- sowie eine Shuffle&Sort-Phase durchgeführt werden. Diese sind für die Zusammenfassung, Sortierung und Verteilung der Teilergebnisse der Map-Phase zuständig. Sowohl die Map- als auch die Reduce-Phase wird in einzelnen Prozessen auf den verschiedenen Knoten des Clusters parallel durchgeführt. Zudem ist eine Aneinanderreihung mehrerer MapReduce-Jobs möglich. Die nachfolgende Abbildung (Quelle: [DeG04]) veranschaulicht die Arbeitsweise des MapReduce-Verfahrens nochmals.

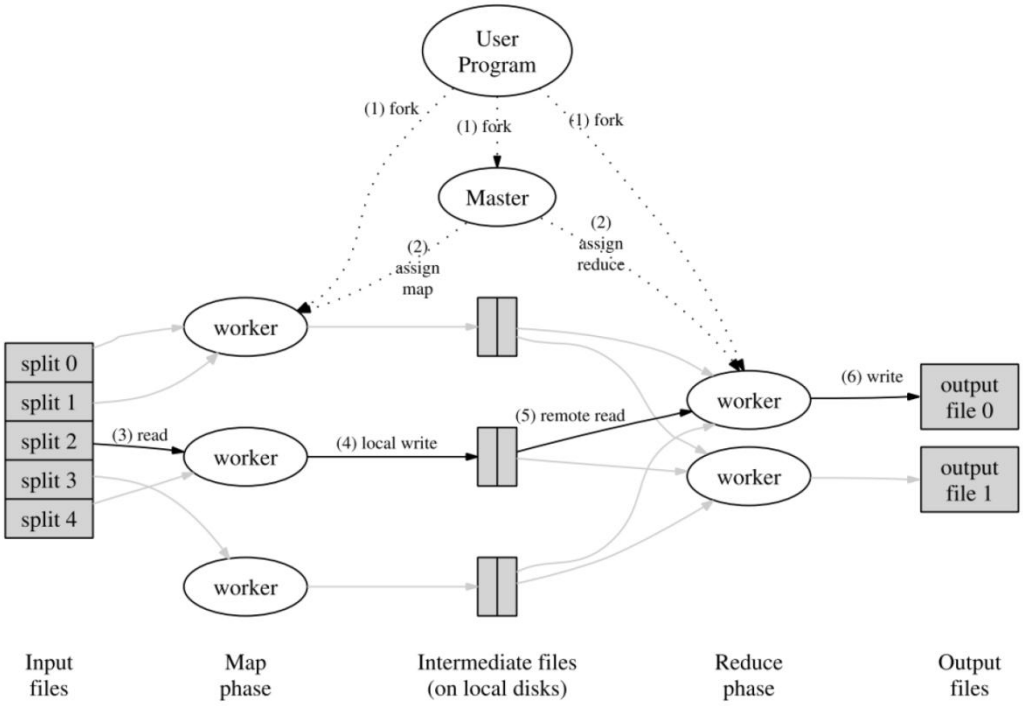

Abbildung 7: Die Arbeitsweise des MapReduce-Verfahrens

Eine ausführliche Beschreibung des MapReduce-Ansatzes sowie des verteilten Dateisystems HDFS würden den Rahmen dieser Abschlussarbeit sprengen. Daher wird an dieser Stelle auf die einschlägige Literatur (vor allem [GhGL03], [DeG04] und [War12]) verwiesen.

4.1.2 Apache HBase

Bei Apache HBase handelt es sich um ein in Java implementiertes, spaltenorientiertes und verteiltes Datenbanksystem. Apache HBase ist Bestandteil des Hadoop-Ökosystems und basiert auf der verteilten Datenbank Bigtable von Google [CDGHW06]. Da HBase auf dem verteilten Dateisystem HDFS aufsetzt, ist eine horizontale Skalierung auf kostengünstiger Standardhardware möglich. Neben dem Zugriff über eine native Java-API ist auch eine REST-Schnittstelle vorhanden sowie MapReduce-Anfragen durchführbar. Die Verwaltung der Daten erfolgt über so genannte Regionen. Diese enthalten alle Datensätze von einem festgelegten Start- bis zu einem festgelegten Endzeilenschlüssel. Der lesende Zugriff sowie die persistente Datenspeicherung erfolgt über die RegionServer bezeichneten Slaves. Die Überwachung und Steuerung übernimmt ein zentraler Master.

4.1.3 SimMetrics

Zu den wichtigsten Prinzipien in der Softwareentwicklung, die von jedem Entwickler beherzigt werden sollten, zählt "DRY: Don't Repeat Yourself". Ziel ist es, redundanten Programmcode zu vermeiden, um die Wart- und Testbarkeit des Quelltextes zu verbessern. Daher bietet es sich an, vor der Implementierung einer Funktionalität nach bestehenden (OpenSource-) Bibliotheken zu suchen, diese in die eigene Anwendung zu integrieren und somit bereits geschriebenen und getesteten Code wiederzuverwenden.

Bei *SimMetrics* [Sim1] handelt es sich um eine umfangreiche, in Java geschriebene und unter der freien Apache Lizenz veröffentlichten Bibliothek mit diversen, weit verbreiteten Ähnlichkeits- und Distanzmetriken. Es handelt sich dabei um eine Abspaltung und Weiterentwicklung des SimMetrics-Projekts von Nicolas Bohorquez Gutierrez [Sim2], welches auf einer an der *UK Sheffield University* entwickelten Bibliothek aufbaut [Sim3].

Die erwähnte SimMetrics-Bibliothek [Sim1] wurde bei der prototypischen Implementierung einsetzt, da sie bereits eine Vielzahl an Ähnlichkeits- und Distanzmetriken sowie weitere nützliche Funktionen für die Vorverarbeitung (siehe Kapitel 2.2) enthält. Darüber hinaus können die einzelnen Metriken sehr einfach genutzt werden. Zu den bereitgestellten Metriken zählen u.a.:

- Levenshtein- und Damerau-Levenshtein-Distanz
- Jaro- und Jaro-Winkler-Distanz
- Smith-Waterman- und Smith-Waterman-Gotoh-Distanz
- Monge-Elkan-Ähnlichkeit
- Sørensen-Dice-Koeffizient und Jaccard-Ähnlichkeit

4.2 Aufbau der Infrastruktur und Clusterkonfiguration

Im nachfolgenden Kapitel wird der Aufbau der Hardware- und Software-Infrastruktur sowie die Konfiguration des in dieser Masterarbeit eingesetzten Hadoop-Clusters detailliert vorgestellt.

4.2.1 Infrastruktur

Das eingesetzte Computer-Cluster wird vom *Competence Center Wirtschaftsinformatik (CCWI)* der *Hochschule München* bereitgestellt und umfasst fünf virtuelle Maschinen, die auf drei physikalischen Servern gehostet werden.

Die Hardware-Ausstattung der drei physikalischen Server ist in nachfolgender Tabelle dargestellt. Neben den fünf virtuellen Maschinen für diese Masterarbeit werden auf dem physikalischen Rechnern noch weitere virtuelle Maschinen für andere Forschungsprojekte der *Hochschule München* gehostet.

Komponente	Eigenschaft
Anzahl der physikalischen Maschinen	3
je physikalischer Maschine:	
Art des Prozessors	Intel® Xeon® Processor X5650 (12M Cache, 2.66 GHz, 6.40 GT/s Intel® QPI)
Anzahl der Kerne / Threads	6 / 12
Arbeitsspeicher	92 Gigabyte
Festplattenkapazität	600 Gigabyte (SAS 6G)
Netzwerkanbindung	10 Gigabit/s

Tabelle 5: Die Leistungsmerkmale der zugrundeliegenden physikalischen Maschinen

Die Hardware-Spezifikationen der fünf virtuellen Maschinen werden nachfolgend tabellarisch dargestellt.

Komponente	Eigenschaft
Anzahl der virtuellen Maschinen	5
je VM:	
Art des Prozessors	Intel® Xeon® Processor X5650 (12M Cache, 2.66 GHz, 6.40 GT/s Intel® QPI)
Anzahl der Kerne / Threads	4 / 4
Arbeitsspeicher	12 Gigabyte
Festplattenkapazität	160 Gigabyte
Netzwerkanbindung	1 Gigabit/s

Tabelle 6: Die Leistungsmerkmale der virtuellen Maschinen

Die eingesetzten Software-Komponenten sowie deren Versionsnummern werden nachfolgend aufgelistet.

Komponente	Eigenschaft
Virtualisierungsumgebung	VMware vCenter Server Version 6.0.0 VMware vSphere Client + Web Client Version 6.0.0
Betriebssystem	CentOS Release 6.7 (Final) - 64 Bit
Hadoop	Version 2.3.2.0-2950
HDFS	Version 2.7.1.2.3
MapReduce2	Version 2.7.1.2.3
YARN	Version 2.7.1.2.3
HBase	Version 1.1.1.2.3
ZooKeeper	Version 3.4.6.2.3
Ambari Metrics	Version 0.1.0

Tabelle 7: Die eingesetzten Software-Komponenten

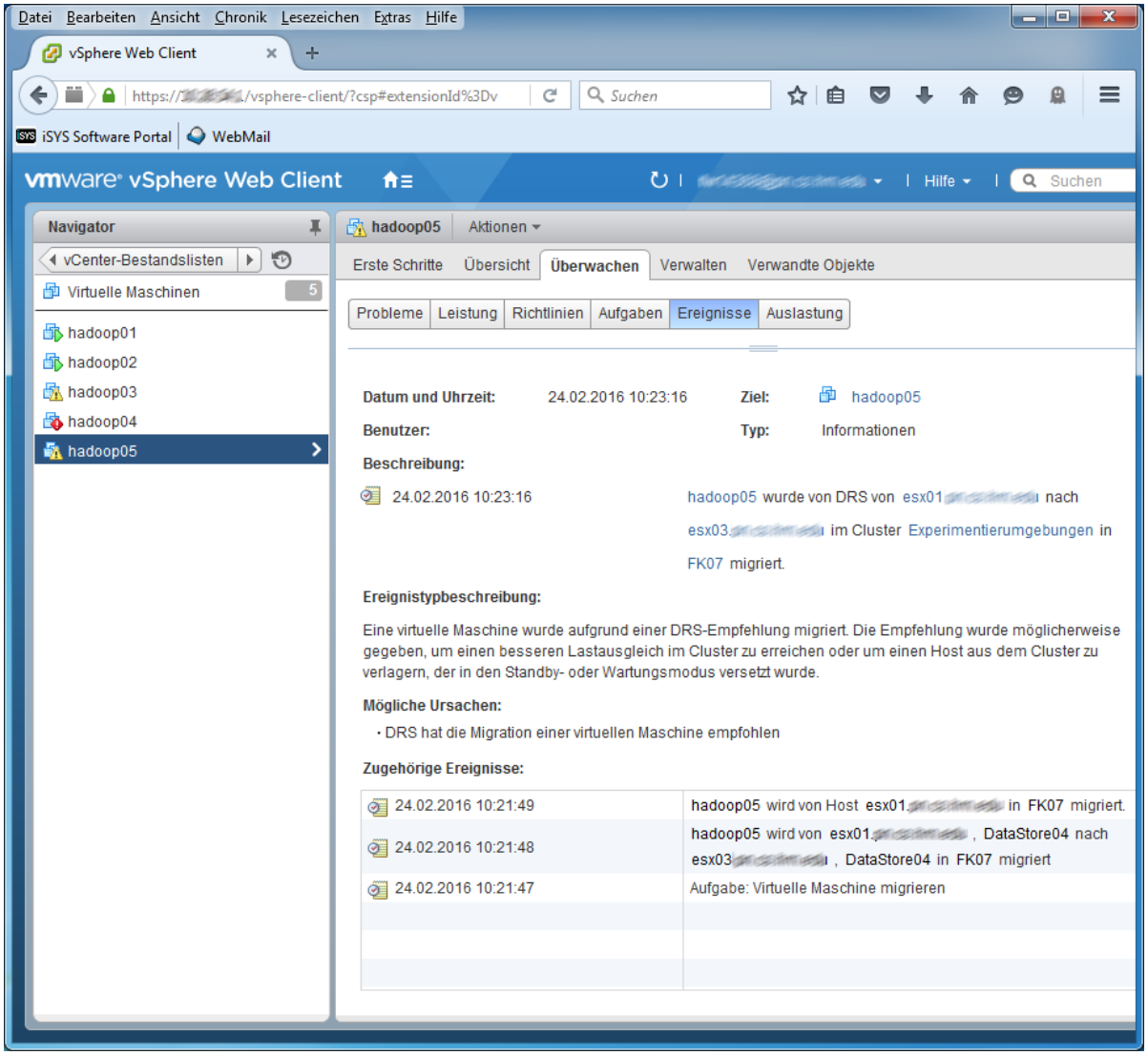

Abbildung 8: Ereignisbeschreibung der Migration einer virtuellen Maschine

Um einen besseren Lastausgleich im Cluster zu erreichen, werden mit Hilfe von vMotion [VMW] einzelne VMs während des laufenden Betriebs ohne Ausfallzeit zwischen physischen Servern migriert. Dabei wird der aktive Speicher und der exakte Ausführungszustand übertragen sowie die Netzwerk-Kennung und die aktiven Netzwerk-Verbindungen beibehalten. Dadurch ist ein schneller und nahtloser Wechsel von der Ausführung der virtuellen Maschine auf dem vSphere-Quellhost zur Ausführung auf dem vSphere-Zielhost möglich. Anwender werden somit nicht von Ausfallzeiten oder Unterbrechungen beeinträchtigt.

Die Migration einer virtuellen Maschine ist beim Einsatz eines Hochgeschwindigkeitsnetzwerks innerhalb weniger Minuten durchgeführt. Durch die 10-Gigabit-Netzwerkanbindung der physischen Server wird eine VM im bereitgestellten Cluster innerhalb von 1,5 Minuten migriert (siehe Abbildung 8: Ereignisbeschreibung der Migration einer virtuellen Maschine).

4.2.2 Clusterkonfiguration

Nachfolgend werden die wichtigsten Konfigurationen der einzelnen Hadoop- und HBase-Komponenten tabellarisch dargestellt.

Komponente	Konfiguration
Anzahl DataNodes	4
Anzahl NodeManager	4
Anzahl ZooKeeper Server	3
HDFS Kapazität	550 Gigabyte
NameNode Heap	1 Gigabyte
ResourceManager Heap	910 Megabyte
Map Memory pro Task	1,5 Gigabyte
Reduce Memory pro Task	3 Gigabyte
Sort Allocation Memory pro Task	860 Megabyte
YARN Memory	24 Gigabyte

Tabelle 8: Apache Hadoop Konfiguration

Komponente	Konfiguration
Anzahl RegionServer	4
HBase Master Heap	2 Gigabyte

Tabelle 9: Apache HBase Konfiguration

Zum besseren Verständnis werden in der nachfolgenden Abbildung die einzelnen Knoten des Hadoop-Clusters sowie die darauf laufenden Services grafisch dargestellt.

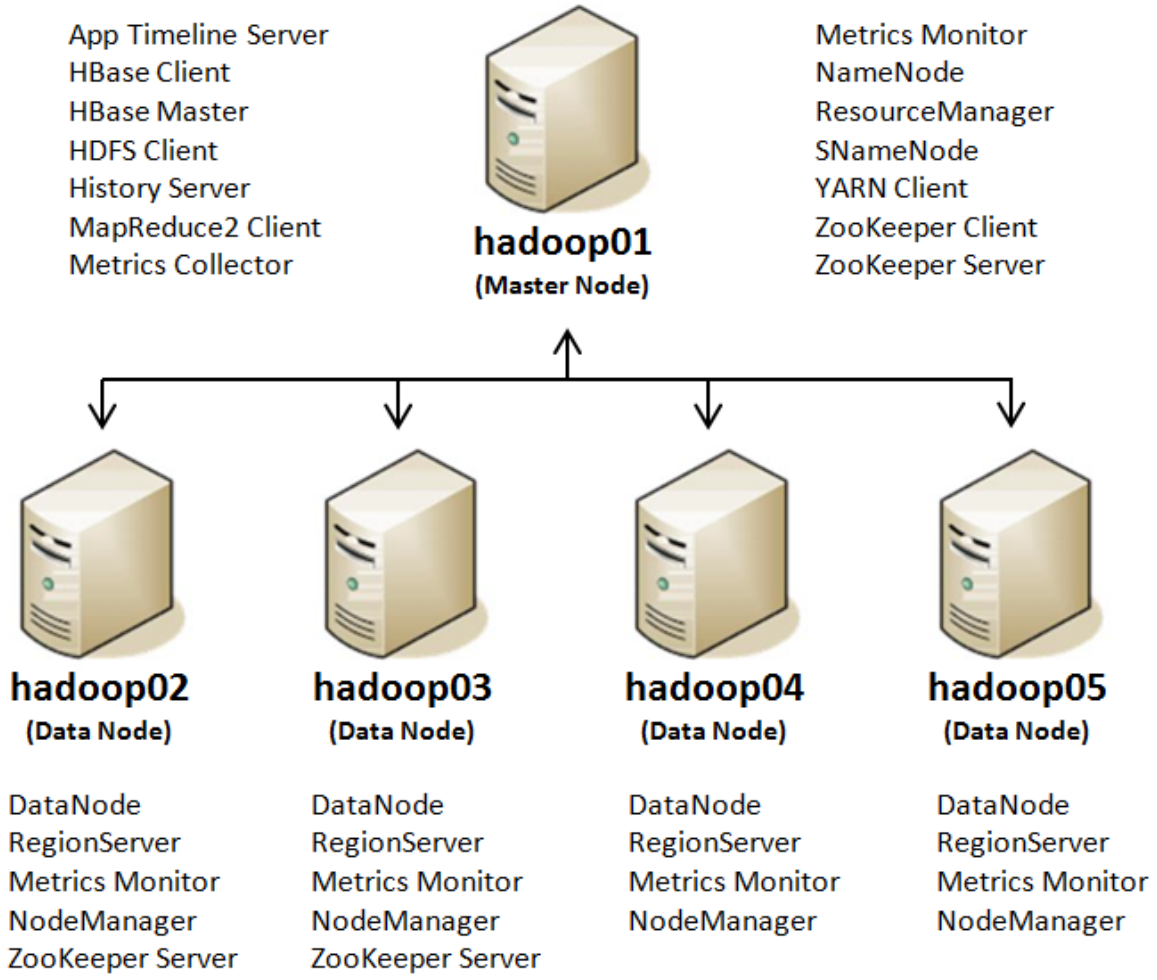

Abbildung 9: Darstellung der Cluster-Infrastruktur

4.3 Analyse der Ausgangsdaten

Für die prototypische Implementierung bzw. für die Qualitätsmessung der einzelnen implementierten Ähnlichkeitsmetriken konnten die Originaldaten der internen Referenzdatenbank der GEMA nicht verwendet werden. Darüber hinaus sieht das entwickelte Konzept (Kapitel 3.3.1) die Einbindung von externen Datenquellen zur Verbesserung der Zuordnungs-Qualität explizit vor. Bei der Recherche nach möglichen Datenbanken mit Informationen und Metadaten über Musikstücke konnten sowohl kommerzielle als auch frei nutzbare Anbieter identifiziert werden. Diese unterscheiden sich in der Aktualität, dem Umfang, der Lizenz sowie der Bereitstellung der Daten. Eine umfangreiche Liste mit über 25 Musik-APIs ist unter [MusicAPI] zu finden. Nachfolgend eine nicht vollständige Auflistung mit diversen kommerziellen sowie frei nutzbaren Online-Datenbank für Diskografien von Musikern und Plattenlabeln.

- Discogs (freie Lizenz)
- MusicBrainz (freie Lizenz)
- CDDB (freie Lizenz, wird nicht mehr weiterentwickelt)
- FreeDB (freie Lizenz, aus CDDB hervorgegangen, wird nicht mehr weiterentwickelt)
- Gracenote (kommerzielle Lizenz, aus CDDB hervorgegangen)
- The Echo Nest (kommerzielle Lizenz)
- AllMusic (kommerzielle Lizenz)
- Last.fm (kommerzielle Lizenz)

Wie bereits in der Anforderungsdefinition in Kapitel 3.2 beschrieben, sollte bei der Auswahl und Nutzung von externen Datenquellen auf eine offene Lizenz geachtet werden. Dadurch haben sich nachfolgende zwei Datenquellen als sehr vielversprechend herauskristallisiert:

Discogs. Bei Discogs [Dis15] handelt es sich um eine im Jahr 2000 gestartete, kostenlose Online-Datenbank für Diskografien von Musikern und Plattenlabeln. Über 200.000 registrierte Mitglieder verwalten ihre private Musiksammlung mit Discogs und erstellen sowie pflegen dadurch die in der Datenbank hinterlegten Informationen. Durch ausgesuchte, stimmberechtigte Moderatoren wird eine hohe Qualität der Daten sichergestellt. Die nachfolgende Tabelle fasst relevante Informationen über Discogs übersichtlich zusammen:

Eigenschaft	Beschreibung
Lizenz	CC0: Public Domain Weltweiter Verzicht auf urheberrechtliche Schutzrechte
Bereitstellung	Monatlich; XML Data Dump
Größe	ca. 4 GB (gepackt); ca. 25 GB (entpackt)
Statistik	ca. 4 Mio. Artists ca. 1 Mio. Labels ca. 6,5 Mio. Releases ca. 56 Mio. Tracks
Inhalt	Artists, Labels, Releases, Tracks

Tabelle 10: Informationen zu Discogs

MusicBrainz. MusicBrainz [MuB15] ist ein 1999 gestartetes Internetprojekt mit dem Ziel, eine offene und freie Musik-Datenbank zu erstellen. Über 250.000 aktive Mitglieder erstellen und pflegen die Informationen der Datenbank. Um die Qualität möglichst hoch zu halten, werden neue Einträge durch Peer-Reviews kontrolliert. Die Musik-Datenbank ist in zahlreichen Computerprogrammen (u.a. Musikplayern, CD-Rippern, Audio-Taggern, uvm.) sowie auf vielen Webseiten (z.B. Last.fm) integriert. Die nachfolgende Tabelle fasst relevante Informationen über MusicBrainz übersichtlich zusammen:

Eigenschaft	Beschreibung
Lizenz	**Kerndaten:** CC0: Public Domain Weltweiter Verzicht auf urheberrechtliche Schutzrechte **Erweiterte Daten:** CC-NC-SA 3.0 Namensnennung, Nicht kommerziell, Weitergabe unter gleichen Bedingungen
Bereitstellung	Wöchentlich; PostgreSQL Data Dump
Größe	ca. 6 GB (gepackt); ca. 30 GB (entpackt)
Statistik	ca. 1 Mio. Artists ca. 0,1 Mio. Labels ca. 1,5 Mio. Releases ca. 20 Mio. Tracks
Inhalt	**Kerndaten:** Artists, Labels, Releases, Tracks **Erweiterte Daten:** Annotations, Tags, Ratings, Statistics, Search Indexes, Edit History

Tabelle 11: Informationen zu MusicBrainz

Für die Entwicklung des Prototypen sowie die Durchführung der Qualitätsmessungen werden die Daten von Discogs verwendet, da diese als XML-Dateien bereitgestellt werden und somit ein Import in das HDFS und in HBase sehr einfach möglich ist. Im Gegensatz dazu müsste das PostgreSQL Data Dump von MusicBrainz zunächst verarbeitet werden, um es im HDFS oder in HBase sinnvoll nutzen zu können. Darüber hinaus ist die Datenbank von Discogs umfangreicher.

4.4 Datenmodell

Um eine Datenbank mit Referenzdaten für das Empfehlungssystem aufzubauen, wird eine HBase-Datenbank verwendet. Wie bereits in Kapitel 4.1.2 kurz beschrieben, handelt es sich bei Apache HBase um eine partiell spaltenorientierte Datenbank. Die Daten werden als Schlüssel-Wert-Paare abgelegt, wobei der Schlüssel dem Spaltennamen entspricht und für jedes Paar individuell festgelegt wird. Da im Vorfeld somit keine festen Spalten definiert werden, ist für jeden Datensatz eine variable Anzahl an Spalten möglich. Wird ein Schlüssel-Wert-Paar in HBase abgelegt, muss diesem eine so genannte ColumnFamily zugeordnet werden. ColumnFamilies dienen zur Gruppierung von Key-Value-Paaren und müssen bereits bei der Erstellung der Tabelle festgelegt werden. Um ein Schlüssel-Wert-Paar innerhalb einer Tabelle eindeutig identifizieren zu können, muss neben dem Schlüssel auch die ColumnFamily mit angegeben werden. Mit Hilfe eines einmaligen RowKeys kann ein bestimmter Datensatz innerhalb einer Tabelle eindeutig identifiziert werden. Da keine Datentypen existieren und die Daten in HBase als Byte-Array kodierte Textwerte gespeichert werden, muss sich die Anwendung um die richtige Interpretation der einzelnen Datenwerte kümmern.

Die nachfolgende Abbildung zeigt beispielhaft die mehrdimensionale Datenstruktur einer partiell spaltenorientierten Datenbank (z.B. HBase). Für tiefergehende Informationen wird an dieser Stelle auf die einschlägige Literatur zum Thema spaltenorientierte Datenbanken und HBase verwiesen.

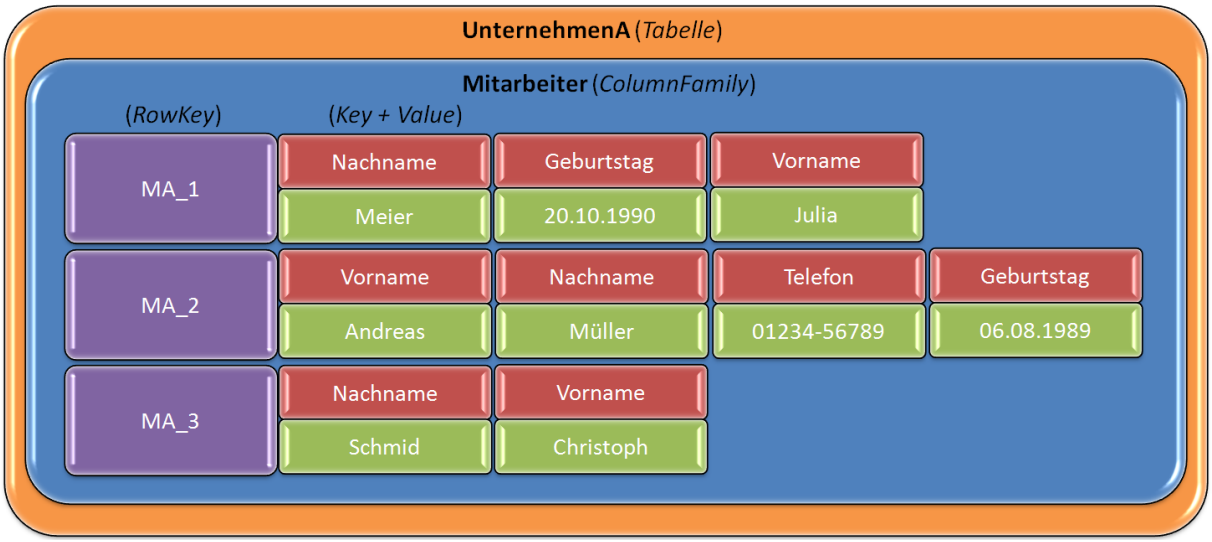

Abbildung 10: Mehrdimensionale Datenstruktur einer partiell spaltenorientierten Datenbank

Für das gegebene Szenario wurden insgesamt drei HBase-Tabellen erstellt. Die *Releases*-Tabelle dient für die Speicherung der Referenzdaten. Da bei der prototypischen Implementierung die Daten nur von Discogs stammen, wurde die Tabelle speziell auf die Datenfelder von Discogs bzw. auf die Nutzung innerhalb des Prototypen optimiert. Diese Tabelle ist für einen späteren produktiven Einsatz ebenfalls notwendig, ggf. jedoch in einem überarbeiteten und an die jeweiligen Anforderungen angepassten Datenmodell. Die beiden anderen Tabellen (*Artists* und *Benchmark*) sind nur für eine Nutzung innerhalb des Prototypen relevant. Auf den nachfolgenden Seiten werden nun die Datenmodelle der drei Tabellen näher erläutert.

Wie bereits in Kapitel 4.1.2 erwähnt, verwaltet HBase die Daten einer Tabelle mit Hilfe von Regionen. Diese Regionen spiegeln dabei u.a. die Anzahl der HFiles wider, in denen die Daten abgelegt werden. Dadurch wird auch festgelegt, mit wie vielen Tasks (eines MapReduce-Prozesses) parallel auf einer Tabelle gearbeitet werden kann. Daher sollte die Anzahl der Regionen möglichst optimiert werden. Beim Erstellen der Tabelle kann die Anzahl der Regionen optional festgelegt werden; andernfalls wird diese automatisch vom System vorgegeben. Es wird jedoch empfohlen, die Anzahl explizit anzugeben, um sowohl eine hohe Performance als auch Stabilität der Datenbank zu gewährleisten. Laut [IBM] sollte die Anzahl der Regionen mit Hilfe der nachfolgenden Formel berechnet werden:

$$(Anzahl\ RegionServer - 1) \times Anzahl\ logischer\ Processoren\ je\ RegionServer\ Knoten$$

Die nachfolgende Abbildung zeigt das vollständige Datenmodell der Tabelle *Releases*. Dieses wird anschließend nochmals detailliert beschrieben.

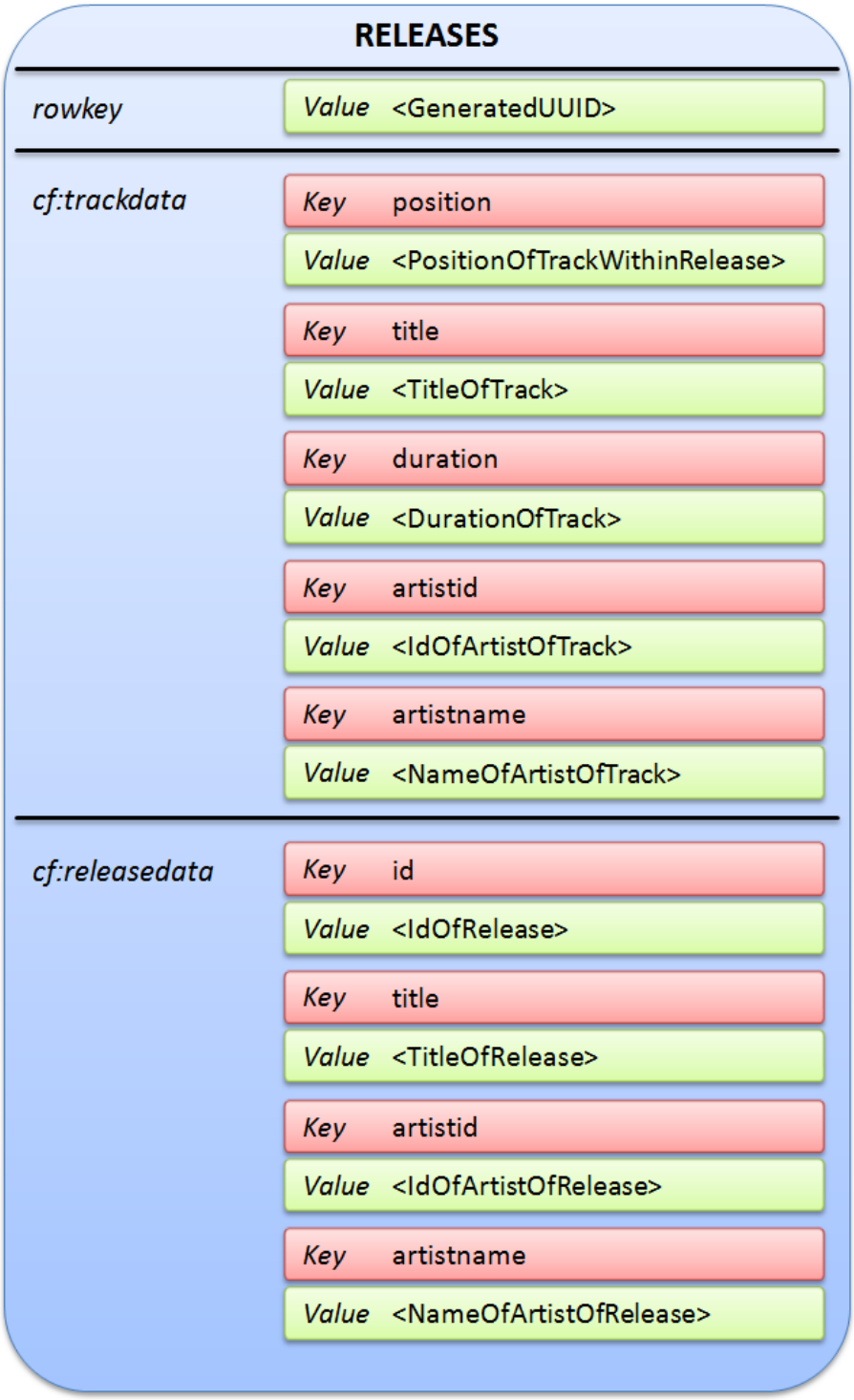

Abbildung 11: Das HBase-Datenmodell zur Speicherung der Releases

Um die Daten innerhalb der Regionen möglichst gleichmäßig zu verteilen und so genanntes Region-Hotspotting zu vermeiden, wird als RowKey für jeden Datensatz eine zufällige 128 Bit lange und hexadezimal notierte sowie eindeutige UUID (Universally Unique Identifier) generiert. Dieses Vorgehen wird von [IBM] als Best Practice im RowKey-Design vorgeschlagen. Beim Erstellen der Tabelle werden zwei ColumnFamilies - *trackdata* und *releasedata* - festgelegt. Die ColumnFamily *trackdata* enthält dabei Informationen über den Track. Dies sind Titel, Artist und Dauer des Musikstückes sowie die Position innerhalb der Veröffentlichung (z.B. einer CD). In der ColumnFamily *releasedata* werden spezielle Informationen zum Release gespeichert. Neben der eindeutigen Nummer sind dies Titel und Artist der Veröffentlichung. Im dargestellten Datenmodell werden beim Artisten neben dem Namen des Künstlers auch die ID gespeichert. Bei HBase handelt es sich jedoch um eine nicht-relationale Datenbank, d.h. Daten werden u.U. mehrfach redundant gespeichert; eine Normalisierung des Datenmodells findet nicht statt. Die ID des Artisten wird in diesem Szenario lediglich gespeichert, um zu einem späteren Zeitpunkt Benchmark-Testdaten mit alternativen Namen der Artisten erzeugen zu können. Diese Alternativ-Namen befinden sich jedoch in einer separaten XML-Datei von Discogs und werden zur Vereinfachung in eine separate Tabelle innerhalb der HBase-Datenbank importiert. Über die gespeicherten IDs der Artisten erfolgt anschließend die Verknüpfung.

Der nachfolgende Quellcode zeigt den Befehl zum Erstellen der HBase-Tabelle. Neben dem Namen der Tabelle müssen nur eine oder mehrere ColumnFamilies angegeben werden. Weitere Parameter sind optional. Wie bereits weiter oben beschrieben, ist es jedoch sinnvoll bzw. empfohlen, die Anzahl der Regionen manuell festzulegen. Im zur Verfügung gestellten Cluster (Konfiguration im Kapitel 4.2) sind vier DataNodes mit jeweils vier logischen Prozessoren vorhanden. Auf allen vier DataNodes ist zudem ein RegionServer aktiv. Hieraus ergibt sich eine Anzahl von zwölf Regionen. Als Algorithmus zur Festlegung der Regionen-Grenzen (Anfangs- und Endschlüssel einer Region) wird der von HBase bereitgestellte HexStringSplit verwendet. Dies ist möglich, da als RowKey eine hexadezimal notierte UUID verwendet wird. Der Quellcode kann in der HBase-Shell ausgeführt werden.

```
create 'releases',
       {NAME => 'trackdata'}, {NAME => 'releasedata'},
       {NUMREGIONS => 12, SPLITALGO => 'HexStringSplit'}
```

Um für die Benchmarks aussagekräftige Testdaten generieren zu können, werden u.a. die Alternativ-Namen der Artisten von Discogs verwendet. Hierbei handelt es sich um Namen, unter denen die Artisten ebenfalls bekannt sind. Beispielsweise Abkürzungen, Mittelnamen oder Namenszusätze.

Diese Informationen werden von Discogs in einer separaten XML-Datei ausgeliefert. Um die Verarbeitung dieser Informationen zu erleichtern, wird für den Import eine gesonderte Tabelle *Artists* erstellt. Da in HBase ein direkter Zugriff auf einen bestimmten Datensatz nur über den RowKey möglich ist, wird statt einer zufällig generierten UUID die bereitgestellte ID des Artisten verwendet. Dadurch ist eine Zuordnung zu den Artisten-IDs in der *Release*-Tabelle mittels Java oder MapReduce leicht möglich. In der ColumnFamily *basedata* wird in diesem Fall nur der Originalname des Artisten abgelegt. Weitere Informationen wären denkbar, sind im vorliegenden Szenario jedoch nicht relevant. In einer zweiten ColumnFamily *namevariations* werden die verschiedenen Alternativ-Namen des Artisten gespeichert. Da sich die Anzahl der Namensvarianten von Artist zu Artist unterscheidet, ist eine spaltenorientierte Datenbank wie HBase (die eine variable Anzahl an Spalten je Datensatz ermöglicht) prädestiniert für die Speicherung dieser Daten. Um die Anzahl der Varianten für einen bestimmten Artisten schnell ermitteln zu können, wird diese in einem Schlüssel-Wert-Paar gespeichert. Die einzelnen Alternativ-Namen werden in eigenen Key-Value-Paaren mit fortlaufender Nummerierung abgelegt. Die nachfolgende Abbildung zeigt das Datenmodell der Tabelle *Artists*.

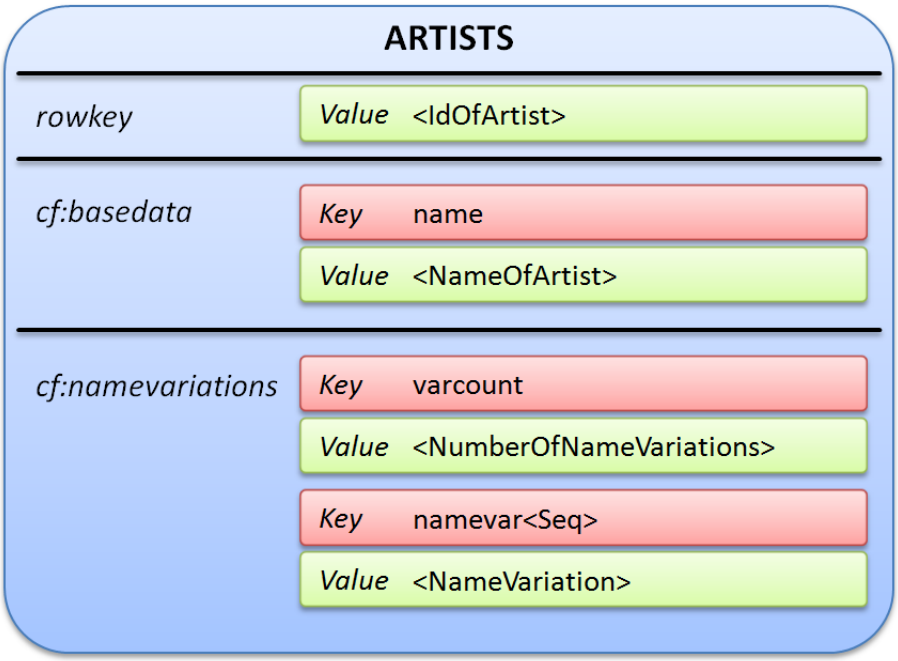

Abbildung 12: Das HBase-Datenmodell zur Speicherung der Artisten

Der nachfolgende Quellcode zeigt den Befehl zum Erstellen der HBase-Tabelle. Da die Tabelle mehr als vier Millionen Datensätze enthält, werden ebenfalls zwölf Regionen festgelegt.

```
create 'artists',
       {NAME => 'basedata'}, {NAME => 'namevariations'},
       {NUMREGIONS => 12, SPLITALGO => 'HexStringSplit'}
```

Für den Prototyp werden die generierten Testdaten für den Benchmark ebenfalls in einer HBase-Tabelle gespeichert bzw. später aus dieser ausgelesen. Für einen zukünftigen produktiven Betrieb ist jedoch auch der Import von Test- bzw. Realdaten über beispielsweise eine CSV-Datei im HDFS denkbar. Der RowKey eines manipulierten Datensatzes der *Benchmark*-Tabelle enthält den RowKey des entsprechenden, originalen Datensatzes der *Release*-Tabelle. In der ColumnFamily *testdata* werden die - ggf. zuvor manipulierten - Artist-, TrackTitle- und ReleaseTitle-Werte gespeichert. Details zur Generierung der Testdaten sind im Kapitel 4.5 und im Kapitel 5.2 zu finden.

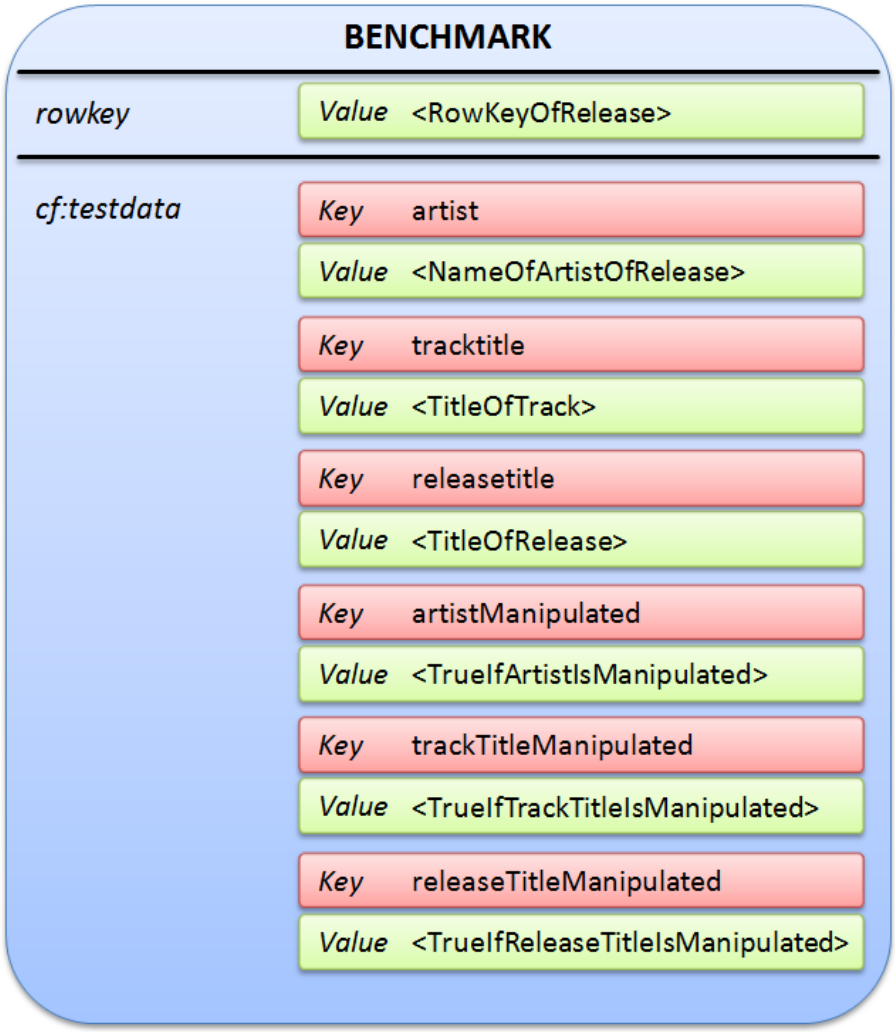

Abbildung 13: Das HBase-Datenmodell zur Speicherung der Benchmark-Daten

Der nachfolgende Quellcode zeigt den Befehl zum Erstellen der HBase-Tabelle. Da die Tabelle kaum Datensätze enthalten wird (max. wenige Tausend), wird diese nicht in einzelne Regionen aufgeteilt.

```
create 'benchmark',
       {NAME => 'testdata'}
```

4.5 Implementierung

Die prototypische Implementierung des Empfehlungssystems besteht aus zwei Java-Projekten: *DataImport-Client* und *Benchmark-Client*. Ersteres enthält die komplette Logik zur Verarbeitung der XML-Dateien von Discogs sowie den Import der Daten in die HBase-Datenbank. Letzteres umfasst den Quellcode für die Generierung von Testdaten sowie für die Durchführung der Benchmarks. Die Programmlogik ist dabei zum Großteil in MapReduce-Jobs implementiert. Auf den nachfolgenden Seiten wird die Implementierung des Prototypen näher erläutert und dabei auf wichtige Abschnitte, Entscheidungen und Erkenntnisse eingegangen. Zum besseren Verständnis wird an manchen Stellen der Quellcode auszugsweise dargestellt.

DataImport-Client. Um Eingabedaten mit einem MapReduce-Job verarbeiten zu können, werden von Apache Hadoop verschiedene Dateieingabeformate (z.B. für CSV-Dateien) bereitgestellt. Für die Verarbeitung von XML-Dateien wird jedoch kein Eingabeformat zur Verfügung gestellt. Daher wurde ein eigenes Dateieingabeformat für XML-Dateien entwickelt, welches auf dem *XmlInputFormat* des Apache Mahout-Projekts [Mahout] aufbaut. Dieses splittet eine gegebene XML-Datei nach zuvor definierten Start- und Ende-Tags in einzelne XML-Fragmente auf. Die so erzeugten XML-Fragmente werden anschließend im Map-Schritt des MapReduce-Jobs weiterverarbeitet.

Um die notwendigen Informationen aus den XML-Fragmenten herausfiltern zu können, müssen diese zunächst geparst werden. Alternativ kann auch ein XML-to-JavaObject-Mapper verwendet werden. In dieser Abschlussarbeit wird die freie Java-Bibliothek Xmappr [Xmappr] genutzt. Diese ermöglicht die Abbildung von Java-Objekten zu XML und umgekehrt. Hierfür muss zunächst die XML-Struktur über Java-Klassen nachgebildet werden. Um eine fehlerfreie Zuordnung durchführen zu können, müssen diese Klassen sowie deren Attribute mit Annotationen (z.B. @Attribute, @RootElement oder @Element) versehen werden. Ein Vorteil von Xmappr ist, dass für eine Durchführung der Zuordnung nicht alle Elemente der XML-Struktur in Java nachgebildet werden müssen, sondern lediglich die für den jeweiligen Anwendungsfall notwendigen. Fehlende Elemente werden beim Mapping-Vorgang einfach ignoriert. Die für die Verarbeitung der XML-Dateien von Discogs erforderlichen Zuordnungs-Klassen sind im Anhang unter 8.2.1 abgedruckt. Zum besseren Verständnis ist ein exemplarischer Auszug der beiden XML-Dateien mit Informationen zu Artisten bzw. Releases im Anhang unter 8.1.1 und 8.1.2 abgebildet.

Innerhalb des Map-Schritts wird zunächst das übergebene XML-Fragment mit Hilfe von Xmappr auf ein Java-Objekt der Release-Klasse gemappt. Anschließend werden die Informationen in die HBase-Datenbank gespeichert. Dies erfolgt aus Geschwindigkeits-Gründen jedoch nicht über einzelne Put-

Anweisungen, die direkt in die Datenbank schreiben, sondern über Bulk-Loading-Mechanismen. Die für den Import notwendige "Bulk Load"-Datei wird durch einen von HBase bereitgestellten Reducer erzeugt. Deshalb ist für den Import der Discogs-Daten nur die Implementierung eines Mappers notwendig. Der nachfolgend abgedruckte Quellcode zeigt die eben beschriebene Map-Funktion.

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
  Release release = (Release)this.xmappr.fromXML(new StringReader(value.toString()));

  if (release != null && release.tracklist != null && release.tracklist.track != null) {
    for (Track track : release.tracklist.track) {
      byte[] rowKey = Bytes.toBytes(UUID.randomUUID().toString());
      this.resultKey = new ImmutableBytesWritable(rowKey);

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                         Bytes.toBytes("position"),
                                                         Bytes.toBytes(track.position)));

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                         Bytes.toBytes("title"),
                                                         Bytes.toBytes(track.title)));

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                         Bytes.toBytes("duration"),
                                                         Bytes.toBytes(track.duration)));

      if (track.artists != null && track.artists.artist != null) {
        context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                            Bytes.toBytes("artistid"),
                                                            Bytes.toBytes(((Artist)track.artists.artist.toArray()[0]).id)));

        context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                            Bytes.toBytes("artistname"),
                                                         Bytes.toBytes(((Artist)track.artists.artist.toArray()[0]).name)));
      } else {
        context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                            Bytes.toBytes("artistid"),
                                                         Bytes.toBytes(((Artist)release.artists.artist.toArray()[0]).id)));

        context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("trackdata"),
                                                            Bytes.toBytes("artistname"),
                                                      Bytes.toBytes(((Artist)release.artists.artist.toArray()[0]).name)));
      }

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("releasedata"),
                                                         Bytes.toBytes("id"),
                                                         Bytes.toBytes(release.id)));

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("releasedata"),
                                                         Bytes.toBytes("title"),
                                                         Bytes.toBytes(release.title)));

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("releasedata"),
                                                         Bytes.toBytes("artistid"),
                                                         Bytes.toBytes(((Artist)release.artists.artist.toArray()[0]).id)));

      context.write(this.resultKey, new KeyValue(rowKey, Bytes.toBytes("releasedata"),
                                                         Bytes.toBytes("artistname"),
                                                      Bytes.toBytes(((Artist)release.artists.artist.toArray()[0]).name)));
    }
  }
}
```

Benchmark-Client. Der Benchmark-Client umfasst zwei Funktionsbereiche: Die Generierung von Testdaten sowie die Durchführung der Benchmarks mit den verschiedenen Ähnlichkeitsmetriken.

Um Testdaten zu generieren, wird eine bestimmte Anzahl an Original-Datensätzen aus der *Release*-Tabelle zufällig ausgewählt, diese mit einer zuvor festgelegten Wahrscheinlichkeit manipuliert und abschließend in der *Benchmark*-Tabelle gespeichert. Hierfür wird kein MapReduce-Job verwendet, sondern die Java-API von HBase eingesetzt. Die Auswahl von zufälligen Datensätzen erfolgt durch

den Einsatz des *RandomRowFilters* der HBase-API. Anschließend wird über diese Datensätze iteriert, ggf. die Manipulationen durchgeführt und per HBase-Java-API in die *Benchmark*-Tabelle gespeichert. Das Vorgehen beim Manipulieren der Datensätze wird in Kapitel 5.2 Erzeugung der Testdaten genauer beschrieben. Aufgrund des Umfangs sowie der "Banalität" des für die Generierung der Testdaten notwendigen Java-Codes (es handelt sich dabei lediglich um die Verschachtelung von Schleifen und diversen Verzweigungen), wird an dieser Stelle auf das Abdrucken des Quelltextes verzichtet.

Der zweite Teil des Benchmark-Clients enthält die Programmlogik für die Durchführung der Benchmarks mit den verschiedenen Ähnlichkeitsmetriken. Diese wird innerhalb eines MapReduce-Jobs implementiert. Ein MapReduce-Job besteht neben der *Map*- und der *Reduce*-Methode, die die Verarbeitungslogik des MapReduce-Jobs enthält, aus zwei *Setup*- und zwei *Cleanup*-Methoden für die Map- bzw. die Reduce-Phase. Diese sind - wie die Namen bereits vermuten lassen - für Initialisierungs- bzw. Aufräumarbeiten zuständig und müssen nicht zwingend implementiert werden. Um eine parallele und verteilte Verarbeitung zu ermöglichen, wird sowohl der Mapper als auch der Reducer mehrfach parallel (auf einem einzelnen Rechner und innerhalb eines Rechnerverbundes) ausgeführt. Diese ausführenden Einheiten werden als Mapper- bzw. Reducer-Tasks eines MapReduce-Jobs bezeichnet. Die Anzahl der parallel ausgeführten Mapper/Reducer-Tasks ist beim Einsatz des TableMappers/TableReducers beispielsweise von der Anzahl der Regionen der HBase-Tabelle abhängig (siehe Kapitel 4.4). Während die Methoden *Map* und *Reduce* innerhalb eines Task mehrmals ausgeführt werden können (beim TableMapper wird die *Map*-Methode beispielsweise für jeden Datensatz einer Region einmal aufgerufen), werden die *Setup*-/*Cleanup*-Methoden innerhalb eines Tasks nur einmal zu Beginn/zum Ende der Verarbeitung aufgerufen. Der nachfolgende Auszug zeigt exemplarisch den Hadoop-Quellcode für die Verarbeitung des Reducer-Tasks.

```
public void run(Context context) throws IOException, InterruptedException {
  setup(context);
  try {
    while (context.nextKey()) {
      reduce(context.getCurrentKey(), context.getValues(), context);
    }
  } finally {
    cleanup(context);
  }
}
```

Beim Benchmark-Client wird neben den Methoden *Map* und *Reduce* die *Setup*-Methode des Mappers sowie die *Cleanup*-Methode des Reducers implementiert. Der Mapper ist dabei für die Initialisierung der Ähnlichkeitsmetriken und das Laden der Testdaten (innerhalb der *Setup*-Methode) sowie für die Berechnung der Ähnlichkeiten zwischen den Testdaten und allen Datensätzen des

Referenzdatenbestandes (in der *Map*-Methode) zuständig. Um die Ähnlichkeit zu ermitteln, kommt die in Kapitel 4.1.3 vorgestellte, Java-basierte OpenSource-Bibliothek *SimMetrics* zum Einsatz. Beim Aufruf des MapReduce-Jobs kann über zwei Parameter die zu verwendende Ähnlichkeitsmetrik sowie die Größe der q-Gramme (nur bei Token-basierten Metriken) festgelegt werden. Diese Parameter werden in der Driver-Klasse des MapReduce-Jobs entgegengenommen und validiert sowie anschließend mit Hilfe der Konfigurations-Klasse des MapReduce-Kontextes an den Mapper weitergegeben. Der nachfolgende Quelltext zeigt die Initialisierung der Ähnlichkeitsmetrik sowie das Laden der Testdaten innerhalb der *Setup*-Methode des Mappers.

```
public void setup(Context context) throws IllegalArgumentException, IOException {
  Configuration conf = context.getConfiguration();
  String metricType = conf.get("metric.type");
  int metricQGramSize = Integer.parseInt(conf.get("metric.qgramsize"));

  switch (metricType) {
  case "jaro":
      this.metric = with(new Jaro()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "jarowinkler":
      this.metric = with(new JaroWinkler()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "levenshtein":
      this.metric = with(new Levenshtein()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "dameraulevenshtein":
      this.metric = with(new DamerauLevenshtein()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "longestcommonsubsequence":
      this.metric = with(new LongestCommonSubsequence()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "longestcommonsubstring":
      this.metric = with(new LongestCommonSubstring()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "smithwaterman":
      this.metric = with(new SmithWaterman()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "smithwatermangotoh":
      this.metric = with(new SmithWatermanGotoh()).simplify(Simplifiers.toLowerCase()).build();
      break;
  case "dice":
      this.metric = with(new Dice<String>()).simplify(Simplifiers.toLowerCase())
                      .tokenize(Tokenizers.qGram(metricQGramSize)).build();
      break;
  case "jaccard":
      this.metric = with(new Jaccard<String>()).simplify(Simplifiers.toLowerCase())
                      .tokenize(Tokenizers.qGram(metricQGramSize)).build();
      break;
  case "mongeelkan":
      this.metric = StringMetrics.mongeElkan();
      break;
  default:
      throw new IllegalArgumentException("Wrong Metric Type: " + metricType);
  }
  HTable table = null;
  ResultScanner scanner = null;
  try {
    table = new HTable(HBaseConfiguration.create(conf), Constants.TABLE_BENCHMARK);
    scanner = table.getScanner(new Scan());

    this.source = new ArrayList<BenchmarkSource>();
    for (Result result : scanner) {
      this.source.add(new BenchmarkSource(Bytes.toString(result.getRow()),
                                          Bytes.toString(result.getColumnLatest(Constants.FAMILY_BENCHMARK_TESTDATA,
                                                                                Constants.QUALIFIER_BENCHMARK_ARTIST)
                                                                                .getValue()),
                                          Bytes.toString(result.getColumnLatest(Constants.FAMILY_BENCHMARK_TESTDATA,
                                                                                Constants.QUALIFIER_BENCHMARK_TRACKTITLE)
                                                                                .getValue()),
                                          Bytes.toString(result.getColumnLatest(Constants.FAMILY_BENCHMARK_TESTDATA,
                                                                                Constants.QUALIFIER_BENCHMARK_RELEASETITLE)
                                                                                .getValue())));
    }
  } finally { ... }
}
```

Für jeden Datensatz des Referenzdatenbestandes wird die *Map*-Methode einmal aufgerufen. Innerhalb dieser werden die notwendigen Informationen (Artist, Titel des Liedes und Titel des Releases) des jeweiligen Datensatz ausgelesen und anschließend mit jedem Datensatz der zuvor in der *Setup*-Methode geladenen Testdatenmenge verglichen. Werden bei den ermittelten Ähnlichkeitsmetriken für Artist, Liedtitel und Releasetitel die zuvor festgelegten Schwellwerte überschritten, handelt es sich um eine potentielle Übereinstimmung und der Datensatz sowie die ermittelten Metriken werden zur späteren Verarbeitung an den Reducer weitergereicht. Die Klasse *SingleMetricOutput* (für den Austausch von individuellen Daten zwischen Mapper und Reducer) ist unter 8.2.2 abgedruckt. Der nachfolgende Quelltext zeigt die beschriebene Logik der *Map*-Methode.

```
protected void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
    this.artist = Bytes.toString(value.getColumnLatest(Constants.FAMILY_RELEASES_TRACKDATA,
                                                       Constants.QUALIFIER_RELEASES_ARTIST).getValue());
    this.trackTitle = Bytes.toString(value.getColumnLatest(Constants.FAMILY_RELEASES_TRACKDATA,
                                                           Constants.QUALIFIER_RELEASES_TRACKTITLE).getValue());
    this.releaseTitle = Bytes.toString(value.getColumnLatest(Constants.FAMILY_RELEASES_RELEASEDATA,
                                                             Constants.QUALIFIER_RELEASES_RELEASETITLE).getValue());

    for (BenchmarkSource src : this.source) {
        this.resultArtist = this.metric.compare(this.artist, src.getArtist());
        if (this.resultArtist >= Constants.THRESHOLD_ARTIST) { // >= 0.8
            this.resultTrackTitle = this.metric.compare(this.trackTitle, src.getTrackTitle());
            if (this.resultTrackTitle >= Constants.THRESHOLD_TRACKTITLE) { // >= 0.8
                this.resultReleaseTitle = this.metric.compare(this.releaseTitle, src.getReleaseTitle());
                if (this.resultReleaseTitle >= Constants.THRESHOLD_RELEASETITLE) { // >= 0.8
                    context.write(new Text(src.getId()),
                                  new SingleMetricsOutput(Bytes.toString(value.getRow()),
                                                          this.resultArtist,
                                                          this.resultTrackTitle,
                                                          this.resultReleaseTitle));
                }
            }
        }
    }
}
```

Wie in der Anforderungsdefinition (Kapitel 3.2) festgelegt, sollen die in der Dokumentation des Master Data Management (MDM) der GEMA beschriebenen allgemeinen Regeln zur Bestimmung von Produktdatendubletten berücksichtigt werden. Diese unterscheiden zwischen vollständig identischen und teilidentischen Dubletten. Als identisch werden zwei Datensätze bezeichnet, wenn der Titel des Liedes sowie der Artist zu 100 Prozent übereinstimmen und alle weiteren Attribute (z.B. ISRC-Nummer, Dauer, Produktionsjahr, und noch einige weitere) vorhanden sind und ebenfalls übereinstimmen. Als teilidentisch werden Datensätze bezeichnet, wenn der Titel des Liedes sowie der Artist zu mindestens 80% übereinstimmen und mindestens eines der weiteren Attribute vorhanden ist und übereinstimmt. Darüber hinaus wird mit einigen weiteren Regeln zwischen verschiedenen Versionen unterschieden. Innerhalb der prototypischen Implementierung werden, zur Vereinfachung und weil nicht alle Informationen (z.B. ISRC-Nummer) vorhanden sind, nur Teile der Regeln umgesetzt: Zwei Datensätze werden dann potentiell übereinstimmend gewertet, wenn sowohl Artist als auch Liedtitel und Releasetitel zu mindestens 80 Prozent übereinstimmen.

Wie eben beschrieben, ist der Mapper für die Ermittlung aller Referenzdatensätze zuständig, die in den drei Bereichen (Artist, Liedtitel und Releasetitel) eine Übereinstimmung von mindestens 80 Prozent zu den Testdatensätzen aufweisen. Diese Datensätze werden an den Reducer weitergeleitet. Da es vorkommen kann, dass für einen Testdatensatz mehrere Referenzdatensätze den Schwellwert von 80 Prozent übersteigen, muss abschließend für jeden Testdatensatz der Referenzdatensatz mit der höchsten Übereinstimmung bestimmt werden. Hierfür ist der Reducer zuständig. Dieser erhält, gruppiert für jeden Testdatensatz, alle gefundenen Referenzdatensätze. Anschließend wird über die drei einzelnen Übereinstimmungswerte pro Datensatz (für Artist, Liedtitel und Releasetitel) ein Durchschnittswert errechnet und mit Hilfe einer einfachen Schleife die beste bzw. höchste Übereinstimmung ermittelt und - im Prototyp - in eine Datei ins HDFS geschrieben. Der nachfolgende Quelltext zeigt dies exemplarisch.

```
protected void reduce(Text key, Iterable<SingleMetricsOutput> values, Context context)
  throws IOException, InterruptedException {

  SingleMetricsOutput bestMatch = new SingleMetricsOutput("", 0.0f, 0.0f, 0.0f);
  for (SingleMetricsOutput value : values) {
    if (bestMatch.getMetricOverall() <= value.getMetricOverall()) {
        bestMatch.setTrackId(value.getTrackId());
        bestMatch.setMetricOverall(value.getMetricOverall());
        bestMatch.setMetricArtist(value.getMetricArtist());
        bestMatch.setMetricTrackTitle(value.getMetricTrackTitle());
        bestMatch.setMetricReleaseTitle(value.getMetricReleaseTitle());
    }
  }
  context.write(key,
                new Text(String.format("%s | %f | %f | %f | %f", bestMatch.getTrackId(),
                                                                 bestMatch.getMetricOverall(),
                                                                 bestMatch.getMetricArtist(),
                                                                 bestMatch.getMetricTrackTitle(),
                                                                 bestMatch.getMetricReleaseTitle())));

  this.setTrueFalsePositive(key.toString(), bestMatch.getTrackId());
}
```

Um innerhalb des Prototyps die Qualität der einzelnen Ähnlichkeitsmetriken erfassen und abschließend bewerten zu können, wird in der *Reduce*-Methode zudem protokolliert, ob es sich bei der ermittelten besten Übereinstimmung tatsächlich um eine gültige Übereinstimmung (also um ein True-Positiv) handelt. Hierfür werden die RowKeys der beiden Datensätze verglichen. Stimmen diese überein, handelt es sich um eine gültige Übereinstimmung. Dies ist möglich, da beim Generieren der Testdaten die RowKeys der Originaldaten übernommen und nur die Attribute manipuliert wurden. Die Berechnung der Qualitätsmetriken Precision, Recall und F-Score (siehe Kapitel 2.4) sowie die Speicherung der Berechnungs-Ergebnisse im HDFS erfolgt abschließend in der *Cleanup*-Methode des Reducers.

5 Evaluation und Validation

5.1 Testkonzept

Die Grundlage der Messungen bilden die in HBase importierten Discogs-Daten vom 1. Dezember 2015, die im Kapitel 4.3 beschrieben und im Kapitel 8.1 auszugsweise dargestellt werden. Der Gesamtumfang der Daten beträgt circa 25 Gigabyte. Nach dem Import enthalten die HBase-Tabellen Informationen zu über vier Millionen Artisten, mehr als sechseinhalb Millionen Veröffentlichungen und circa 56 Millionen Liedern.

Da keine validierten Testdaten auf Basis der Discogs-Daten existieren, werden für die Durchführung der Benchmarks eigene Testdatensätze generiert. Die Erzeugung dieser Testdaten wird im nachfolgenden Kapitel detailliert beschrieben. Für die Qualitätsmessung werden insgesamt 1.000 Testdatensätze erzeugt.

Die Messungen werden auf Basis einer prototypischen Implementierung durchgeführt. Das zugrunde liegende Datenmodell der HBase-Tabellen sowie die Beschreibung des entwickelten Prototyps wird in den Kapiteln 4.4 und 4.5 ausführlich dargestellt. Das Ziel der Qualitätsmessung ist die Ermittlung der Zuordnungs-Qualität der verschiedenen Ähnlichkeitsmetriken. Hierfür kommen die bekannten Qualitäts-Kennzahlen Precision, Recall und F-Score zum Einsatz. Darüber hinaus wird die Ausführungsgeschwindigkeit der einzelnen Ähnlichkeitsmetriken gemessen. Hieraus lässt sich neben der Gesamtdauer des Zuordnungsprozesses auch die Anzahl der Berechnungen pro Sekunde innerhalb des Clusters ermitteln.

Die nachfolgenden Ähnlichkeitsmetriken werden untersucht:

- Zeichen-basiert
  - Damerau-Levenshtein-Distanz
  - Jaro-Distanz
  - Jaro-Winkler-Distanz
  - Levenshtein-Distanz
  - LongestCommonSubsequence
  - LongestCommonSubstring
  - Smith-Waterman-Distanz
  - Smith-Waterman-Gotoh-Distanz

- Token-basiert
  - Sørensen-Dice-Koeffizient (Tokenizer: Leerzeichen)
  - Sørensen-Dice-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 1)
  - Sørensen-Dice-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 2)
  - Sørensen-Dice-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 3)
  - Jaccard-Koeffizient (Tokenizer: Leerzeichen)
  - Jaccard-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 1)
  - Jaccard-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 2)
  - Jaccard-Koeffizient (Tokenizer: n-Gramme; n-Gramm-Größe: 3)
- Hybrid
  - Monge-Elkan (Tokenizer: Leerzeichen; interne Metrik: Smith-Waterman-Gotoh)

Zu beachten ist, dass für alle Messungen und somit auch für alle getesteten Ähnlichkeitsmetriken der in Kapitel 4.5 genannte Schwellwert von 80 Prozent für die drei Bereiche (Artist, Liedtitel und Releasetitel) genutzt wird. Wie bereits in Kapitel 2.1.4.3 erwähnt, beeinflusst die Wahl des Schwellwerts die Genauigkeit und somit die Qualität der Duplikaterkennung maßgeblich. Deshalb sind die in den nachfolgenden Kapiteln vorgestellten Ergebnisse im Hinblick auf die Nutzung des festen Schwellwertes von 80 Prozent zu betrachten. Die Ermittlung des optimalen Schwellwertes für jede Ähnlichkeitsmetrik würden den Rahmen dieser Abschlussarbeit bei weitem sprengen.

Die Durchführung der Messungen erfolgt auf dem in Kapitel 4.2 vorgestellten, fünf Knoten umfassenden Hadoop-Cluster.

Wie in Kapitel 2.1.4.2 erwähnt, ist es bei der Duplikaterkennung notwendig, jeden Testdatensatz mit jedem Referenzdatensatz abzugleichen. Dies führt bei der hier beschriebenen Qualitätsmessung mit 1.000 Testdatensätzen, circa 56 Millionen Referenzdatensätzen und 17 verschiedenen Metriken zu insgesamt mehr als 952 Milliarden durchzuführenden Vergleichen - bzw. mehr als 56 Milliarden Vergleichen pro Ähnlichkeitsmetrik. Die Anzahl der durchgeführten Vergleiche ist für eine bessere Einschätzung, Bewertung und Gegenüberstellung der ermittelten Ausführungsgeschwindigkeiten der einzelnen Ähnlichkeitsmetriken hilfreich.

5.2 Erzeugung der Testdaten

Um Testdaten zu erzeugen, wird eine bestimmte Anzahl an Original-Datensätzen zufällig ausgewählt. Diese werden anschließend mit einer zuvor festgelegten Wahrscheinlichkeit manipuliert: Ein Viertel der Daten verbleiben im Originalzustand, drei Viertel der Datensätze werden manipuliert. Die nachfolgende Abbildung zeigt die verschiedenen Manipulationen sowie deren Wahrscheinlichkeiten.

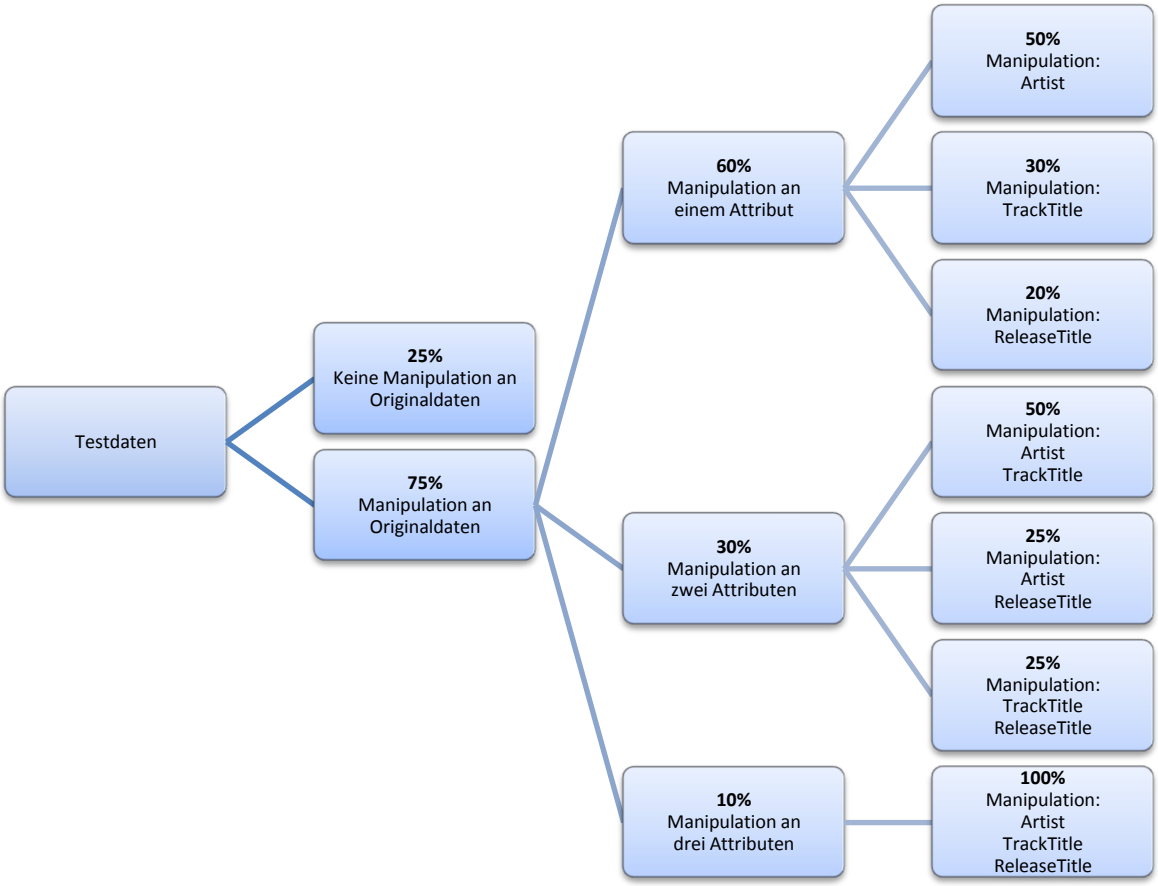

Abbildung 14: Manipulation der Testdaten

Um eine breite Palette an Testdaten zu generieren, werden alle drei Attribute - also sowohl Artist als auch Liedtitel und ReleaseTitel - manipuliert. Neben der Manipulation der einzelnen Attribute werden auch die verschiedenen Kombinationen der Attribute berücksichtigt. Die Details können der obigen Abbildung entnommen werden.

Im folgenden werden nun die verschiedenen Manipulationen im Einzelnen dargestellt.

Manipulationen am Artist. Discogs liefert umfangreiche Informationen zu mehr als vier Millionen Artisten. Dazu zählen u.a. auch so genannte Alternativ-Namen der Artisten. Hierbei handelt es sich um Namen, unter denen die Artisten ebenfalls bekannt sind. Beispielsweise als Abkürzungen oder mit zusätzlichen Bestandteilen, wie Mittelnamen oder Namenszusätzen. Diese Informationen werden für die Manipulation verwendet. Sollten für einen Artisten keine Alternativ-Namen bekannt sein, wird der bestehende Artist mit einem der folgenden Zusätze modifiziert:

- feat. Tobias Lindner
- ft. Toby
- featuring Toby
- vs. Toby Linde

Manipulationen am Liedtitel. Da für den Liedtitel keine offiziellen Alternativen bekannt sind, wird der bestehende Liedtitel mit einem der folgenden Zusätze modifiziert:

- (Cover Version)
- (Live Edition)
- (Radio Edit)
- (DJ Musix Remix)
- (Remix)
- (Studio Version)
- (Single)
- feat. Tobias Lindner
- ft. Toby
- featuring Toby
- vs. Toby Linde

Manipulationen am Albumtitel. Da für den Albumtitel keine offiziellen Alternativen bekannt sind, wird der bestehende Albumtitel mit einem der folgenden Zusätze modifiziert:

- (Acapella Version)
- (Digital EP)
- (Live CD)
- (Live Version)
- (Live at New York)
- (10 Years Special)
- (DJ Musix Remix)
- (Mash-Up Album)
- (Tribute Album)
- (Demo Tape)
- (Promo CD)

5.3 Ergebnisse der Qualitätsmessung

Auf den nachfolgenden Seiten werden die Precision-, Recall- und F-Score-Werte sowie die True-Positives, False-Positives und False-Negatives der getesteten Ähnlichkeitsmetriken tabellarisch und grafisch dargestellt. Eine Analyse und Beurteilung der Ergebnisse folgt im nächsten Kapitel. Zur besseren Übersicht werden unter 8.3.1 die Metriken nochmals nach ihrem Rang sortiert dargestellt.

Kategorie	Ähnlichkeitsmetrik	Precision	Recall	F-Score
Zeichen	Damerau-Levenshtein	100,00%	30,20%	46,39%
Zeichen	Jaro	99,01%	69,80%	81,88%
Zeichen	Jaro-Winkler	98,40%	85,90%	91,72%
Zeichen	Levenshtein	100,00%	30,20%	46,39%
Zeichen	LongestCommonSubsequence	99,34%	30,30%	46,44%
Zeichen	LongestCommonSubstring	100,00%	26,20%	41,52%
Zeichen	Smith-Waterman	93,58%	86,00%	89,63%
Zeichen	Smith-Waterman-Gotoh	93,36%	85,70%	89,36%
Token	Dice (Leerzeichen)	100,00%	36,10%	53,05%
Token	Dice (n-Gramme: n=1)	63,99%	51,00%	56,76%
Token	Dice (n-Gramme: n=2)	99,72%	35,60%	52,47%
Token	Dice (n-Gramme: n=3)	100,00%	31,70%	48,14%
Token	Jaccard (Leerzeichen)	100,00%	27,00%	42,52%
Token	Jaccard (n-Gramme: n=1)	97,30%	36,00%	52,55%
Token	Jaccard (n-Gramme: n=2)	100,00%	27,30%	42,89%
Token	Jaccard (n-Gramme: n=3)	100,00%	26,30%	41,65%
Hybrid	Monge-Elkan	96,95%	54,10%	69,45%

Tabelle 12: Precision, Recall und F-Score der getesteten Ähnlichkeitsmetriken (gruppiert)

Kategorie	Ähnlichkeitsmetrik	True-Positives	False-Positives	False-Negatives
Zeichen	Damerau-Levenshtein	302	0	698
Zeichen	Jaro	698	7	302
Zeichen	Jaro-Winkler	859	14	141
Zeichen	Levenshtein	302	0	698
Zeichen	LongestCommonSubsequence	303	2	697
Zeichen	LongestCommonSubstring	262	0	738
Zeichen	Smith-Waterman	860	59	140
Zeichen	Smith-Waterman-Gotoh	857	61	143
Token	Dice (Leerzeichen)	361	0	639
Token	Dice (n-Gramme: n=1)	510	287	490
Token	Dice (n-Gramme: n=2)	356	1	644
Token	Dice (n-Gramme: n=3)	317	0	683
Token	Jaccard (Leerzeichen)	270	0	730
Token	Jaccard (n-Gramme: n=1)	360	10	640
Token	Jaccard (n-Gramme: n=2)	273	0	727
Token	Jaccard (n-Gramme: n=3)	263	0	737
Hybrid	Monge-Elkan	541	17	459

Tabelle 13: True-Positives, False-Positives und False-Negatives der getesteten Ähnlichkeitsmetriken

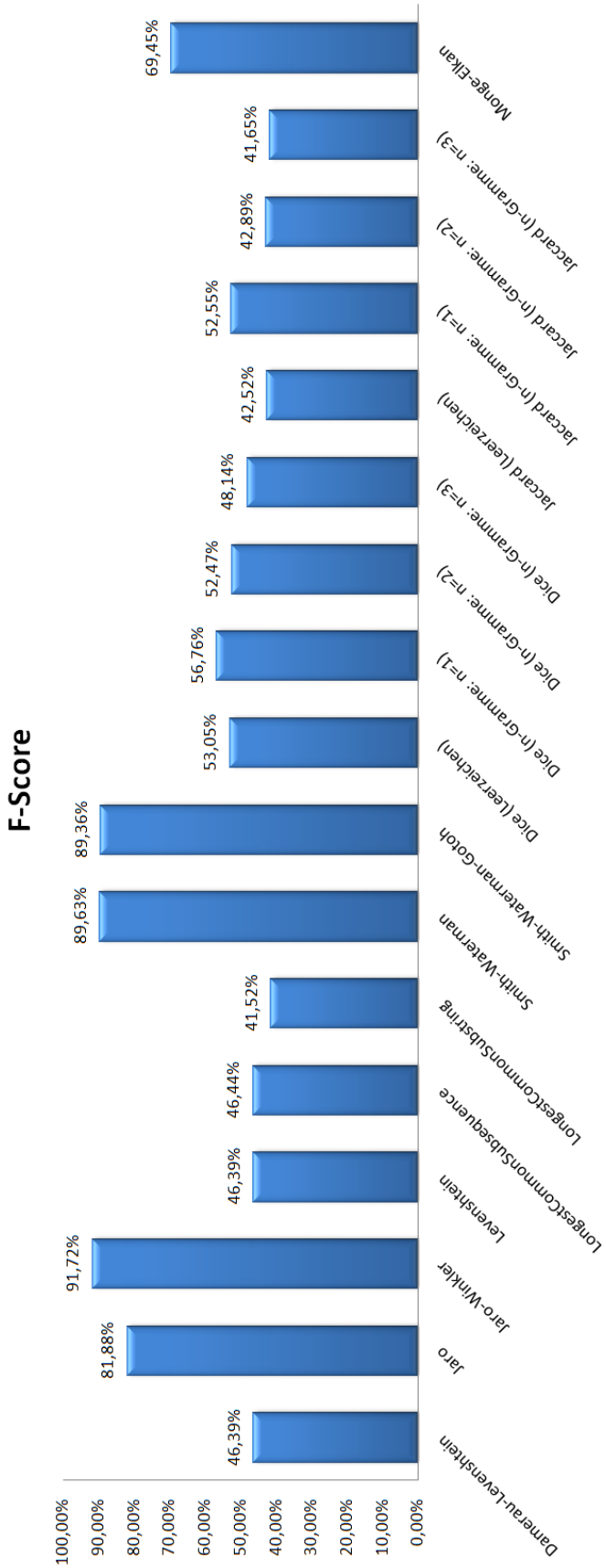

Abbildung 15: F-Score-Werte der getesteten Ähnlichkeitsmetriken (gruppiert)

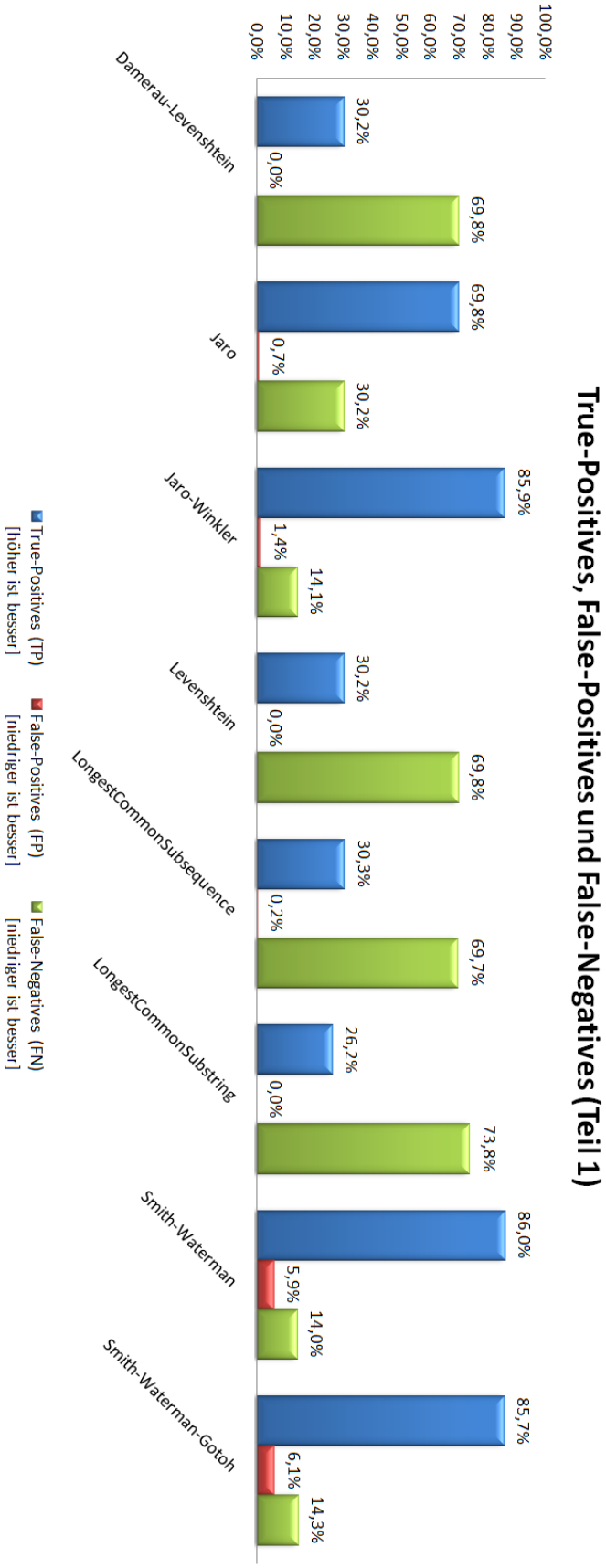

Abbildung 16: True-Positives, False-Positives und False-Negatives der getesteten Ähnlichkeitsmetriken (1)

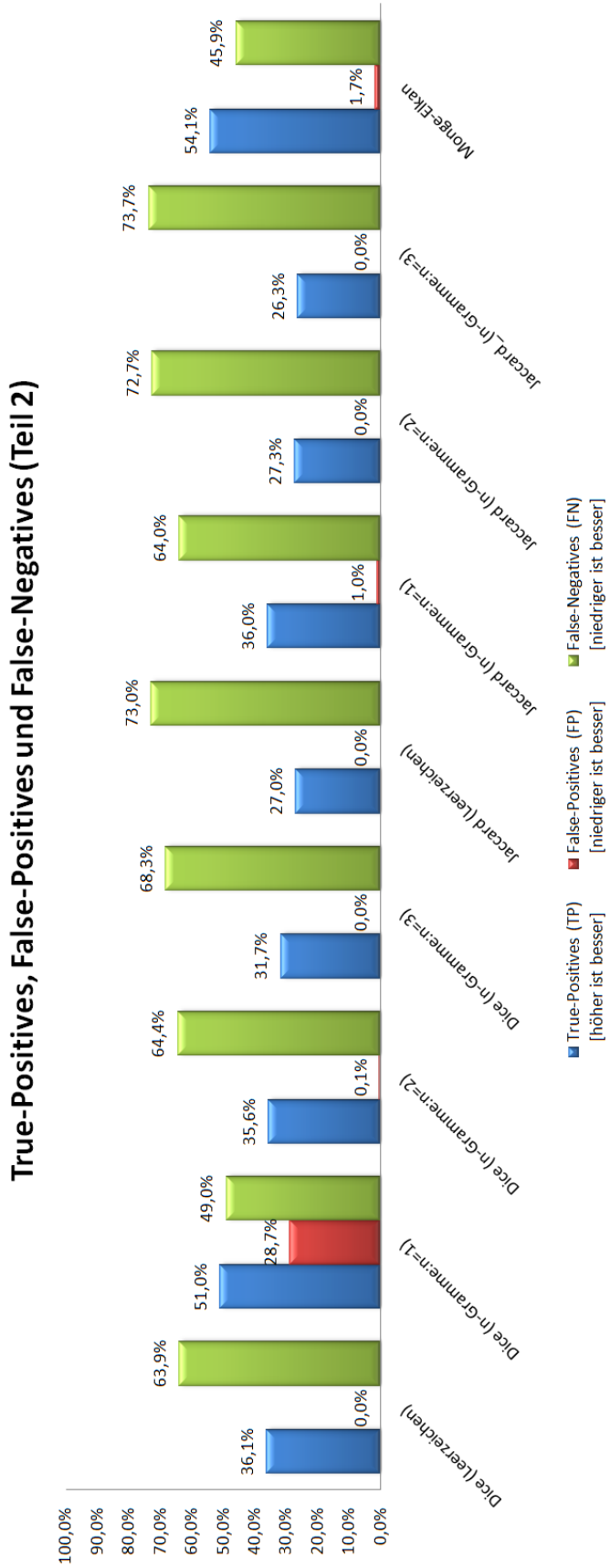

Abbildung 17: True-Positives, False-Positives und False-Negatives der getesteten Ähnlichkeitsmetriken (2)

Auf den nachfolgenden Seiten werden die Ausführungszeiten sowie die Anzahl der Berechnungen pro Sekunde der getesteten Ähnlichkeitsmetriken tabellarisch und grafisch dargestellt. Eine Analyse und Beurteilung der Ergebnisse folgt im nächsten Kapitel. Zur besseren Übersicht werden im Anhang unter 8.3.2 und 8.3.3 die Ergebnisse nochmals nach ihrem Rang sortiert dargestellt.

Kategorie	Ähnlichkeitsmetrik	Ausführungszeit	In Minuten	In Sekunden
Zeichen	Damerau-Levenshtein	05:10:41,129	311	18.641
Zeichen	Jaro	01:40:06,952	101	6.006
Zeichen	Jaro-Winkler	01:43:18,131	104	6.198
Zeichen	Levenshtein	02:56:48,572	177	10.608
Zeichen	LongestCommonSubsequence	01:37:48,505	98	5.868
Zeichen	LongestCommonSubstring	01:16:40,550	77	4.600
Zeichen	Smith-Waterman	27:23:06,936	1.644	98.586
Zeichen	Smith-Waterman-Gotoh	04:18:14,829	259	15.494
Token	Dice (Leerzeichen)	04:49:10,610	290	17.350
Token	Dice (n-Gramme: n=1)	04:27:05,384	268	16.025
Token	Dice (n-Gramme: n=2)	04:35:42,351	276	16.542
Token	Dice (n-Gramme: n=3)	03:54:30,675	235	14.070
Token	Jaccard (Leerzeichen)	03:00:42,750	181	10.842
Token	Jaccard (n-Gramme: n=1)	04:37:04,358	278	16.624
Token	Jaccard (n-Gramme: n=2)	04:11:35,539	252	15.095
Token	Jaccard (n-Gramme: n=3)	04:08:25,116	249	14.905
Hybrid	Monge-Elkan	11:58:10,466	719	43.090

Tabelle 14: Ausführungszeiten der getesteten Ähnlichkeitsmetriken (gruppiert)

Kategorie	Ähnlichkeitsmetrik	Im Cluster	Pro Knoten	Pro Map-Task
Zeichen	Damerau-Levenshtein	3.016.813	754.203	251.401
Zeichen	Jaro	9.361.952	2.340.488	780.163
Zeichen	Jaro-Winkler	9.073.187	2.268.297	756.099
Zeichen	Levenshtein	5.301.072	1.325.268	441.756
Zeichen	LongestCommonSubsequence	9.582.815	2.395.704	798.568
Zeichen	LongestCommonSubstring	12.223.930	3.055.982	1.018.661
Zeichen	Smith-Waterman	570.429	142.607	47.536
Zeichen	Smith-Waterman-Gotoh	3.629.391	907.348	302.449
Token	Dice (Leerzeichen)	3.241.200	810.300	270.100
Token	Dice (n-Gramme: n=1)	3.509.233	877.308	292.436
Token	Dice (n-Gramme: n=2)	3.399.565	849.891	283.297
Token	Dice (n-Gramme: n=3)	3.996.738	999.184	333.061
Token	Jaccard (Leerzeichen)	5.186.581	1.296.645	432.215
Token	Jaccard (n-Gramme: n=1)	3.382.795	845.699	281.900
Token	Jaccard (n-Gramme: n=2)	3.725.392	931.348	310.449
Token	Jaccard (n-Gramme: n=3)	3.772.986	943.247	314.416
Hybrid	Monge-Elkan	1.305.087	326.272	108.757

Tabelle 15: Anzahl der Berechnungen pro Sekunde der getesteten Ähnlichkeitsmetriken (gruppiert)

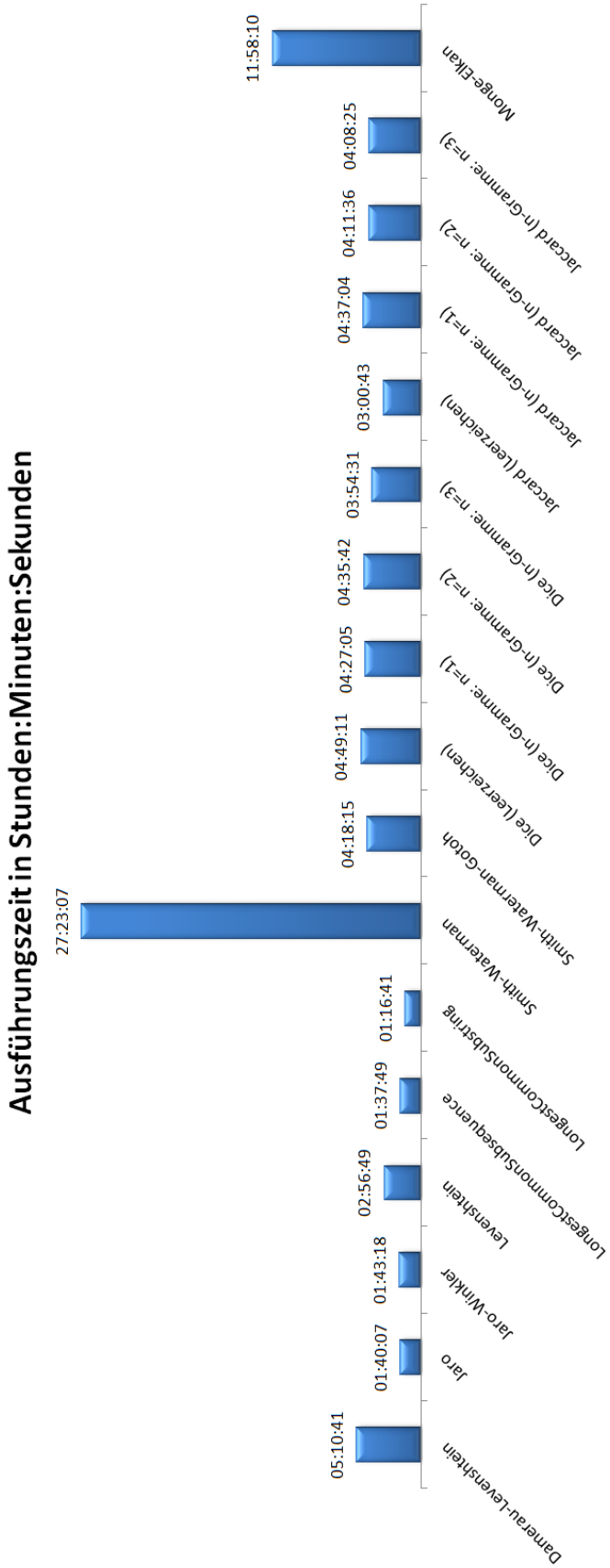

Abbildung 18: Ausführungszeiten der getesteten Ähnlichkeitsmetriken (gruppiert)

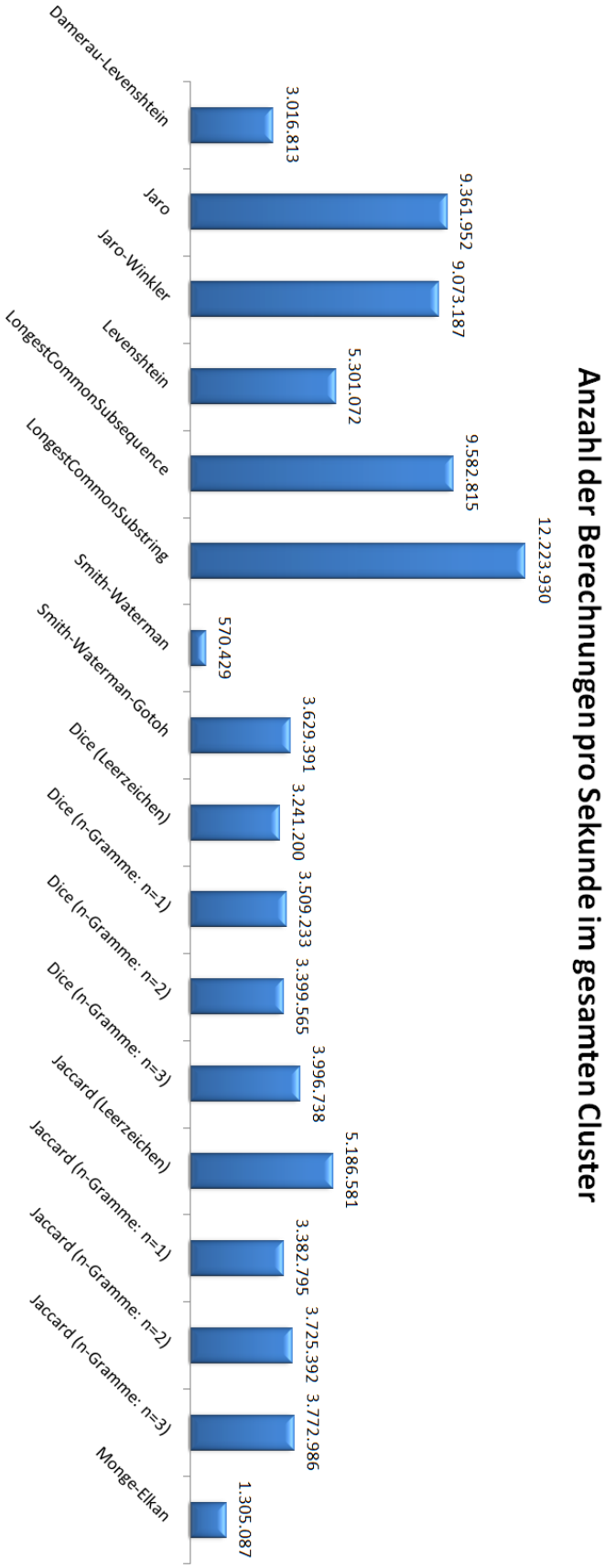

Abbildung 19: Anzahl der Berechnungen pro Sekunde der getesteten Ähnlichkeitsmetriken (gruppiert)

5.4 Analyse und Beurteilung der Ergebnisse

In diesem Kapitel werden die ermittelten und im vorherigen Kapitel tabellarisch und grafisch dargestellten Ergebnisse analysiert und beurteilt. Wie bereits erwähnt, ist bei der Betrachtung der Ergebnisse darauf zu achten, dass ein fester Schwellwert von 80 Prozent für alle Ähnlichkeitsmetriken verwendet wurde. Da die Wahl des Schwellwerts die Genauigkeit und somit die Qualität der Duplikaterkennung maßgeblich beeinflusst, kann es bei der Verwendung eines anderen Schwellwerts bei den einzelnen Metriken zu abweichenden Ergebnissen (sowohl besseren als auch schlechteren) kommen.

Beurteilung hinsichtlich der Zuordnungsqualität. Mit einem F-Score von knapp 92 Prozent ist die Jaro-Winkler-Distanz auf Platz eins der getesteten Ähnlichkeitsmetriken. Dicht gefolgt von der Smith-Waterman- und Smith-Waterman-Gotoh-Distanz mit jeweils knapp 90 Prozent. Auf Platz vier und fünf folgen die Jaro-Distanz mit knapp 82 Prozent sowie die Monge-Elkan-Ähnlichkeit mit knapp 70 Prozent. Die weiteren zwölf getesteten Metriken bzw. Metrik-Ausprägungen erzielen nur F-Score-Werte um 50 Prozent oder darunter. Diese, zum Teil recht niedrigen Werte könnten jedoch mit dem gewählten Schwellwert zusammenhängen.

Werden alle getesteten Metriken bzgl. ihres Rangs beurteilt, fällt auf, dass es sich bei den oberen Plätzen nur um Zeichen-basierte Metriken handelt, während sich die Token-basierten Metriken zusammen mit weiteren Zeichen-basierten Metriken die hinteren Plätze teilen. Die hybride Monge-Elkan-Ähnlichkeit positioniert sich genau dazwischen.

Die hohen Unterschiede bei den Zeichen-basierten Metriken lassen sich vermutlich durch die zum Teil unterschiedlichen Arbeitsweisen der verschiedenen Ähnlichkeitsmetriken erklären: Metriken mit hohen F-Score-Werten (u.a. Jaro-Winkler und Smith-Waterman-Gotoh) berücksichtigen bei der Ermittlung der Ähnlichkeit neben einzelnen Zeichen auch ganze Zeichenfolgen sowie Präfixe, Suffixe und Abkürzungen. Im Gegensatz dazu arbeiten zum Beispiel die Levenshtein- und Damerau-Levenshtein-Distanz nur mit einzelnen Zeichen und LongestCommonSubsequence nur mit Zeichenfolgen. Da es sich bei den manipulierten Testdaten nicht um einzelne Zeichendreher oder Rechtschreibfehler handelt, sondern Namen und Bezeichnungen vertauscht, abgekürzt oder angehängt werden, erzielen die "umfangreicheren" Metriken deutlich bessere Ergebnisse.

Bei den Token-basierten Ähnlichkeitsmetriken ist eine Auffälligkeit bei den n-Gramm-Ausprägungen zu erkennen: Mit zunehmender n-Gramm-Größe sinkt der F-Score-Wert. Sowohl beim Jaccard- als auch beim Sørensen-Dice-Koeffizienten. Dies hat vermutlich folgenden Zusammenhang: Die Metriken ermitteln die Ähnlichkeit anhand von Token-Mengen. Token mit der Länge eins (entspricht der

Betrachtung einzelner Buchstaben) führen zu einer höheren Anzahl an gemeinsamen Token als beispielsweise Tri-Gramme (Betrachtung von drei aufeinanderfolgenden Buchstaben). Die höhere Anzahl der übereinstimmenden Token bei n-Grammen der Länge eins kann jedoch auch zu einer höheren Anzahl an fälschlich erkannten Duplikaten führen. Dies ist vor allem beim Sørensen-Dice-Koeffizienten mit n-Grammen der Länge eins sehr gut an der stark erhöhten Anzahl an False-Positives bzw. der niedrigen Precision zu erkennen (siehe Tabelle 12 und Tabelle 13 auf Seite 72). Die generell niedrigen F-Score-Werte bei den Token-basierten Metriken könnten mit dem gewählten Schwellwert zusammenhängen.

Werden die Precision- und Recall-Werte näher betrachtet, fällt folgendes auf: Metriken, die nur einen geringen Anteil der tatsächlichen Duplikate erkennen (also einen niedrigen Recall-Wert aufweisen), sind dabei sehr treffsicher und haben einen Precision-Wert von 100 Prozent. Es werden also keine Datensätze fälschlicherweise als Duplikate ausgewiesen. Die Anzahl der gefundene Duplikate ist dafür jedoch sehr gering. Dies hängt mit dem verwendeten Schwellwert zusammen. Diese Ergebnisse bestätigen die Aussage von [ABEM10] bzgl. dem Zusammenhang von Precision und Recall (siehe Kapitel 2.4).

Beurteilung hinsichtlich der Ausführungsgeschwindigkeit. Mit mehr als zwölf Millionen Vergleichs-Berechnungen pro Sekunde im gesamten Cluster ist LongestCommonSubstring auf Platz eins, gefolgt von LongestCommonSubsequence sowie der Jaro- und Jaro-Winkler-Distanz mit jeweils über neun Millionen Vergleichsberechnungen. Das Schlusslicht bildet die Smith-Waterman-Distanz mit etwas mehr als einer halben Million Berechnungen pro Sekunde. Sie ist somit um den Faktor 24 langsamer als der Erstplatzierte. Dies liegt vor allem daran, dass die Metrik für das Auffinden der längsten gemeinsamen Sequenz konzipiert wurde. Dies kommt z.B. bei DNA-Sequenzen zum Einsatz und ist sehr rechenintensiv. Erste Ansätze (z.B. [StB13]) versuchen den Algorithmus zu beschleunigen.

Werden alle getesteten Metriken bzgl. ihres Rangs beurteilt sowie miteinander verglichen, fällt auf, dass die Zeichen-basierten Ähnlichkeitsmetriken zu einem großen Teil eine wesentlich schnellere Ausführungsgeschwindigkeit aufweisen als die Token-basierten Metriken. Dies hängt damit zusammen, dass letztere Metriken die zu vergleichenden Zeichenketten zunächst in einzelne Token aufsplitten und anschließend rechenintensiv Vektor-, Mengen- oder Geometrie-Operationen durchführen.

Die hybride Metrik Monge-Elkan, welche bei vorliegenden Messungen die Zeichenketten zunächst in einzelne Wörter zerlegt und anschließend intern die Smith-Waterman-Gotoh-Metrik zum Vergleich der Wörter anwendet, positioniert sich bzgl. ihrer Ausführungsgeschwindigkeit wie erwartet.

Beurteilung hinsichtlich der Zuordnungsqualität und der Ausführungsgeschwindigkeit. Werden die getesteten Ähnlichkeitsmetriken hinsichtlich der Zuordnungsqualität in Kombination mit der Ausführungsgeschwindigkeit beurteilt, geht die Jaro-Winkler-Distanz als klarer Gewinner hervor. Eine mögliche Alternative ist die etwas schnellere dafür jedoch qualitativ schlechtere Jaro-Distanz.

Die beiden Metriken Smith-Waterman- und Smith-Waterman-Gotoh-Distanz erzielen zwar einen sehr hohen F-Score, sind durch ihre langsame Ausführungsgeschwindigkeit (vor allem Smith-Waterman) für einen produktiven Einsatz nicht geeignet. Die schnellsten Metriken LongestCommonSubstring und LongestCommonSubsequence können durch ihre niedrigen F-Score-Werte nur bedingt überzeugen. Die Ermittlung des optimalen Schwellwerts bei diesen beiden Metriken könnte die Zuordnungsqualität jedoch steigern. Dadurch könnten diese für einen produktiven Einsatz in die engere Auswahl rücken. Die häufig verwendete Levenshtein-Distanz liegt in beiden Disziplinen lediglich im Mittelfeld.

Die getesteten Token-basierten Metriken landen sowohl bei der Zuordnungsqualität als auch bei der Geschwindigkeit nur im Mittelfeld. Die Ermittlung des optimalen Schwellwerts könnte hier jedoch ebenfalls zu einer Steigerung der Qualität führen. Bezogen auf die aktuell vorliegenden Ergebnisse sind die Token-basierten Metriken für einen produktiven Einsatz allerdings eher nicht geeignet.

Auch die hybride Monge-Elkan-Ähnlichkeit kann mit den getesteten Einstellungen - vor allem aufgrund der sehr langsamen Ausführungsgeschwindigkeit - nicht überzeugen. Der Einsatz einer anderen internen Zeichen-basierten Metrik könnte jedoch zu besseren Ergebnissen führen - sowohl bzgl. der Qualität als auch hinsichtlich der Geschwindigkeit.

6 Zusammenfassung und Ausblick

6.1 Fazit

Diese Abschlussarbeit wurde mit dem Ziel erstellt, innerhalb des *Forschungsprojekts MPI* der *Hochschule München* ein neues Vorgehen zur automatischen und manuellen Zuordnung von Musiknutzungsmeldungen zu erarbeiten und anschließend prototypisch zu untersuchen. Hierfür wurde ein erstes Konzept für ein horizontal skalierbares, BigData-gestütztes Empfehlungssystem zur urheberrechtlichen Zuordnung von Musiknutzungen erstellt sowie mögliche Lösungs- bzw. Umsetzungsansätze aufgezeigt und näher betrachtet. Durch die Implementierung eines Prototypen konnte zudem die Umsetzbarkeit sowie Leistungsfähigkeit des erarbeiteten Konzepts erprobt werden. Durch die theoretische Betrachtung von diversen Ähnlichkeitsmetriken sowie die anschließende Implementierung von ausgewählten Algorithmen zur Ähnlichkeitsbestimmung konnten sowohl Potenziale als auch Grenzen der verschiedenen Metriken aufgezeigt werden. Überraschendes Ergebnis ist, dass bei den Qualitäts- und Performance-Messungen in dieser Arbeit die sehr häufig verwendete und auch bei der GEMA zum Einsatz kommende Levenshtein-Distanz nur im Mittelfeld gelandet ist. Dies bedeutet, dass bereits durch den Einsatz anderer Metriken (beispielsweise die Verwendung der Jaro-Winkler-Distanz) bessere Ergebnisse möglich sind - sowohl bei der Zuordnungsqualität als auch bei der Verarbeitungsgeschwindigkeit.

Wie bereits im Kapitel 3.3 genannt, ist die Nutzung eines Hadoop-Clusters nur ein möglicher Ansatz zur Implementierung des erstellten Konzepts. Auch der Einsatz von Apache Blur (Kapitel 3.3.3) oder ElasticSearch (Kapitel 3.3.4) sowie die Anwendung von maschinellem Lernen und künstlicher Intelligenz mit Hilfe von beispielsweise Apache Mahout (Kapitel 3.3.5) sind denkbare und vielversprechende Lösungsmöglichkeiten, die näher untersucht werden können.

Zudem ist beim Einsatz von Ähnlichkeitsmetriken der verwendete Schwellwert maßgeblich für die Qualität der Duplikaterkennung. Die Ergebnisse der Messungen in dieser Arbeit zeigen dies deutlich: Der in der MDM-Dokumentation der GEMA fest vorgeschriebene Schwellwert von 80 Prozent führt nur bei einem Bruchteil der getesteten Metriken zu qualitativ hochwertigen und brauchbaren Ergebnissen. Die Ermittlung des optimalen Schwellwerts für die verschiedenen Ähnlichkeitsmetriken kann sowohl für das Forschungsprojekt MPI und deren Projektpartner als auch für andere Forschungsprojekte und für die Wirtschaft von großem Interesse sein und könnte daher weiter untersucht werden.

6.2 Ausblick

Produktiver Einsatz des erarbeiteten Konzepts. In dieser Abschlussarbeit kam bei der prototypischen Implementierung sowie bei den Qualitäts- und Performance-Messungen ein fünf Knoten umfassendes Hadoop-Cluster (ein Master-Node, vier Worker-/Data-Nodes) zum Einsatz. Bei Hadoop ist, bedingt durch die massiv parallele Verarbeitung der Daten, von einer annähernd proportionalen Steigerung der Verarbeitungsgeschwindigkeit zur Anzahl der zur Verfügung stehenden Knoten auszugehen. Dies bedeutet, dass sich beim Einsatz von beispielsweise 400 Worker-/Data-Nodes die gemessenen Ausführungszeiten um den Faktor 100 reduzieren würden. Dadurch ist es bei ausgewählten Ähnlichkeitsmetriken (z.B. der Jaro-Winkler-Distanz) möglich, innerhalb weniger Minuten mehrere tausend Datensätze auf Übereinstimmung mit dem umfangreichen Referenzdatenbestand zu prüfen.

Das hierfür notwendige Hadoop-Cluster könnte dabei über mehrere Wege bereit gestellt werden: Neben dem Betrieb eines eigenen, GEMA-internen Clusters könnten auch extern gehostete Hadoop-Cluster zum Einsatz kommen. Denkbar wäre hierbei beispielsweise das Hadoop-Cluster des Forschungsprojekts MPI der Hochschule München [CCWI] sowie die (stundenweise) Nutzung einer Cloud-basierten Lösung wie zum Beispiel Amazon Elastic MapReduce (Amazon EMR) [EMR].

Ausweitung des Konzepts auf andere Anwendungsbereiche. Das in dieser Abschlussarbeit entwickelte Konzept sowie die Erkenntnisse der prototypischen Implementierung und die Ergebnisse der Qualitäts- und Performance-Messungen lassen sich nicht nur innerhalb der Musikbranche erfolgreich nutzen und anwenden. Wie bereits in der Einleitung beschrieben, ist die Zusammenführung von Daten aus verschiedenen Quellen sowie die Sicherung der Datenqualität ein wichtiger Bestandteil vieler heutiger IT-Systeme: Vom Adressabgleich im E-Commerce (z.B. für die Identifizierung von Neu- und Bestandskunden anhand der Adresse) über die korrekte Zusammenführung von Patientendaten (z.B. [SBR09]) bis hin zur Auswertung von demografischen Studien (z.B. [Sch10]). Dies sind nur einige Anwendungsfälle für die der erarbeitete Lösungsansatz interessant sein könnte.

7 Literaturverzeichnis

[ABEM10] Apel, Detlef; Behme, Wolfgang; Eberlein, Rüdiger; Merighi, Christian: *Datenqualität erfolgreich steuern – Praxislösungen für Business-Intelligence-Projekte*. 2., vollständig überarbeitete und erweiterte Auflage, Carl Hanser Verlag, 2010

[ASP] SourceCode Double-Metaphone by Lawrence Philips
http://aspell.net/metaphone/dmetaph.cpp
Zugriff am: 11. Januar 2016

[AMO] SourceCode Metaphone 3 by Lawrence Philips
http://www.amorphics.com
Zugriff am: 11. Januar 2016

[BaR99] Baeza-Yates, Ricardo; Ribeiro-Neto, Berthier: *Modern Information Retrieval.* Addison-Wesley, Harlow, 1999

[Bat95] Bates, M: *Models of natural language understanding.* Proceedings of the National Academy of Sciences of the United States of America 92 (22), pp. 9977–9982, 1995

[Bel57] Bellman, Richard: *Dynamic Programming.* Princeton University Press, 1957

[Bet08] Bethlehem, Jelke: *Surveys without questions.* In: E. D. de Leeuw, J. D. Hox und D. A. Dillman (Hg.): International handbook of survey methodology, New York: Erlbaum, pp. 500-511, 2008

[BHR00] Bergroth, L.; Hakonen, H.; Raita, T.: *A Survey of Longest Common Subsequence Algorithms.* In: SPIRE (String Processing and Information Retrieval), pp. 39-48, 2000

[Blur] Apache Blur
https://incubator.apache.org/blur/
Zugriff am: 15. Februar 2016

[Bri92] Brill, Eric: *A simple rule-based part-of-speech tagger.* In Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP-92), pp. 152-155, 1992

[CDGHW06] Chang, Fay; Dean,Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C.; Wallach, Deborah A.; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E.: *Bigtable: A Distributed Storage System for Structured Data.* In: OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006

[ChL04] Charras, Christian; Lecroq, Thierry: *Handbook of Exact String Matching Algorithms.* King's College Publications, 2004

[CC13] Einführung in Hadoop – Die wichtigsten Komponenten von Hadoop
https://blog.codecentric.de/2013/08/einfuhrung-in-hadoop-die-wichtigsten-komponenten-von-hadoop-teil-3-von-5/
Zugriff am: 8. Februar 2016

[CCWI] Hadoop Research Cluster Initiative des Competence Center Wirtschaftsinformatik (CCWI) der Hoschule München
http://www.wirtschaftsinformatik-muenchen.de/?page_id=1308
Zugriff am: 14. März 2016

[Coh98] Cohen, William: *Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity.* In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998

[Dam64] Damerau, Fred J.: *A technique for computer detection and correction of spelling errors*. In: Communications of the ACM. 7, Nr. 3, pp. 171-176, 1964

[DeG04] Dean, Jeffrey; Ghemawat, Sanjay: *MapReduce: Simplified Data Processing on Large Clusters.* Google Labs, In: OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004

[DGI07] Informationsqualität nach Deutsche Gesellschaft für Informations- und Datenqualität
https://web.archive.org/web/20140901172858/http://88.198.68.171:8080
/confluence/download/attachments/111411219/15.+IQ+Dimensionen.pdf
?version=1&modificationDate=1301666789000
Zugriff am: 4. November 2015

[Dic45] Dice, Lee R.: *Measures of the Amount of Ecologic Association Between Species.* Ecology, Volume 26, Issue 3, pp. 297-302, 1945

[Dis15] Discogs - Datenbank und Marktplatz für Musik auf Schallplatte, CD, Kassette und anderen Formaten
http://www.discogs.com
Zugriff am: 2. November 2015

[Elastic] Elasticsearch
https://www.elastic.co/products/elasticsearch
Zugriff am: 15. Februar 2016

[EMR] Amazon Elastic MapReduce (EMR)
https://aws.amazon.com/de/elasticmapreduce/
Zugriff am: 14. März 2016

[FeS69] Fellegia, Ivan P.; Suntera, Alan B.: *A Theory for Record Linkage.* In: Journal of the American Statistical Association, Volume 64, Issue 328, pp. 1183-1210, 1969

[Fri07] Friedl, Jeffrey E. F.: *Reguläre Ausdrücke*, 3. Auflage, O'Reilly Verlag, Oktober 2007

[Gal99] Gallagher, E.D.: *COMPAH Documentation.* University of Massachusetts, Boston, 1999

[GhGL03] Ghemawat, Sanjay; Gobioff, Howard; Leung, Shun-Tak: *The Google File System.* Google Labs, In: 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003

[GoB] Google Books: n-Gram Viewer
https://books.google.com/ngrams
Zugriff am: 14. Dezember 2015

[GoS] Google Scholar: Natural Language Processing
https://scholar.google.de/scholar?q=natural+language+processing
Zugriff am: 16. November 2015

[Got82] Gotoh, Osamu: *An Improved Algorithm for Matching Biological Sequences.* In: Journal of Molecular Biology, 162, pp.705-708, 1982

[Ham50] Hamming, Richard W.: *Error-detecting and error-correcting codes.* In: Bell System Technical Journal, XXIX (2), 1950, pp. 147-160

[Hir75] Hirschberg, D. S.: *A linear space algorithm for computing maximal common subsequences*. In: Communications of the ACM. 18, Nr. 6, pp. 341-343, 1975

[HuS77] Hunt, James W.; Szymanski, Thomas G.: *A Fast Algorithm for Computing Longest Common Subsequences.* In: Programming Techniques, ACM, Volume 20, Number 5, pp. 350-353, 1977

[IBM] Best practices for loading data in HBase
https://www.ibm.com/support/knowledgecenter/SSWSR9_11.4.0/
com.ibm.swg.im.mdmhs.pmebi.doc/topics/bestpractice_loadhbase.html
Zugriff am: 18. Februar 2016

[IDC] The Digital Universe In 2020: An EMC-sponsored study by IDC
http://germany.emc.com/infographics/digital-universe-business-infographic.htm
http://germany.emc.com/leadership/digital-universe/index.htm
Zugriff am: 2. November 2015

[Jac01] Jaccard, Paul: *Étude comparative de la distribution florale dans une portion des Alpes et des Jura.* In: Bulletin de la Société Vaudoise des Sciences Naturelles 37, pp. 547-579, 1901

[Jar89] Jaro, M. A.: *Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida.* In: Journal of the American Statistical Association, Volume 84, Issue 406, pp. 414-420, 1989

[Jar95] Jaro, M. A.: *Probabilistic linkage of large public health data files.* In: Statistics in Medicine 14 (5-7), pp. 491-498, 1995

[JuM08] Jurafsky, Daniel; Martin, James H.: *Speech and Language Processing, 2nd edition.* Pearson Prentice Hall, 2008

[Jur51] Juran, J. M.: *Quality control handbook.* New York, McGraw-Hill, 1951

[KeM02] Kedad, Z.; Metais, E.: *Ontology-based data cleaning.* In: Natural Language Processing and Information Systems, 6th International Conference on Applications of Natural Language to Information Systems, Sweden, 2002

[KMK03] Kondrak, Grzegorz; Marcu, Daniel; Knight, Kevin: *Cognates Can Improve Statistical Translation Models.* Proceedings of HLT-NAACL 2003: Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, pp. 46-48, 2003

[Kro93] Krovetz, R.: *Viewing Morphology as an Inference Process*. In: Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 191–203, 1993

[Lev66] Levenshtein, Vladimir I.: *Binary codes capable of correcting deletions, insertions, and reversals.* In: Doklady Akademii Nauk SSSR. 163, Nr. 4, 1965, pp. 845-848 (Russisch, Englische Übersetzung in: Soviet Physics Doklady, 10(8) pp. 707-710, 1966)

[LeW71] Levandowsky, Michael; Winter, David: *Distance between sets.* In: Nature 234, Volume 5, pp. 34-35, 1971

[Lip99] Lipkus, Alan H: *A proof of the triangle inequality for the Tanimoto distance.* In: J Math Chem, Volume 26, Issue 1-3, pp. 263-265, 1999

[Lov68] Lovins, Julie Beth: *Development of a stemming algorithm.* In: Mechanical Translation and Computational Linguistics Vol. 11 no. 2, pp. 22–31, 1968

[Lucene] Apache Lucene
https://lucene.apache.org
Zugriff am: 15. Februar 2016

[Mahout] Apache Mahout
https://mahout.apache.org
Zugriff am: 15. Februar 2016

[MaP80] Masek, William J.; Paterson, Michael S.: *A faster algorithm computing string edit distances.* In: Journal of Computer and System Sciences, Volume 20, Issue 1, pp 18-31, 1980

[MoE96] Monge, Alvaro E.; Elkan, Charles P.: *The field matching problem: Algorithms and applications.* In: Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, pp. 267–270, 1996

[MPI] Forschungsprojekt MPI - Massively Adhoc Processing in the Internet des Competence Center Wirtschaftsinformatik der Hochschule München
http://www.wirtschaftsinformatik-muenchen.de/?page_id=540
Zugriff am: 2. November 2015

[MRS08] Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich: *Introduction to Information Retrieval.* Cambridge University Press, 2008

[MuB15] MusicBrainz is an open music encyclopedia that collects music metadata and makes it available to the public
https://www.musicbrainz.org
Zugriff am: 2. November 2015

[MusicAPI] Liste von Musik-APIs
http://www.programmableweb.com/news/25-music-apis/2008/02/21
Zugriff am: 2. November 2015

[NaH10] Naumann, Felix; Herschel, Melanie: *An Introduction to Duplicate Detection.* Morgan and Claypool, 2010

[Nav99] Navarro, Gonzalo: *A Guided Tour to Approximate String Matching.* In: ACM Computing Surveys, 1999

[NeW70] Needleman, Saul B.; Wunsch, Christian D.: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* In: Journal of Molecular Biology. 48, pp. 443-453, 1970

[NKAJ59] Newcombe, H. B.; Kennedy, J. M.; Axford, S. J.; James, A. P.: *Automatic Linkage of Vital Records.* In: Science Vol. 130 no. 3381 pp. 954-959, 1959

[NLPG1] The Stanford Natural Language Processing Group
http://nlp.stanford.edu
Zugriff am: 16. November 2015

[NLPG2] The Berkeley NLP Group
http://nlp.cs.berkeley.edu
Zugriff am: 16. November 2015

[NLPG3] Microsoft Research: Natural Language Processing
http://research.microsoft.com/en-us/groups/nlp/
Zugriff am: 16. November 2015

[NLPG4] The University of Leipzig NLP Group
http://asv.informatik.uni-leipzig.de/en/
Zugriff am: 16. November 2015

[Ode56] Odell, Margaret King: *The profit in records management.* In: Systems, Volume 20, New York, 1956

[Phi90] Philips, Lawrence: Hanging on the Metaphone. In: Computer Language, Vol. 7, No. 12 (December), 1990

[Por80] Porter, Martin: *An algorithm for suffix stripping.* In: Program Vol. 3 no. 14, pp. 130-137, 1980

[Pos69] Postel, Hans Joachim: Die Kölner Phonetik. Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. In: IBM-Nachrichten, 19. Jahrgang, pp. 925-931, 1969

[SBR09] Schnell, Rainer; Bachteler, Tobias; Reiher, Jörg: *Privacy-preserving record linkage using Bloom filters*. In: *BMC Medical Informatics and Decision Making* 9 (41), 2009

[Sch10] Schnell, Rainer: *Record Linkage from a Technical Point of View*. In: German Data Forum (RatSWD) (ed.): Building on progress: Expanding the research infrastructure for the social, economic, and behavioral sciences (Vol. 1), Opladen: Budrich UniPress, pp. 531-545, 2010

[Sch69] Schönpflug, Wolfgang: *N-Gramm-Häufigkeiten in der deutschen Sprache. I. Monogramme und Digramme.* In: Zeitschrift für experimentelle und angewandte Psychologie XVI, pp. 157-183, 1969

[ScH94] Schmid, Helmut: *Probabilistic part-of-speech tagging using decision trees.* In: Proceedings of the International Conference on New Methods in Language Processing 1994, 1994

[ScU01] Schöning, Uwe: *Algorithmik.* 13. Ausgabe, Spektrum Akademischer Verlag, 2001

[Sim1] Die bei der Implementierung verwendete SimMetrics-Bibliothek
https://github.com/Simmetrics/simmetrics
Zugriff am: 8. Februar 2016

[Sim2] SimMetrics-Bibliothek von Nicolas Bohorquez Gutierrez
https://github.com/nickmancol/simmetrics
Zugriff am: 8. Februar 2016

[Sim3] SimMetrics-Bibliothek der UK Sheffield University
https://sourceforge.net/projects/simmetrics/
Zugriff am: 8. Februar 2016

[Sin01] Singhal, Amit: *Modern Information Retrieval: A Brief Overview.* In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Volume 24, Issue 4, pp. 35-43, 2001

[SmW81] Smith, Temple F.; Waterman, Michael S.: *Identification of common molecular subsequences.* In: Journal of Molecular Biology, 147, pp. 195-197, 1981

[Sor48] Sørensen, T.: *A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons.* In: Kongelige Danske Videnskabernes Selskab, Volume 5, Issue 4, pp. 1-34, 1948

[StB13] Strengholt, B.; Brobbel, M.: *Acceleration of the Smith-Waterman algorithm for DNA sequence alignment using an FPGA platform.* Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, 2013

[STTS95] Stuttgart-Tübingen-Tagset (STTS), 1995/1999
http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html
Zugriff am: 18. November 2015

[VMW] Durchführen von Live-Migrationen mit vMotion von VMware
https://www.vmware.com/de/products/vsphere/features/vmotion
Zugriff am: 24. Februar 2016

[War12] Wartala, Ramon: *Hadoop: Zuverlässige, verteilte und skalierbare Big-Data-Anwendungen.* Open Source Press, München 2012

[WaS96] Wang, R.; Strong, D.: *Beyond Accuracy: What Data Quality Means to Data Consumers.* Journal of Management Information Systems 12 (4): pp. 5-34, 1996

[Win90] Winkler, William E.: *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.* In: Proceedings of the Section on Survey Research Methods (American Statistical Association), pp. 354–359, 1990

[WiT91] Winkler, William E.; Thibaudeau, Yves: *An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 U.S. Census.* Technical report, US Bureau of the Census, 1991

[WSB76] Waterman, Michael S.; Smith, Temple F.; Beyer, W.A.: *Some Biological Sequence Metrics.* In: Advances in Math., Volume 20, Issue 4, pp. 367-387, 1976

[Xmappr] A simple XML-to-Java Mapper
https://github.com/PeterKnego/xmappr
Zugriff am: 17. Februar 2016

8 Anhang

8.1 Aufbau und Inhalt der discogs.com-Dateien

8.1.1 Aufbau und Inhalt der "Artists"-Datei von discogs.com

Nachfolgend wird der Aufbau und Inhalt der XML-Datei mit den Informationen über die bei discogs.com hinterlegten Artisten auszugsweise dargestellt. Es handelt sich dabei um einen Auszug aus der in dieser Masterarbeit verwendeten XML-Datei vom 1. Dezember 2015.

```
<artists>
  <artist>
    ...
  </artist>
  ...
  <artist>
    <images>
      <image width="500" uri150="" type="primary" uri="" height="630"/>
      <image width="219" uri150="" type="secondary" uri="" height="288"/>
      <image width="380" uri150="" type="secondary" uri="" height="280"/>
      <image width="399" uri150="" type="secondary" uri="" height="370"/>
      <image width="200" uri150="" type="secondary" uri="" height="199"/>
      <image width="550" uri150="" type="secondary" uri="" height="436"/>
      <image width="600" uri150="" type="secondary" uri="" height="661"/>
      <image width="461" uri150="" type="secondary" uri="" height="270"/>
    </images>
    <id>3</id>
    <name>Josh Wink</name>
    <realname>Joshua Winkelman</realname>
    <profile>After forming [l=Ovum Recordings] as an independent label in October 1994
with former partner [a=King Britt], Josh recorded the cult classic 'Liquid Summer'. He
went on to release singles for a wide variety of revered European labels ranging from
Belgium's [l=R &amp; S Records] to England's [l=XL Recordings]. In 1995, Wink became one
of the first DJ-producers to translate his hard work into mainstream success when he
unleashed a string of classics including 'Don't Laugh', 'I'm Ready' and 'Higher State of
Consciousness' that topped charts worldwide. More recently he has had massive club hits
such as 'How's Your Evening So Far' and 'Superfreak' but he has also gained a lot of
attention trough his remixes for [a=FC Kahuna], [a=Paul Oakenfold], [a=Ladytron],
[a=Clint Mansell], [a=Sting] and [a=Depeche Mode], among others.</profile>
    <data_quality>Needs Vote</data_quality>
    <urls>
      <url>http://www.joshwink.com/</url>
      <url>http://www.ovumrecordings.com/artists/josh-wink/</url>
      <url>http://www.facebook.com/JoshWinkOfficial</url>
      <url>http://twitter.com/joshwink1</url>
      <url>http://instagram.com/joshwink1</url>
      <url>http://myspace.com/joshwink</url>
      <url>http://myspace.com/ovumrecordings</url>
      <url>http://www.discogs.com/user/JoshWink</url>
      <url>http://www.discogs.com/user/josh_wink</url>
      <url>http://www.songkick.com/artists/250682-josh-wink</url>
      <url>http://en.wikipedia.org/wiki/Josh_Wink</url>
      <url>http://www.whosampled.com/Josh-Wink/</url>
      <url>https://soundcloud.com/joshwinkofficial</url>
      <url>http://www.youtube.com/user/JoshWinkVEVO</url>
      <url>http://www.dailymotion.com/JoshWink-vevo</url>
    </urls>
    ...
```

```
    ...
    <namevariations>
      <name>DJ Josh Wink</name>
      <name>DJ Wink</name>
      <name>Dosh Wink</name>
      <name>J Wink</name>
      <name>J. Wink</name>
      <name>J. Wink (DJ  Wink)</name>
      <name>J. Winkelman</name>
      <name>J. Winkelmann</name>
      <name>J.Wink</name>
      <name>J.Winkelman</name>
      <name>Josh</name>
      <name>Josh Wink (DJ WINK)</name>
      <name>Josh Winkelman</name>
      <name>Josh Winkelmann</name>
      <name>Josh Winks</name>
      <name>Josh Winx</name>
      <name>JW</name>
      <name>Winc</name>
      <name>Wink</name>
      <name>Winks</name>
      <name>Winx</name>
      <name>Winxs</name>
    </namevariations>
    <aliases>
      <name>Accent (3)</name>
      <name>Dinky Dog</name>
      <name>J. Dawg</name>
      <name>Size 9</name>
      <name>The Crusher</name>
    </aliases>
    <groups>
      <name>Abundance Of Cups</name>
      <name>E-Culture</name>
      <name>Jack Jones (4)</name>
      <name>Just King And Wink</name>
      <name>The Force (23)</name>
    </groups>
  </artist>
  ...
  <artist>
    ...
  </artist>
</artists>
```

8.1.2 Aufbau und Inhalt der "Releases"-Datei von discogs.com

Nachfolgend wird der Aufbau und Inhalt der XML-Datei mit den Informationen über die bei discogs.com hinterlegten Releases auszugsweise dargestellt. Es handelt sich dabei um einen Auszug aus der in dieser Masterarbeit verwendeten XML-Datei vom 1. Dezember 2015.

```
<releases>
  <release id="1" status="Accepted">
    ...
  </release>
  ...
  <release id="3" status="Accepted">
    <images>
      <image height="595" type="primary" uri="" uri150="" width="600" />
      <image height="472" type="secondary" uri="" uri150="" width="600" />
      <image height="600" type="secondary" uri="" uri150="" width="599" />
      <image height="470" type="secondary" uri="" uri150="" width="600" />
    </images>
    <artists>
      <artist>
        <id>3</id>
        <name>Josh Wink</name>
        <anv />
        <join />
        <role />
        <tracks />
      </artist>
    </artists>
    <title>Profound Sounds Vol. 1</title>
    <labels>
      <label catno="CK 63628" name="Ruffhouse Records" />
    </labels>
    <extraartists>
      <artist>
        <id>3</id>
        <name>Josh Wink</name>
        <anv />
        <join />
        <role>DJ Mix</role>
        <tracks />
      </artist>
    </extraartists>
    <formats>
      <format name="CD" qty="1" text="">
        <descriptions>
          <description>Compilation</description>
          <description>Mixed</description>
        </descriptions>
      </format>
    </formats>
    <genres>
      <genre>Electronic</genre>
    </genres>
    <styles>
      <style>Techno</style>
      <style>Tech House</style>
    </styles>
    <country>US</country>
    <released>1999-07-13</released>
    ...
```

```
    ...
    <notes>1: Track title is given as "D2" (which is the side of record on the vinyl
version of i220-010 release). This was also released on CD where this track is listed on
8th position. On both version no titles are given (only writing/producing credits). Both
versions of i220-010 can be seen on the master release page [m27265]. Additionally this
track contains female vocals that aren't present on original i220-010 release.
  4: Credited as J. Dahlbäck.
  5: Track title wrongly given as "Vol. 1".
  6: Credited as Gez Varley presents Tony Montana.
  12: Track exclusive to Profound Sounds Vol. 1.</notes>
    <master_id>66526</master_id>
    <data_quality>Correct</data_quality>
    <tracklist>
      <track>
        <position>1</position>
        <title>Untitled 8</title>
        <duration>7:00</duration>
        <artists>
          <artist>
            <id>5</id>
            <name>Heiko Laux</name>
            <anv />
            <join>&amp;</join>
            <role />
            <tracks />
          </artist>
          <artist>
            <id>4</id>
            <name>Johannes Heil</name>
            <anv />
            <join>,</join>
            <role />
            <tracks />
          </artist>
        </artists>
      </track>
      <track>
        <position>2</position>
        <title>Anjua (Sneaky 3)</title>
        <duration>5:28</duration>
        <artists>
          <artist>
            <id>15525</id>
            <name>Karl Axel Bissler</name>
            <anv>K.A.B.</anv>
            <join>,</join>
            <role />
            <tracks />
          </artist>
        </artists>
      </track>
      ...
```

```
      ...
      <track>
        <position>13</position>
        <title>Boom Box</title>
        <duration>3:41</duration>
        <artists>
          <artist>
            <id>19</id>
            <name>Sound Associates</name>
            <anv />
            <join>,</join>
            <role />
            <tracks />
          </artist>
        </artists>
      </track>
      <track>
        <position>14</position>
        <title>Track 2</title>
        <duration>3:39</duration>
        <artists>
          <artist>
            <id>20</id>
            <name>Percy X</name>
            <anv />
            <join>,</join>
            <role />
            <tracks />
          </artist>
        </artists>
      </track>
    </tracklist>
    <identifiers>
      <identifier type="Barcode" value="074646362822" />
    </identifiers>
    <companies>
      <company>
        <id>93330</id>
        <name>Columbia Records</name>
        <catno />
        <entity_type>10</entity_type>
        <entity_type_name>Manufactured By</entity_type_name>
        <resource_url>http://api.discogs.com/labels/93330</resource_url>
      </company>
      <company>
        <id>93330</id>
        <name>Columbia Records</name>
        <catno />
        <entity_type>9</entity_type>
        <entity_type_name>Distributed By</entity_type_name>
        <resource_url>http://api.discogs.com/labels/93330</resource_url>
      </company>
    </companies>
  </release>
  ...
  <release id="6584736" status="Accepted">
    ...
  </release>
</releases>
```

8.2 Quellcode

8.2.1 Quellcode der XML-Element-Klassen

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *NameVariations*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import java.util.List;

import org.xmappr.Element;

@RootElement
public class NameVariations {
    @Element
    public List<String> name;
}
```

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *Artist*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import org.xmappr.Element;
import org.xmappr.RootElement;

@RootElement
public class Artist {
    @Element
    public int id;

    @Element
    public String name;

    @Element
    public NameVariations namevariations;
}
```

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *Artists*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import java.util.List;

import org.xmappr.Element;
import org.xmappr.RootElement;

@RootElement
public class Artists {
    @Element
    public List<Artist> artist;
}
```

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *Track*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import org.xmappr.Element;
import org.xmappr.RootElement;

@RootElement
public class Track {
    @Element
    public String position;

    @Element
    public String title;

    @Element
    public String duration;

    @Element
    public Artists artists;
}
```

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *Tracklist*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import java.util.List;

import org.xmappr.Element;
import org.xmappr.RootElement;

@RootElement
public class Tracklist {
    @Element
    public List<Track> track;
}
```

Der nachfolgende Quellcode-Auszug zeigt die XML-Element-Klasse *Release*:

```
package edu.hm.ccwi.mpi.lindner.dataimport.util;

import org.xmappr.Attribute;
import org.xmappr.Element;
import org.xmappr.RootElement;

@RootElement
public class Release {
    @Attribute
    public int id;

    @Element
    public Artists artists;

    @Element
    public String title;

    @Element
    public Tracklist tracklist;
}
```

8.2.2 Quellcode für die Datenaustausch-Klasse

Der nachfolgende Quellcode-Auszug zeigt die speziell für den Benchmark entwickelte Klasse *SingleMetricsOutput*. Diese ist für den Austausch von individuellen Daten zwischen Mapper und Reducer notwendig.

```
public class SingleMetricsOutput implements Writable {
    private Text trackId;
    private Text metricOverall;
    private Text metricArtist;
    private Text metricTrackTitle;
    private Text metricReleaseTitle;

    public SingleMetricsOutput() {
        this.trackId = new Text();
        this.metricOverall = new Text();
        this.metricArtist = new Text();
        this.metricTrackTitle = new Text();
        this.metricReleaseTitle = new Text();
    }

    public SingleMetricsOutput(Text trackId, Text metricArtist,
                               Text metricTrackTitle, Text metricReleaseTitle) {
        this.setTrackId(trackId);
        this.setMetricOverall((Float.parseFloat(metricArtist.toString()) +
                               Float.parseFloat(metricTrackTitle.toString()) +
                               Float.parseFloat(metricReleaseTitle.toString())) / 3.0f);
        this.setMetricArtist(metricArtist);
        this.setMetricTrackTitle(metricTrackTitle);
        this.setMetricReleaseTitle(metricReleaseTitle);
    }

    public SingleMetricsOutput(String trackId, float metricArtist,
                               float metricTrackTitle, float metricReleaseTitle) {
        this.setTrackId(trackId);
        this.setMetricOverall((metricArtist+metricTrackTitle+metricReleaseTitle)/3.0f);
        this.setMetricArtist(metricArtist);
        this.setMetricTrackTitle(metricTrackTitle);
        this.setMetricReleaseTitle(metricReleaseTitle);
    }

    public String getTrackId() {
        return this.trackId.toString();
    }

    public float getMetricOverall() {
        return Float.parseFloat(this.metricOverall.toString());
    }

    public float getMetricArtist() {
        return Float.parseFloat(this.metricArtist.toString());
    }

    public float getMetricTrackTitle() {
        return Float.parseFloat(this.metricTrackTitle.toString());
    }

    public float getMetricReleaseTitle() {
        return Float.parseFloat(this.metricReleaseTitle.toString());
    }
    ...
```

```
    ...
    public void setTrackId(Text value) {
        this.trackId = value;
    }

    public void setTrackId(String value) {
        this.trackId = new Text(value);
    }

    public void setMetricOverall(Text value) {
        this.metricOverall = value;
    }

    public void setMetricOverall(float value) {
        this.metricOverall = new Text(Float.toString(value));
    }

    public void setMetricArtist(Text value) {
        this.metricArtist = value;
    }

    public void setMetricArtist(float value) {
        this.metricArtist = new Text(Float.toString(value));
    }

    public void setMetricTrackTitle(Text value) {
        this.metricTrackTitle = value;
    }

    public void setMetricTrackTitle(float value) {
        this.metricTrackTitle = new Text(Float.toString(value));
    }

    public void setMetricReleaseTitle(Text value) {
        this.metricReleaseTitle = value;
    }

    public void setMetricReleaseTitle(float value) {
        this.metricReleaseTitle = new Text(Float.toString(value));
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        this.trackId.readFields(in);
        this.metricOverall.readFields(in);
        this.metricArtist.readFields(in);
        this.metricTrackTitle.readFields(in);
        this.metricReleaseTitle.readFields(in);
    }

    @Override
    public void write(DataOutput out) throws IOException {
        this.trackId.write(out);
        this.metricOverall.write(out);
        this.metricArtist.write(out);
        this.metricTrackTitle.write(out);
        this.metricReleaseTitle.write(out);
    }
}
```

8.3 Weitere Ergebnisse der Qualitätsmessung

8.3.1 F-Score (sortiert)

Auf den nachfolgenden beiden Seiten werden die Ergebnisse der Qualitätsmessung der getesteten Ähnlichkeitsmetriken nach dem F-Score absteigend sortiert dargestellt. Dadurch wird der Rang der einzelnen Metriken sofort ersichtlich.

Rang	Algorithmus	Precision	Recall	F-Score
1	Jaro-Winkler	98,40%	85,90%	91,72%
2	Smith-Waterman	93,58%	86,00%	89,63%
3	Smith-Waterman-Gotoh	93,36%	85,70%	89,36%
4	Jaro	99,01%	69,80%	81,88%
5	Monge-Elkan	96,95%	54,10%	69,45%
6	Dice (n-Gramme: n=1)	63,99%	51,00%	56,76%
7	Dice (Leerzeichen)	100,00%	36,10%	53,05%
8	Jaccard (n-Gramme: n=1)	97,30%	36,00%	52,55%
9	Dice (n-Gramme: n=2)	99,72%	35,60%	52,47%
10	Dice (n-Gramme: n=3)	100,00%	31,70%	48,14%
11	LongestCommonSubsequence	99,34%	30,30%	46,44%
12	Damerau-Levenshtein	100,00%	30,20%	46,39%
13	Levenshtein	100,00%	30,20%	46,39%
14	Jaccard (n-Gramme: n=2)	100,00%	27,30%	42,89%
15	Jaccard (Leerzeichen)	100,00%	27,00%	42,52%
16	Jaccard (n-Gramme: n=3)	100,00%	26,30%	41,65%
17	LongestCommonSubstring	100,00%	26,20%	41,52%

Tabelle 16: Precision, Recall und F-Score der getesteten Ähnlichkeitsmetriken (sortiert)

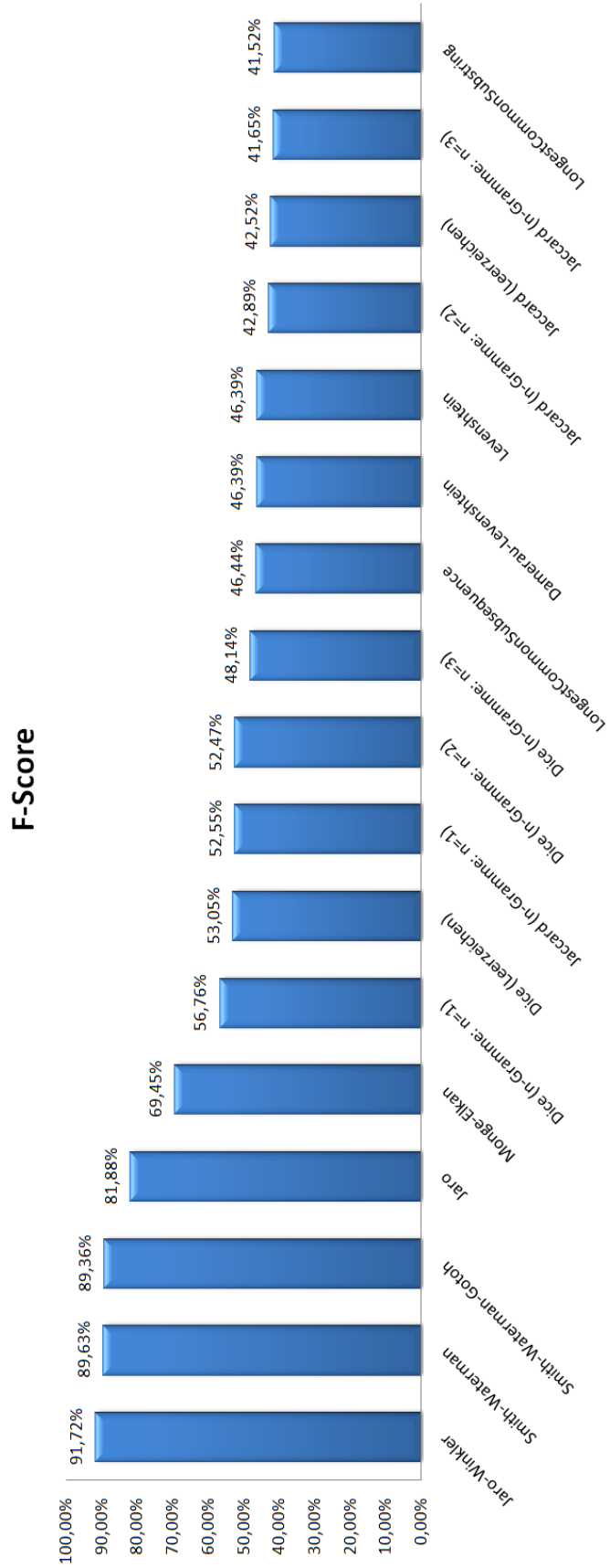

Abbildung 20: F-Score-Werte der getesteten Ähnlichkeitsmetriken (sortiert)

8.3.2 Ausführungszeit (sortiert)

Auf den nachfolgenden beiden Seiten werden die Ergebnisse der Qualitätsmessung der getesteten Ähnlichkeitsmetriken nach der Ausführungszeit aufsteigend sortiert dargestellt. Dadurch wird der Rang der einzelnen Metriken sofort ersichtlich.

Rang	Algorithmus	Ausführungszeit	In Minuten	In Sekunden
1	LongestCommonSubstring	01:16:41	77	4.600
2	LongestCommonSubsequence	01:37:49	98	5.868
3	Jaro	01:40:07	101	6.006
4	Jaro-Winkler	01:43:18	104	6.198
5	Levenshtein	02:56:49	177	10.608
6	Jaccard (Leerzeichen)	03:00:43	181	10.842
7	Dice (n-Gramme: n=3)	03:54:31	235	14.070
8	Jaccard (n-Gramme: n=3)	04:08:25	249	14.905
9	Jaccard (n-Gramme: n=2)	04:11:36	252	15.095
10	Smith-Waterman-Gotoh	04:18:15	259	15.494
11	Dice (n-Gramme: n=1)	04:27:05	268	16.025
12	Dice (n-Gramme: n=2)	04:35:42	276	16.542
13	Jaccard (n-Gramme: n=1)	04:37:04	278	16.624
14	Dice (Leerzeichen)	04:49:11	290	17.350
15	Damerau-Levenshtein	05:10:41	311	18.641
16	Monge-Elkan	11:58:10	719	43.090
17	Smith-Waterman	27:23:07	1.644	98.586

Tabelle 17: Ausführungszeiten der getesteten Ähnlichkeitsmetriken (sortiert)

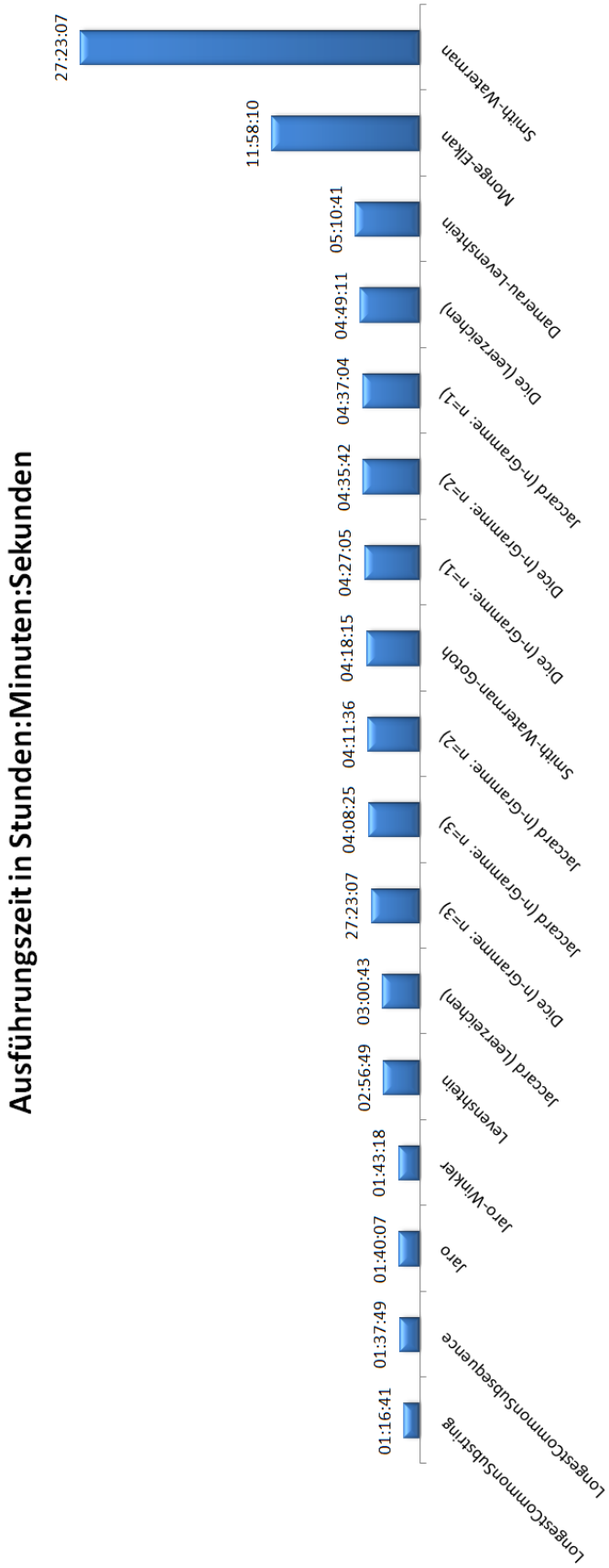

Abbildung 21: Ausführungszeiten der getesteten Ähnlichkeitsmetriken (sortiert)

8.3.3 Anzahl der Berechnungen (sortiert)

Auf den nachfolgenden beiden Seiten werden die Ergebnisse der Qualitätsmessung der getesteten Ähnlichkeitsmetriken nach der Anzahl der Berechnungen pro Sekunde absteigend sortiert dargestellt. Dadurch wird der Rang der einzelnen Metriken sofort ersichtlich.

Rang	Algorithmus	Im Cluster	Pro Knoten	Pro Map-Task
1	LongestCommonSubstring	12.223.930	3.055.982	1.018.661
2	LongestCommonSubsequence	9.582.815	2.395.704	798.568
3	Jaro	9.361.952	2.340.488	780.163
4	Jaro-Winkler	9.073.187	2.268.297	756.099
5	Levenshtein	5.301.072	1.325.268	441.756
6	Jaccard (Leerzeichen)	5.186.581	1.296.645	432.215
7	Dice (n-Gramme: n=3)	3.996.738	999.184	333.061
8	Jaccard (n-Gramme: n=3)	3.772.986	943.247	314.416
9	Jaccard (n-Gramme: n=2)	3.725.392	931.348	310.449
10	Smith-Waterman-Gotoh	3.629.391	907.348	302.449
11	Dice (n-Gramme: n=1)	3.509.233	877.308	292.436
12	Dice (n-Gramme: n=2)	3.399.565	849.891	283.297
13	Jaccard (n-Gramme: n=1)	3.382.795	845.699	281.900
14	Dice (Leerzeichen)	3.241.200	810.300	270.100
15	Damerau-Levenshtein	3.016.813	754.203	251.401
16	Monge-Elkan	1.305.087	326.272	108.757
17	Smith-Waterman	570.429	142.607	47.536

Tabelle 18: Anzahl der Berechnungen pro Sekunde der getesteten Ähnlichkeitsmetriken (sortiert)

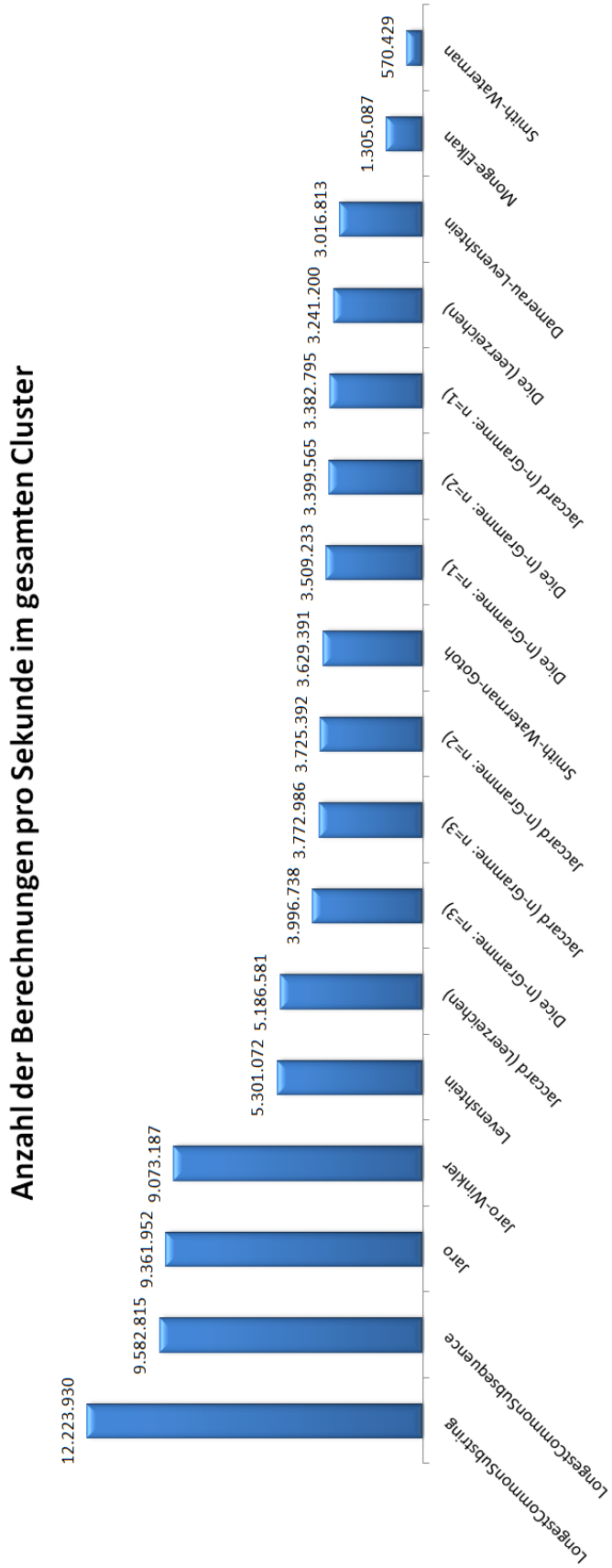

Abbildung 22: Anzahl der Berechnungen pro Sekunde der getesteten Ähnlichkeitsmetriken (sortiert)