TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Datenbanken

Belegarbeit

WEBTABELLENKLASSIFIZIERUNG MITTELS MACHINE LEARNING

Markus Hentsch

Matr.-Nr.: 3656951

Betreut durch:

Prof. Dr.-Ing. Wolfgang Lehner

Eingereicht am 27. Februar 2015

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 27. Februar 2015

ABSTRACT

Eine wertvolle Informationsquelle im World Wide Web sind Tabellen auf Webseiten. Diese Tabellen und deren Inhalt zu extrahieren birgt jedoch einige Herausforderungen. Es ist einerseits notwendig, Tabellen zu identifizieren, die sinnvolle Daten enthalten. Andererseits muss die Struktur einer solchen Tabelle korrekt erkannt werden, damit diese weiterverarbeitet werden kann, da die Anordnung von Daten innerhalb der Tabelle essentiell für deren Bedeutung ist.
Die vorliegende Ausarbeitung stützt sich auf den Ansatz des maschinellen Lernens und evaluiert verschiedene Klassifikationsalgorithmen um diese Aufgaben zu meistern und eine Klassifizierung von beliebigen Webseitentabellen zu ermöglichen.
Es wurden verschiedene Lernalgorithmen, darunter Entscheidungsbäume sowie Support Vector Machines, gegeneinander verglichen. Dabei wurden Genauigkeiten erreicht, die sowohl simples Raten als auch eine feste, regelbasierte Baseline-Klassifizierung überboten. Weiterhin konnte gezeigt werden, dass eine Aufteilung der Klassifikation in zwei Teilprobleme eine genauere Klassifikation ermöglicht. Zusätzlich wurde ein Selektionsverfahren für die Features gewählt, um den Rechenaufwand bei der Klassifizierung weiter zu reduzieren. Dabei konnte die Menge der Features zwar drastisch reduziert, der resultierende Aufwand dagegen nur geringfügig verringert werden.

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 MOTIVATION

Das World Wide Web enthält eine Fülle von Daten, viele davon eingebettet in Webseiten. Eine sehr wertvolle Quelle für Informationen sind dabei Tabellen, da sie durch ihre Struktur einen guten Ausgangspunkt für die maschinelle Verarbeitung bieten. Im Gegensatz zu reinen Texten, ist ihre strukturelle Bedeutung oft nicht an sprachliche Regeln wie etwa Grammatiken gebunden und kann automatisiert ausgewertet werden.
Die für das Erstellen von Webseiten verwendete Hypertext Markup Language (HTML) bietet durch das `<table>`-Tag eine einfache Möglichkeit, Tabellen in Webseiten einzubinden. Alles innerhalb eines solchen Tags wird von Webbrowsern als Tabelle dargestellt. Dennoch ist nicht jede in `<table>`-Tags gehüllte Struktur auch eine Tabelle mit sinnvollem Inhalt. So wird die praktische Eigenschaft des `<table>`-Tags in HTML sehr oft dazu benutzt, um beliebige Elemente einer Webseite einheitlich anzuordnen. Dabei werden meist Navigations- oder Bildelemente innerhalb solcher Tags platziert und oft auch `<table>`-Tags ineinander verschachtelt, um die gewünschte Anordnung von Elementen auf der Seite zu erzielen. Aufgrund dessen dienen viele `<table>`-Tags und deren Inhalt in HTML-Seiten lediglich dem Layout der Webseite. Die dadurch definierten Tabellen werden im Folgenden als Layout-Tabelle bezeichnet. Um also sinnvolle Informationen gewinnen zu können, ist es notwendig, diejenigen Tabellenstrukturen aus den `<table>`-Tags herauszufiltern, bei denen es sich um Layout-Tabellen handelt.
Die verbleibenden Datentabellen, im Folgenden Inhaltstabellen genannt, können darüber hinaus auch von unterschiedlicher Struktur sein. Soll eine gezielte Weiterverarbeitung gewährleistet werden, müssen sie zusätzlich unterschieden werden, da ihre Struktur essentielle Informationen über die Bedeutung der enthaltenen Daten verrät. So existieren neben klassischen, relationalen Tabellen, auch andere, die sich deutlich von deren Struktur unterscheiden. Es finden sich auf vielen Webseiten beispielsweise auch listenartige Strukturen oder Tabellen deren Attribute vertikal angeordnet sind. Für diese ist demnach eine andere Verarbeitung notwendig um die Inhalte zu extrahieren.
Es ergibt sich daher die Notwendigkeit, dass Webseitentabellen im Vorfeld korrekt klassifiziert werden müssen um ihren Inhalt maschinell weiterverarbeiten zu können.

1.2 PROBLEMANALYSE

Da die Unterscheidung für Millionen von Tabellen aus Webseitendaten automatisiert durchgeführt werden soll, ist eine Methodik nötig, die eine maschinelle Erkennung und Zuordnung der Tabellen auf effiziente Weise ermöglicht. Die als HTML-Code vorliegenden Tabellen müssen daher in eine Repräsentation umgewandelt werden, die von einem Algorithmus ausgewertet werden kann. Dafür kann eine solche Tabelle auf Werte einzelner Merkmale, engl. Features, reduziert werden, die sich maschinell berechnen lassen. Anhand deren Werte lassen sich Unterschiede zwischen Tabellen erkennen.
So können layoutbezogene Tabellen oft allein schon durch die Häufigkeitsverteilung von Bildern und Hyperlinks identifiziert werden, da sie meist einen hohen Anteil dieser Elemente aufweisen. Auch eine geringe Anzahl an Zeilen und Spalten innerhalb der Tabelle kann ein Indiz für eine Layout-Tabelle sein. Im Allgemeinen unterscheidet sich ihre Struktur stark genug von Inhaltstabellen um mit festen, von Hand erstellten Regeln eine Layout-Tabelle anhand solcher einfacher Merkmale erkennen zu können. Wie bereits in 1.1 erwähnt, muss eine Inhaltstabelle jedoch noch feiner unterschieden werden. Hier ist es oft unmöglich, einfache Features und Regeln von Hand zu erstellen, da sich die einzelnen Typen nicht derartig klar voneinander abgrenzen oder deren Zuordnung von zu vielen Faktoren abhängig ist. Aufgrund dessen ist ein Ansatz nötig, der sich weniger auf die manuelle Erstellung von Unterscheidungskriterien stützt.
Ein Ansatz der dieser Forderung gerecht wird ist das maschinelle Lernen. Dieser ist in der Lage, die Unterscheidungskriterien selbst zu erlernen und anzuwenden. Um für den eingangs beschriebenen Sachverhalt eine passende Methodik zu finden, müssen sowohl verschiedene Features als auch verschiedene Lernalgorithmen evaluiert werden um die folgenden Ziele zu erreichen:

- Definition geeigneter Features zur Repräsentation von Webseitentabellen
- Hohe Genauigkeit bei der Klassifizierung, die im Minimum sowohl primitives Raten als auch gegebenenfalls eine definierte, regelbasierte Baseline übertrifft
- Performante Verarbeitung von großen Mengen an Webseitendaten bei der Klassifizierung durch den Lernalgorithmus

2 GRUNDLAGEN DES MASCHINELLEN LERNENS

Beim maschinellen Lernen generiert ein künstliches System aus Eingabedaten Wissen, das es nach einer Lernphase verallgemeinern und auf neue Eingabedaten anwenden kann. Dabei versucht es Muster beziehungsweise Gesetzmäßigkeiten in den Lerndaten zu erkennen, um auch völlig neuartige Daten zuordnen zu können. Das maschinelle Lernen wird in 3 grundlegende Ansätze unterteilt: überwachtes Lernen, unüberwachtes Lernen und bestärkendes Lernen.

Überwachtes Lernen: Beim überwachten Lernen wird dem lernenden System die Art der möglichen Ausgaben vorgegeben. Darüber hinaus wird ein Datensatz bereitgestellt, der sowohl Eingabewerte als auch die dazugehörigen, korrekten Ausgaben enthält. Anhand dieses vorgefertigten Beispieldatensatzes lernt das künstliche System anschließend sein Wissen. Beim überwachten Lernen wird besonders auf die feste Definition der Ausgabe sowie die Nutzung vorhandener Beispiele Wert gelegt.

Unüberwachtes Lernen: Beim unüberwachten Lernen, werden lediglich Eingabedaten bereitgestellt. Das künstliche System übernimmt hier den Lernvorgang komplett selbst und legt eigenständig mögliche Ausgabestrukturen fest. Dabei versucht es Muster in den Eingabedaten zu erkennen und diese in geeignete Gruppen zu unterteilen. Hierbei liegt das Hauptaugenmerk auf der Selbständigkeit des Lernvorgangs, sodass ohne vorherigen Aufwand und Vorwissen eine Einteilung von Eingabedaten erzeugt werden kann.

Bestärkendes Lernen: Der Ansatz des bestärkenden Lernens basiert auf der Evaluation der vom künstlichen System getroffenen Entscheidungen oder dessen ausgeführten Aktionen. Hierbei wird das System je nach Ergebnis belohnt oder bestraft. Das System gewinnt an Erfahrung über vergangene Entscheidungen und deren Lob oder Tadel. Es strebt dabei nach möglichst großer Belohnung und versucht neue Entscheidungssituationen entsprechend gewinnmaximierend zu lösen. Der Lösungsweg ist dabei dem System selbst überlassen.

Um den konkreten Anwendungsfall der Webtabellen-Klassifizierung realisieren zu können, wurde

das Prinzip des überwachten Lernens gewählt, da die klare Definition der möglichen Ausgabewerte hierbei besonders wichtig ist. Darüber hinaus ist es leicht möglich, für Tabellenklassifikationen vorgefertigte Beispieldaten zu erstellen.
Das Prinzip des überwachten Lernens kann dabei in 2 Phasen unterteilt werden: die Lernphase und die Klassifizierungsphase, die im Folgenden beschrieben sind.

2.1 LERNPHASE

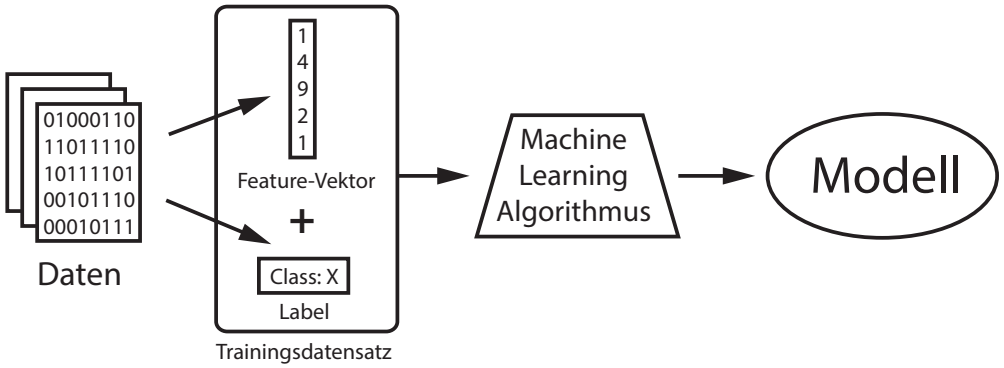

Abbildung 2.1: Lernvorgang beim überwachten Lernen

Grundlage für den Lernvorgang des überwachten Lernens bilden Daten, deren korrekte Klassifizierung bereits bekannt ist. Da die Daten beliebigen Formates sein können, müssen sie zur weiteren Verarbeitung erst in ein Format übertragen werden, dass für das künstliche System verständlich ist. Ganz so, wie ein Mensch sich unter dem HTML-Code einer Webtabelle wenig vorstellen kann, die gerenderte Tabelle jedoch intuitiv liest. Dafür müssen die Rohdaten auf Merkmale, engl. Features, reduziert werden, die sich maschinell verarbeiten lassen. Es bieten sich hier mathematische Werte an, da sie besonders gut maschinell berechnet und ausgewertet werden können.
Die für jedes Eingabedatum erzeugten Feature-Werte bilden zusammen den Feature-Vektor des jeweiligen Eingabedatums. Zusätzlich erhält jedes Datum noch ein Label, welches die korrekte Klassifizierung des Datums angibt. Das künstliche System wird hier durch einen Machine Learning Algorithmus, im Folgenden vereinfacht Lernalgorithmus genannt, verkörpert. Die Feature-Vektoren bilden zusammen mit den Labels einen Trainingsdatensatz, anhand dessen der Lernalgorithmus lernen kann. Als Ergebnis der Lernphase erzeugt er ein Modell, das sein erlerntes Wissen enthält.

2.2 KLASSIFIZIERUNGSPHASE

Beim Klassifizierungsvorgang benutzt der Algorithmus das zuvor in der Lernphase erzeugte Modell, um für beliebige Eingabedaten Aussagen über deren Zuordnung im Sinne der Klassifizierung zu treffen. Auch hierfür müssen die Eingabedaten wieder in die gleiche Repräsentation wie in 2.1 überführt werden, da das erstellte Modell nur auf diese Zusammensetzung der Feature-Vektoren anwendbar ist. Anschließend kann der Algorithmus anhand der Feature-Werte und seines Modells Vorhersagen über das Label eines Eingabedatums treffen.

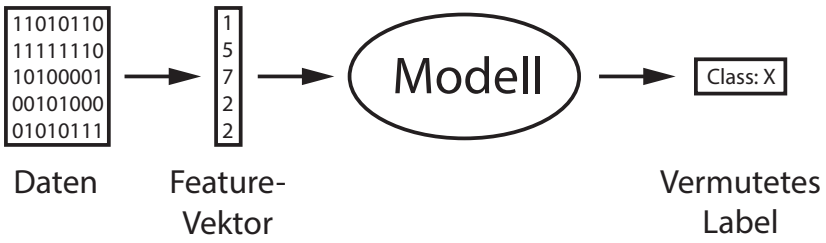

Abbildung 2.2: Klassifizierungsvorgang beim überwachten Lernen

2.3 FAKTOREN DES LERNERFOLGS

Der Lernerfolg und die daraus resultierende Zuverlässigkeit des Algorithmus in der Klassifizierungsphase sind besonders von den gewählten Features abhängig. Es ist zwar zunächst von Vorteil, eine breite Auswahl an Features zur Verfügung zu haben, aber die mögliche Beeinflussung der Features untereinander macht eine Selektion selbiger nötig. So kann es geschehen, dass sich Features während der Lernphase überlagern und ein statistisches Rauschen erzeugen, das andere, möglicherweise wichtigere Features übertönt. Eine manuelle Auswahl der Features führt hierbei nur selten zum Erfolg, da von vornherein oft nicht alle Abhängigkeiten und Redundanzen ersichtlich sind und meist der subjektiven Beurteilung durch Menschen unterliegen.
Aus diesem Grund bedient man sich dem Ansatz der Featureauswahl, engl. Feature Subset Selection. Hierbei werden Features maschinell ausgewählt und eine reduzierte Feature-Menge erstellt. Man unterscheidet dabei 3 Arten der Feature Subset Selection: Filter, Wrapper und Embedded.

Der **Filter-Ansatz** benutzt die Werte innerhalb der in der Lernphase erstellten Feature-Vektoren und berechnet für jedes Feature ein Gewicht. Anschließend sortiert er diese nach ihrem Gewicht und wählt eine vorher festgelegte Anzahl an besten Features aus. Dabei ist er völlig entkoppelt vom eigentlichen Lernalgorithmus und wird noch vor diesem ausgeführt.

Der **Wrapper-Ansatz** gliedert sich direkt an den Lernalgorithmus selbst. Er durchsucht die Menge aller Teilmengen der gesamten Feature-Menge. Dabei benutzt er den Lernalgorithmus zur Bewertung der einzelnen Teilmengen. Er unterteilt den Trainingsdatensatz und lernt mit den Feature-Teilmengen auf einem Teil und prüft deren Genauigkeit auf dem anderen. Anhand der Bewertung wählt er schließlich die beste Teilmenge an Features aus.

Beim **Embedded-Ansatz** ist die Featureauswahl direkt mit dem Lernalgorithmus verbunden und erfordert eine direkte Implementierung in den Lernalgorithmus. Die Funktionsweise ist hierbei vom Algorithmus abhängig.

Um den Ansatz des maschinellen Lernens zu Verwenden, ist es also vonnöten, sich für passende Verfahren und Algorithmen zu entscheiden. Da die Klassifizierung von Webseiteninhalten und -tabellen mittels maschinellem Lernen bereits durch mehrere Werke betrachtet wurde, lohnt sich zunächst ein Blick auf verwandte Arbeiten, um auf bisherigen Herangehensweisen aufbauen zu können.

3 VERWANDTE ARBEITEN

Der folgende Abschnitt gibt einen Überblick über verwandte Arbeiten und geht auf vorhandene Ansätze zur Webtabellenklassifizierung und Feature Subset Selection ein.

Crestan und Pantel beschreiben in ihrer Arbeit "Web-scale Table Census and Classification" [3] aus dem Jahre 2011 bereits gute Ansätze, das Tabellenklassifizierungsproblem zu lösen. Neben Layout-Tabellen, die sie in "Navigational" und "Formatting" unterteilen, stellen sie 7 Strukturtypen für Tabellen vor: Vertikale Listen, horizontale Listen, Attribut/Wert Tabellen (relational), Matrix-Tabellen, Kalender, Aufzählungen und Formulare.
Sie benutzen einen groben, regelbasierten Vorfilter um mutmaßliche Layout-Tabellen herauszufiltern, die nicht den folgenden Kriterien genügen: mindestens 2 Spalten und 2 Zeilen, keine Zelle enthält mehr als 100 Zeichen. Danach setzen sie auf einen "Gradient Boosted Decision Tree" als Machine Learning Modell für die Klassifizierung. Dafür stellen sie insgesamt 20 Features vor, die sie in 2 Gruppen unterteilen: Globale Features und Lokale Features.
Globale Features umfassen dabei allgemeine Angaben zur Tabellenstruktur, so etwa die maximale Anzahl an Zeilen bzw. Spalten oder maximale Zeichenkettenlänge innerhalb einer Zelle. Globale Features werden dabei einmal für die gesamte Tabelle berechnet, wobei lokale Features zeilen- und spaltenweise ermittelt werden. Sie werden jeweils für die ersten beiden Zeilen und Spalten, sowie jeweils für die letzte Zeile und Spalte berechnet.
Crestan und Pantel teilen die Strukturtypen jedoch in sehr viele Untertypen auf. So ist es für unseren Fall beispielsweise nicht hilfreich, ein Formular als informationstragende Tabelle zu behandeln oder Kalender-Tabellen als eigenständige Klasse zu kategorisieren. Auch konzentriert sich der Feature-Satz sehr stark auf die lokalen Features, welche sehr intensiv in der Rechenzeit sein können und lässt viele Möglichkeiten für globale Features außen vor.

Lautert, Scheidt und Dorneles bauen in ihrer Ausarbeitung "Web Table Taxonomy and Formalization" [7] 2 Jahre nach Crestan et al. auf deren Arbeit auf. Dabei beziehen sie alle 20 Features mit ein, die von Crestan et al. vorgestellt wurden und ergänzen den Satz der Features um 5 neue: Position innerer HTML-Tabellen sowie den Anteil an Zellen mit unsortierten Listen, sortierten Listen, Kommas bzw. Klammern.
Die Klassifizierung nehmen sie mithilfe eines Classifiers vor, der auf einem Neuralen Netz basiert. Dabei unterteilen sie die Klassifizierung in 2 Schritte. In der primären Hauptklassifizierung

werden die Tabellen den 5 disjunkten Kategorien Horizontal, Vertikal, Matrix, Formatierung und Navigation zugeordnet. Danach erfolgt eine sekundäre Klassifikation, in der die horizontalen, vertikalen und Matrix-Tabellen erneut 5 Kategorien anhand ihrer Struktur oder etwaigen Verschachtelung zugeordnet werden. Diese sekundäre Zuordnung ist jedoch im Vergleich zur primären nicht disjunkt, demnach können Tabellen auch mehreren Klassen innerhalb der sekundären angehören.
Die sekundäre Klassifikation ist für unseren Fall wenig sinnvoll, da wir keine verschachtelten Tabellen betrachten.

Wang und Hu behandeln 2002 in ihrer Arbeit "A Machine Learning Based Approach for Table Detection on The Web" [12] lediglich die Trennung zwischen echten relationalen Tabellen ("Genuine Tables") und Layout-Tabellen ("Non-genuine Tables") mittels maschinellem Lernens, stellen jedoch einige interessante Features vor. So benutzen sie nicht nur die Häufigkeiten von Inhaltstypen in Zellen (beispielsweise Anteil an Bildern, Form-Tags, Hyperlinks etc.), sondern präsentieren dabei auch ein komplexeres Feature, dass sie "Average Content Type Consistency" nennen, um die inhaltliche Konsistenz über Spalten und Zeilen zu repräsentieren. Auch die layoutbezogenen Features erweitern sie hier nicht nur um gängige Mittelwerte (beispielsweise mittlere Anzahl an Spalten bzw. Zeilen), sondern fügen sowohl Standardabweichungen (Spalten-/Zeilenzahl, Länge der Zelleninhalte) als auch ein Feature namens "Average Cumulative Length Consistency" hinzu. Letzteres ist ähnlich der Content Type Consistency ein komplexeres Feature, dass ein Maß für die Längenkonsistenz von Zelleninhalten repräsentiert.
Obwohl Wang und Hu auf die weitere Klassifikation von relationalen Tabellen nicht eingehen, sind die vorgestellten Features durchaus einen Blick wert, was deren Nützlichkeit für die weitere Klassifikation betrifft.

Cafarella et al. konzentrieren sich in ihrer Arbeit "Uncovering the Relational Web" [2] von 2008 auf die Extraktion von relationalen Tabellen und ihren Metadaten aus dem Web. Dabei beschränken sie sich ausschließlich auf Tabellen mit relationaler Struktur. In ihrer Ausarbeitung beschreiben sie die erste Phase des von ihnen "WebTables" getauften Projektes, welches zur Erstellung einer Sammlung von korrekt erkannten relationalen Tabellen und zusätzlichen Metadaten dienen soll.
Sie stellen dafür ein System zum Herausfiltern von relationalen Tabellen aus rohen HTML-Daten vor. Dieses basiert auf einem von ihnen handgeschriebenen Feature-Satz, welcher zusammen mit dem Ansatz des maschinellen Lernens und durch Probanden erzeugten Trainingsdaten als Filter für relationale Tabellen fungiert. Zusätzlich benutzen Cafarella et al. einen zweiten Classifier zur Erkennung von Tabellenköpfen innerhalb all jener Tabellen, die als relational erkannt wurden. Auch hierfür stellen sie einen separaten, handgeschriebenen Feature-Satz und Trainingsdatensatz bereit.
Anhand der erkannten Tabellenköpfe versuchen Cafarella et al. Metadaten in Form von Attributnamen der Tabellenspalten zu extrahieren. Sie geben dabei zu verstehen, dass qualitativ hochwertige Attributlabel die extrahierte Sammlung um wertvolle Daten bereichern. So erlauben sie die Erstellung einer Statistiksammlung, die sie "Attribute Coocurrence Statistics Database", kurz ACSDb, nennen. Darin speichern sie, wie häufig ein Attribut im Kontext zu anderen Attributen vorkommt. Dies erlaubt die Bestimmung der Wahrscheinlichkeit, dass eine bestimmte Auswahl an Attributen als Schema einer Tabelle vorkommen. Des Weiteren lassen sich auch Schema-Zusammenhänge erkennen um etwa Autovervollständigung beim Design von Schemata bereitzustellen oder schemaübergreifend synonyme Attribute zu erkennen.

Cafarella et al. konzentrieren sich in ihrer Arbeit auf spezifische Anwendungsgebiete von Tabellen und bestimmten Metadaten, wobei sie ausschließlich relational strukturierte Tabellen betrachten. Die vorgestellten Feature-Sätze sind für unseren Fall nicht brauchbar, da sie sich auf die binäre Klassifikation und die Erstellung von spezifischen Statistiken spezialisieren.

Mark A. Hall widmet sich in seiner Arbeit "Correlation-based Feature Selection for Machine Learning" [5] von 1991 einem zentralen Problem bei der Anwendung des maschinellen Lernens: der Auswahl einer repräsentativen Menge an Features. Er stellt zunächst die These auf, dass eine gute Menge nur diejenigen Features enthält, die stark mit der Klasse korrelieren aber untereinander nicht korrelieren. Er stellt einen neuen Algorithmus vor, den er "Correlation based Feature Selection" kurz CFS nennt, um diesem Problem zu begegnen und seine These zu untermauern. Bei diesem Algorithmus handelt es sich um einen vollautomatischen Filter, der die Reduzierung der Feature-Menge noch vor Anwendung des Lernalgorithmus vornimmt. Er kombiniert dafür eine Feature-Evaluierungsformel aus der klassischen Testtheorie mit einer heuristischen Suchstrategie und bildet daraus einen Algorithmus.
Hall evaluiert seinen CFS Algorithmus auf künstlichen sowie natürlichen Datensätzen mithilfe drei verschiedener Lernalgorithmen. Er zeigt, dass CFS irrelevante und redundante Features schnell identifiziert und filtert und auf natürlichen Datensätzen im Großteil der Fälle über die Hälfte der Features verwirft. Bedeutende Features werden dabei korrekt erkannt solange sie keine zu große Abhängigkeit zu anderen Features aufweisen, so Hall. Er zeigt, dass im Allgemeinen die Genauigkeit von Lernalgorithmen, die den reduzierten Satz an Features benutzen auf gleichem oder besserem Niveau liegen als mit der Menge aller Features.
Außerdem legt Hall dar, dass CFS vergleichbare Resultate wie ein Wrapper-Ansatz bei einem naiven Bayes-Klassifikator erzielt und diesen bei kleinen Datensätzen in der Genauigkeit sogar übertrifft. Dabei sei jedoch die Laufzeit des CFS um ein Vielfaches geringer und skaliere somit besser auf großen Datensätzen.
Mark A. Hall kommt zu dem Schluss, dass sein CFS Algorithmus in einem Großteil der Fälle in der Lage ist, eine gleichbleibende oder gar bessere Genauigkeiten beim maschinellen Lernen mit einem reduzierten Satz an Features zu erzielen und aufgrund seiner Schnelligkeit, Skalierbarkeit und Selbständigkeit einen guten allgemeinen Ansatz beim maschinellen Lernen bietet.

4 TABELLENTYPEN, FEATURES UND ALGORITHMEN

Im folgenden Kapitel werden zunächst die zu unterscheidenden Strukturtypen für Inhaltstabellen definiert, welche die Grundlage für die nachfolgenden Abschnitte bilden. Danach werden die verwendeten Features und deren Berechnung ausführlich beschrieben. Abschließend werden die betrachteten Lernalgorithmen vorgestellt.

4.1 TABELLENTYPEN

Um überhaupt eine Klassifizierung durchführen zu können müssen zunächst die Tabellentypen definiert werden, zwischen denen unterschieden werden soll. Diese Arbeit orientiert sich dazu an der Einteilung von Crestan et al. [3], restrukturiert und reduziert diese dabei jedoch auf die folgenden Typen: Layout, Relational, Entity, Matrix und Sonstige. Die Charakteristiken der einzelnen Typen und ihre Unterschiede zu [3] sind in den folgenden Abschnitten beschrieben.

4.1.1 Layout-Tabellen

Layout-Tabellen umfassen all jene Tabellenstrukturen die lediglich für das Layout von Webseiten genutzt wurden und keine sinnvollen, strukturierten Daten im Sinne einer Datentabelle enthalten. Sie dienen meist der Anordnung von Elementen wie beispielsweise Bildern und Links oder stellen Navigationselemente wie Menüs bereit. Ein Beispiel dafür ist in Abbildung 4.1 zu sehen. Es handelt sich hierbei um ein Navigationsmenü auf der Seite des SELFHTML e.V.[1], welches auf den ersten Blick keine Tabelle vermuten lässt. Inspiziert man jedoch den zugehörigen Quelltext (Abbildung 4.2), so wird deutlich, dass hier ein `<table>`-Tag benutzt wurde um diese Struktur zu erstellen. Layout-Tabellen sind anhand ihrer Code-Struktur vergleichsweise leicht identifizierbar,

[1] SELFHTML e.V. Selfhtml: Navigationshilfen / kurzreferenz: Html, 2014. [Online; Stand 27. Oktober 2014].

Abbildung 4.1: Beispiel einer Layout-Tabelle als Navigationselement auf selfhtml.org

```
<table ... >
  <tbody>
    <tr>
      <td colspan="2" class="nav">
        ...
      </td>
    </tr>
    <tr>
      <td class="doc" width="110">
        ...
      </td>
      <td class="docbot" width="100%">
        <h1 class="ph1">
          ...
        </h1>
      </td>
    </tr>
      ...
    </tr>
  </tbody>
</table>
```

Abbildung 4.2: Gekürzter HTML-Quelltextauszug zu Abbildung 4.1

da sie meist viele Bilder oder Hyperlinks enthalten. Im Gegensatz zu Crestan et al. [3] verzichten wir hier auf die Unterteilung von Layout-Tabellen in Unterklassen. Außerdem gliedern wir hier die durch Crestan et al. [3] als “Form” beschriebene Klasse mit ein, da wir Formulare nicht als inhaltlich wertvoll betrachten.

Land	Hauptstadt	Fläche in km²
Baden-Württemberg	Stuttgart	35.752
Bayern	München	70.552
Berlin	—	892
Brandenburg	Potsdam	29.479
Bremen	Bremen[33]	419
Hamburg	—	755
Hessen	Wiesbaden[36]	21.115
Mecklenburg-Vorpommern	Schwerin	23.180

Abbildung 4.3: Auszug einer relationalen Tabelle aus dem Wikipedia-Artikel zu Deutschland

Bundesrepublik Deutschland	
 Flagge	 Wappen
Amtssprache	Deutsch;[1] zudem die anerkannten Minderheitensprachen, teilweise nur regional
Hauptstadt	Berlin
Staatsform	parlamentarische Bundesrepublik

Abbildung 4.4: Auszug einer Entity-Tabelle aus dem Wikipedia-Artikel zu Deutschland (gekürzt)

4.1.2 Relationale Tabellen

Relationale Tabellen besitzen die Aufteilung einer Tabelle im klassischen Sinne. Einträge sind zeilenweise angeordnet und jede Spalte enthält eine Attributausprägung für den entsprechenden Eintrag. Optional können diese Tabellen auch mit einem Tabellenkopf versehen sein, der die Attributnamen enthält. Sie entsprechen den "Vertical Listings" von Crestan et al. [3]. In Abbildung 4.3 ist eine solche relationale Tabelle abgebildet. Sie entstammt dem deutschen Wikipedia-Artikel zu Deutschland[2] und listet die Bundesländer und dazugehörige Daten auf. Zudem ist sie mit einem Tabellenkopf versehen, der die Attributnamen enthält.

4.1.3 Entity-Tabellen

Bei Entity-Tabellen handelt es sich um listenartige Strukturen, die zumeist ein einziges Objekt (Entity) beschreiben. Sie können sowohl horizontal als auch vertikal orientiert sein. Horizontale Entity-Tabellen entsprechen dabei von der Struktur her einer relationalen Tabelle, die nur einen Eintrag besitzt. Bei vertikalen Entity-Tabellen stehen die Attributnamen in der ersten Spalte und die Attributausprägungen in vertikaler Richtung in der zweiten. Diese Klasse entspricht der Vereinigung von "Attribute/Value" und "Enumeration" Typen von Crestan et al. [3]. In Abbildung 4.4 findet sich das Beispiel einer Entity-Tabelle, entnommen aus dem deutschen Wikipedia-Artikel zu Deutschland[2]. Es handelt sich hierbei um eine vertikal orientierte Entity-Tabelle.

4.1.4 Matrix-Tabellen

Matrixtabellen enthalten eine Matrix aus Attributen und Ausprägungsdimensionen. In einer Richtung (meist vertikal) sind die Attribute abgetragen, in der anderen (meist horizontal) befinden sich verschiedene Dimensionen von Attributausprägungen. Matrixtabellen bilden demnach eine Erweiterung der Entity-Tabellen um mehrere Dimensionen. In Abbildung 4.5 ist eine solche

[2] Wikipedia. Deutschland — wikipedia, die freie enzyklopädie, 2014. [Online; Stand 27. Oktober 2014].

Verkehrsmittel	2013	2008	2003
Motorisierter Individualverkehr (Auto, Motorrad, Moped)	38 %	41 %	43 %
Radverkehr	17 %	16 %	12 %
Fußverkehr	24 %	22 %	24 %
Öffentlicher Personennahverkehr (ÖPNV)	21 %	21 %	20 %

Abbildung 4.5: Beispiel einer Matrix-Tabelle aus dem Wikipedia-Artikel zu Dresden

Matrix-Tabelle dargestellt. Sie wurde dem deutschen Wikipedia-Artikel zu Dresden[3] entnommen und stellt die Verkehrsaufteilung über verschiedene Jahre dar. Auf der linken Seite sind die Attribute (Verkehrsarten) in vertikaler Richtung gelistet. In horizontaler Richtung befinden sich die Dimensionen (Jahreszahlen) im Tabellenkopf. Die Semantik eines Zellenwertes ergibt sich immer aus Kombination des jeweiligen Attributs und der Dimension. Die hier beschriebene Matrix-Klasse unterscheidet sich zu der von Crestan et al. [3] dadurch, dass Kalender nicht als Teil dieser Klasse aufgefasst werden. Jedoch werden die darin beschriebenen “Horizontal Listings” mit in diese Klasse einbezogen, da deren Subjekte hier ebenfalls als Dimensionen aufgefasst werden.

4.1.5 Sonstige Tabellen

Jedwede andere Tabelle, die nicht durch 4.1.1, 4.1.2, 4.1.3 oder 4.1.4 definiert ist, findet ihre Zuordnung als “Sonstige” Tabelle. So beispielsweise kalenderartige Strukturen.

[3] Wikipedia. Dresden — wikipedia, die freie enzyklopädie, 2014. [Online; Stand 27. Oktober 2014].

Damit ein Algorithmus Tabellen analysieren kann, müssen diese in eine ihm verständliche Repräsentation überführt werden. Beim maschinellen Lernen werden hierfür Features definiert, die sich maschinell berechnen lassen und dem Lernalgorithmus als Eingabedaten sowohl für den Lernvorgang als auch für die Klassifizierung dienen. Diese Features müssen ein möglichst breites Spektrum von dem abdecken, was eine Tabelle in ihrer Struktur ausmacht, um den Informationsverlust bei der Überführung in diese Repräsentation möglichst gering zu halten. Es ist daher angebracht, eine möglichst große Anzahl an Features zu definieren und diese anschließend einer Selektion zu unterziehen.
Im nachfolgenden Abschnitt werden alle in dieser Arbeit verwendeten Features benannt und deren Berechnung beschrieben.

4.2 GLOBALE FEATURES

Globale Features repräsentieren Merkmale der Tabelle als Gesamtes und werden jeweils genau ein Mal pro Tabelle berechnet. Features aus 4.2.1 entstammen Crestan und Pantel's Arbeit [3]. Feature 4.2.6 wurde neu hinzugefügt. Alle restlichen Features orientieren sich an der Ausarbeitung von Wang et al. [12].

4.2.1 Maxima und Mittelwerte

Maxima und Mittelwerte über die Anzahl von Spalten und Zeilen sowie der Zelleninhaltslänge ermöglichen Aussagen über die Abmessungen, Größenverhältnisse und Ausrichtung einer Tabelle.

• **Maximale Anzahl an Zeilen** in einer Tabelle. Sei C die Menge aller Spalten der Tabelle und $rows()$ die Funktion welche die Anzahl der Zellen innerhalb einer Spalte angibt, welche nicht durch ein `<span>`-Tag erzeugt wurden, dann gilt:

$$\texttt{MAX\_ROWS} = \max_{\forall c_i \in C} rows(c_i)$$

• **Maximale Anzahl an Spalten** in einer Tabelle. Sei R die Menge aller Spalten der Tabelle und $cols()$ die Funktion welche die Anzahl der Zellen innerhalb einer Zeile angibt, welche nicht durch ein `<span>`-Tag erzeugt wurden, dann gilt:

$$\texttt{MAX\_COLS} = \max_{\forall r_i \in R} cols(r_i)$$

• **Maximale Anzahl an Zeichen einer Zelle** innerhalb der Tabelle. Sei Z die Menge aller Zellen der Tabelle und $len()$ die Funktion welche die Anzahl an Zeichen innerhalb einer Zelle angibt, dann gilt:

$$\texttt{MAX\_CELL\_LENGTH} = \max_{\forall z_i \in Z} len(z_i)$$

• **Mittlere Anzahl an Zeilen** einer Tabelle. Sei n die Anzahl aller Spalten der Tabelle und r_i die

Anzahl der Zellen innerhalb der Spalte $i, i = 1, ..., n$, dann gilt:

$$\texttt{AVG\_ROWS} = \frac{1}{n} \sum_{i=1}^{n} r_i$$

• **Mittlere Anzahl an Spalten** einer Tabelle. Sei n die Anzahl aller Zeilen der Tabelle und c_i die Anzahl der Zellen innerhalb der Zeile $i, i = 1, ..., n$, dann gilt:

$$\texttt{AVG\_COLS} = \frac{1}{n} \sum_{i=1}^{n} c_i$$

• **Mittlere Länge von Zelleninhalten** einer Tabelle. Sei n die Anzahl aller Zellen der Tabelle und z_i die Anzahl an Zeichen innerhalb der Zelle $i, i = 1, ..., n$, dann gilt:

$$\texttt{AVG\_CELL\_LENGTH} = \frac{1}{n} \sum_{i=1}^{n} z_i$$

4.2.2 Standardabweichungen

Standardabweichungen über Struktureigenschaften einer Tabelle sind ein Maß für die Streuung und erlauben die Beurteilung der Gleichförmigkeit beziehungsweise Ungleichförmigkeit ihrer Struktur.

• **Standardabweichung über die Anzahl an Zeilen** innerhalb einer Tabelle. Sei r die mittlere Zeilenanzahl wie beschrieben in AvgRows, n die Anzahl an Spalten und r_i die Anzahl der Zellen innerhalb der Spalte $i, i = 1, ..., n$, dann gilt:

$$\texttt{STD\_DEV\_ROWS} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (r_i - r)^2}$$

• **Standardabweichung über die Anzahl an Spalten** innerhalb einer Tabelle. Sei c die mittlere Spaltenanzahl wie beschrieben in AvgCols, n die Anzahl an Zeilen und c_i die Anzahl der Zellen innerhalb der Zeile $i, i = 1, ..., n$, dann gilt:

$$\texttt{STD\_DEV\_COLS} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (c_i - c)^2}$$

• **Standardabweichung über die Anzahl an Zeichen in Zellen** innerhalb einer Tabelle. Sei z die mittlere Spaltenanzahl wie beschrieben in AvgCellLength, n die Anzahl an Zellen und z_i die Anzahl der Zeichen innerhalb der Zelle $i, i = 1, ..., n$, dann gilt:

$$\texttt{STD\_DEV\_CELL\_LENGTH} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (z_i - z)^2}$$

4.2.3 Durchschnittliche Längenkonsistenz

Die durchschnittliche Längenkonsistenz erlaubt Aussagen über die Konsistenz der Tabelle in Hinblick auf die Längen der Zeichenketten innerhalb der Zellen.

Sei R_i die Menge aller Zellen der Zeile $i, i = 1, ..., r$ und m_i die mittlere Zelleninhaltslänge für Zeile R_i, so berechnet sich die kumulative Längenkonsistenz pro Zeile CLC_i wie folgt:

$$CLC_i = \sum_{cl \in R_i} LC_{cl}, \text{ wobei } LC_{cl} = 0.5 - D \text{ mit } D = min(|length(cl) - m_i|/m_i, 1.0)$$

Das Mittel der kumulativen Längenkonsistenz über alle Zeilen ergibt sich aus:

$$CLC_r = \frac{1}{r} \sum_{i=1}^{r} CLC_i$$

Die Berechnung der mittleren Längenkonsistenz für die Spalten CLC_c ist analog zur Berechnung von CLC_r. Schließlich erhält man die durchschnittliche kumulative Längenkonsistenz CLC aus:

$$\texttt{CUMULATIVE\_LENGTH\_CONSISTENCY} = max(CLC_r, CLC_c)$$

4.2.4 Inhaltsanteile

Inhaltsanteile ermöglichen Aussagen über den dominanten Inhalt einer Tabelle. Überwiegt ein bestimmter Inhaltstyp, können leichter Vorhersagen über die Zuordnung einer Tabelle getätigt werden.

Sei n die Anzahl aller Zellen einer Tabelle und t_i der Inhaltstyp der Zelle $i, i = 1, ..., n$. Der Anteil eines Inhaltstyps t berechnet sich wie folgt:

$$ratio(t) = \frac{1}{n} \sum_{i=1}^{n} X_i, \text{ wobei } X_i = \begin{cases} 1, & \text{falls } t_i = t \\ 0, & \text{sonst} \end{cases}$$

Im Folgenden seien folgende Zelleninhaltstypen definiert:

Name	Beschreibung	Name des zugehörigen Features
Image	Zelle enthält `<img>`-Tag	`RATIO_IMG`
Form	Zelle enthält `<form>`-Tag	`RATIO_FORM`
Hyperlink	Zelle enthält `<a>`-Tag	`RATIO_HYPERLINK`
Alphabetical	Zelle enthält überwiegend alphabetische Zeichen	`RATIO_ALPHABETICAL`
Digit	Zelle enthält überwiegend Zahlen	`RATIO_DIGIT`
Empty	Zelle ist leer	`RATIO_EMPTY`
Other	Inhalt der Zelle trifft auf keines der obigen zu	`RATIO_OTHER`

4.2.5 Durchschnittliche Inhaltstypkonsistenz

Die durchschnittliche Inhaltstypkonsistenz erlaubt Aussagen über die Konsistenz der Tabelle in Hinblick auf die in den Zellen vorkommenden Arten von Inhalten.

Sei R_i die Menge aller Zellen der Zeile $i, i = 1, ..., r$. Zunächst finde man den dominanten Inhaltstyp DT_i für R_i. Anschließend berechne man die kumulative Typkonsistenz für jede Zeile $R_i, i = 1, ..., r$:

$$CTC_i = \sum_{cl \in R_i} D, \text{ wobei } D = \begin{cases} 1, & \text{falls } type(cl) = DT_i \\ -1, & \text{sonst} \end{cases}$$

Das Mittel über alle Zeilen ergibt sich aus:

$$CTC_r = \frac{1}{r} \sum_{i=1}^{r} CTC_i$$

Die Berechnung der Typkonsistenz für die Spalten CTC_c ist analog zur Berechnung von CTC_r. Schließlich erhält man die durchschnittliche kumulative Inhaltstypkonsistenz CTC wie folgt:

$$\texttt{CUMULATIVE\_CONTENT\_CONSISTENCY} = max(CTC_r, CTC_c)$$

4.2.6 Tabellenkopf

Die Anwesenheit eines Tabellenkopfes kann ein Indiz für spezielle Tabellenarten sein, so beispielsweise relationale Tabellen. Dieses Feature definiert sich wie folgt:

$$\texttt{HAS\_HEADER} = \begin{cases} 1, & \text{falls Tabelle <th>-Tags enthält} \\ 0, & \text{sonst} \end{cases}$$

4.3 LOKALE FEATURES

Im Gegensatz zu den globalen Features werden lokale Features nicht über die gesamte Tabelle berechnet. Sie werden jeweils pro Spalte bzw. Zeile berechnet. Für die Erstellung der Features und die weitere Verarbeitung beschränken wir uns hier ähnlich Crestan und Pantel [3] jeweils auf die ersten beiden Zeilen und Spalten sowie die letzte Zeile und Spalte einer Tabelle. Die Feature-Werte erhalten bei ihrer Berechnung jeweils einen Suffix entsprechend der Zeile bzw. Spalte für die sie berechnet wurden (beispielsweise `LOCAL_AVG_LENGTH_COL_0`). So entstehen pro lokalem Feature jeweils 6 Werte, mit den Suffixen `_COL_0`, `_COL_1`, `_COL_2` sowie `_ROW_0`, `_ROW_1` und `_ROW_2`. Dabei stehen Index `0` und `1` jeweils für die ersten beiden und Index `2` für die letzte Zeile/Spalte.

4.3.1 Lokale Strukturmerkmale

• **Lokale mittlere Zellenlänge**. Sei n die Anzahl aller Zellen innerhalb einer Spalte bzw. Zeile und lz_i die Anzahl an Zeichen innerhalb der Zelle $i, i = 1, ..., n$ aus dieser Spalte bzw. Zeile, dann berechnet sich die lokale mittlere Zellenlänge:

$$\texttt{LOCAL\_AVG\_LENGTH} = \frac{1}{n}\sum_{i=1}^{n} lz_i$$

• **Lokale Längenvarianz**. Sei n die Anzahl aller Zellen innerhalb einer Spalte bzw. Zeile und lz_i die Anzahl an Zeichen innerhalb der Zelle $i, i = 1, ..., n$ aus dieser Spalte bzw. Zeile, dann berechnet sich die lokale Längenvarianz:

$$\texttt{LOCAL\_LENGTH\_VARIANCE} = \frac{1}{n}\sum_{i=1}^{n}(lz_i - lz)^2 \quad \text{mit } lz = \frac{1}{n}\sum_{i=1}^{n} lz_i$$

• **Lokales `Span`-Verhältnis**. Sei n die Anzahl aller Zellen innerhalb einer Zeile bzw. Spalte und z_i Zelle $i, i = 1, ..., n$ innerhalb dieser Zeile/Spalte. Das lokale `Span`-Verhältnis berechnet sich wie folgt:

$$\texttt{LOCAL\_SPAN\_RATIO} = \frac{1}{n}\sum_{i=1}^{n} X_i, \text{ wobei } X_i = \begin{cases} 1, & \text{Zelle } z_i \text{ durch } \texttt{<span>}\text{-Tag erzeugt} \\ 0, & \text{sonst} \end{cases}$$

Zellen die einen `<span>`-Tag enthalten, spannen virtuelle Zellen auf. Erstere zählen hier jedoch nicht als „durch einen `<span>`-Tag erzeugt“.

4.3.2 Lokale Inhaltsanteile

Sei n die Anzahl aller Zellen innerhalb einer Zeile bzw. Spalte und T_i die Menge aller Inhaltstypen der Zelle $i, i = 1, ..., n$ innerhalb dieser Zeile/Spalte. Der Anteil eines Inhaltstyps t berechnet sich

wie folgt:

$$ratio(t) = \frac{1}{n}\sum_{i=1}^{n} X_i, \text{ wobei } X_i = \begin{cases} 1, & \text{falls } t \in T_i \\ 0, & \text{sonst} \end{cases}$$

Lokale Inhaltstypen sind im Gegensatz zu den globalen nicht exklusiv, dass heißt einer Zelle können mehrere Inhaltstypen zugeordnet sein.

Es seien folgende lokale Zelleninhaltstypen definiert:

Name	Beschreibung	Name des zugehörigen Features
Header	Zelle enthält `<th>`-Tag	`LOCAL_RATIO_HEADER`
Anchor	Zelle enthält `<a>`-Tag	`LOCAL_RATIO_ANCHOR`
Image	Zelle enthält `<img>`-Tag	`LOCAL_RATIO_IMAGE`
Input	Zelle enthält `<input>`-Tag	`LOCAL_RATIO_INPUT`
Select	Zelle enthält `<select>`-Tag	`LOCAL_RATIO_SELECT`
Font	Zelle enthält `<b>`-, `<i>`-, `<u>`-, `<font>`-Tags	`LOCAL_RATIO_FONT`
Linebreak	Zelle enthält ` `-Tag	`LOCAL_RATIO_BR`
Colon	Zelle enthält Doppelpunkt-Zeichen „:“	`LOCAL_RATIO_COLON`
Contains Number	Zelle enthält Zahl	`LOCAL_RATIO_CONTAINS_NUMBER`
Is Number	Zelle enthält ausschließlich eine Zahl	`LOCAL_RATIO_IS_NUMBER`
Non-Empty	Zelle ist nicht leer	`LOCAL_RATIO_NONEMPTY`
Unordered List	Zelle enthält -Tag	`LOCAL_RATIO_UNORDERED_LISTS`
Ordered List	Zelle enthält -Tag	`LOCAL_RATIO_ORDERED_LISTS`
Comma	Zelle enthält Komma-Zeichen „,“	`LOCAL_RATIO_COMMA`
Bracket	Zelle enthält „(“ und/oder „)“	`LOCAL_RATIO_BRACKET`

4.4 LERNALGORITHMEN

Das Kernstück beim Ansatz des maschinellen Lernens bildet der Lernalgorithmus. Er wird sowohl für den Lernvorgang als auch für den Klassifizierungsvorgang benutzt. Neben der Featureauswahl ist insbesondere die Wahl des Lernalgorithmus ausschlaggebend für den Erfolg und die Ergebnisse des Lernvorgangs.
Im Bereich des maschinellen Lernens existiert bereits eine Vielzahl von unterschiedlichen Algorithmen auf Basis unterschiedlicher Ansätze. Im folgenden Abschnitt werden all jene Algorithmen besprochen, die für die Betrachtung in dieser Arbeit gewählt wurden.
Implementierungen für die einzelnen Algorithmen wurden dem WEKA-Framework der Universität von Waikato entnommen. Dabei handelt es sich um eine freie Sammlung von Implementierungen verschiedener Algorithmen zum Data-Mining und maschinellen Lernen geschrieben in Java [9]. Zusätzlich wird ein eigener, regelbasierter Klassifizierungsalgorithmus vorgestellt, welcher als Vergleichsbasis diente.

4.4.1 Lernalgorithmen auf Basis von Entscheidungsbäumen

Bei Entscheidungsbäumen handelt es sich um geordnete, gerichtete Bäume aus der Graphentheorie. Sie werden dazu benutzt, um hierarchisch aufeinander folgende Entscheidungen abzubilden. Bei Lernalgorithmen, die auf Entscheidungsbäumen basieren, werden im Laufe der Lernphase Entscheidungsbäume erstellt, auf die dann in der Klassifizierung zurückgegriffen wird. Sie stellen somit die Wissensbasis beziehungsweise das gelernte Modell des Lernalgorithmus dar. In Abbildung 4.6 ist ein Beispiel eines solchen Entscheidungsbaumes dargestellt.
Bei der Klassifizierung traversiert ein solcher Algorithmus ausgehend vom Wurzelknoten entlang der Pfade des Entscheidungsbaums. An jedem Verzweigungsknoten wird der Wert von Features ausgewertet und eine Entscheidung über die Wahl des nächsten Knotens getroffen. Dies wird wiederholt, bis ein Blattknoten erreicht wird, welcher wiederum einer bestimmten Klassifikation entspricht. In dieser Arbeit wurden Implementierungen von drei verschiedenen, auf Entscheidungsbäumen basierenden, Lernalgorithmen benutzt. In dem folgenden Abschnitt werden die Konzepte der entsprechenden Algorithmen vorgestellt und ihre Besonderheiten aufgezeigt.

CART

CART, kurz für "Classification and Regression Trees" ist ein Algorithmus, der sich auf binäre Entscheidungsbäume beschränkt. CART unterteilt den Feature-Raum so lange binär, bis keine Features mehr verbleiben und der Baum vollständig ist. Dazu erstellt der Algorithmus zu jedem Feature einen Knoten und bestimmt für diesen einen Schwellwert, anhand dessen die binäre Trennung der beiden Pfade zu den beiden Kindknoten vorgenommen wird. Dabei erachtet der Algorithmus ein Feature, welches eine hohe Trefferquote in Bezug auf die Klassifikation aufweist, als wichtiger und platziert es entsprechend weit oben im Baum. Das Feature, welches den Wurzelknoten bildet, ist demnach das mit der höchsten Trefferquote. Grundlage für die Wichtung von Features bildet also deren Informationsgehalt. CART kürzt den entstehenden Baum anschließend mittels "Cost-Complexity Pruning" rückwärts traversierend ein. Dabei erzeugt er dafür mehrere Unterbäume, die gegen neue, noch nicht klassifizierte Daten getestet und evaluiert werden.
Es wurde hier die Implementierung "SimpleCart" des WEKA-Frameworks gewählt.

C4.5

C4.5 ist eine Erweiterung des ID3-Algorithmus, einem Algorithmus zum maschinellem Lernen anhand von Entscheidungsbäumen. Er verhält sich ähnlich zum CART-Algorithmus, weist jedoch einige Besonderheiten auf [11]. So verzichtet C4.5 auf die Notwendigkeit einer strikt binären Aufteilung, sodass eine beliebige Zahl von Verzweigungen in den Entscheidungsbaum aufgenommen werden können. Der entstehende Baum wird meist breiter aber weniger tief als ein vergleichbarer, durch CART generierter Baum. Des Weiteren wird beim Pruning im Vergleich zu CART nicht gegen neue, unklassifizierte Daten geprüft, sondern anhand der gleichen Daten evaluiert, die auch zur Erstellung des Baums genutzt wurden.
Es wurde hier die Implementierung "J48" des WEKA-Frameworks gewählt.

RandomForest

RandomForest ist ein Lernalgorithmus, bei dem während des Lernprozesses anstatt eines Entscheidungsbaumes mehrere generiert werden, die als Modell für die Klassifikationsphase dienen. Im Gegensatz zu den meisten anderen Algorithmen, teilt RandomForest den Feature-Raum an jedem Knoten nicht anhand der optimalen Teilung für die gesamte Feature-Menge, sondern anhand einer Untermenge an Features, die für jeden Knoten zufällig gewählt wird [8]. Bei der Klassifikation wird ein zu klassifizierendes Objekt von jedem Baum im Modell ausgewertet und die von den Bäumen am häufigsten bestimmte Klasse als Ergebnis gewählt. Bei RandomForest wird kein Pruning verwendet.
Mit dieser Methodik kann sich der RandomForest-Algorithmus auch gut gegen andere Algorithmen wie beispielsweise Support Vector Machines und neurale Netze behaupten [1].
Für unsere Untersuchungen wurde die gleichnamige Implementierung "RandomForest" des WEKA-Frameworks benutzt.

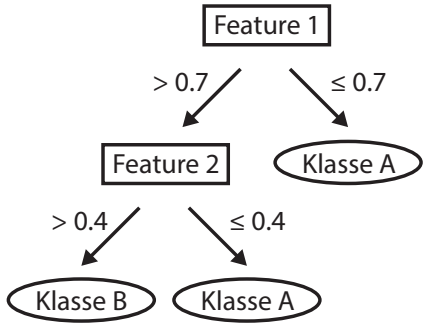

Abbildung 4.6: Schematisches Beispiel eines binären Entscheidungsbaums für eine Klassifizierung

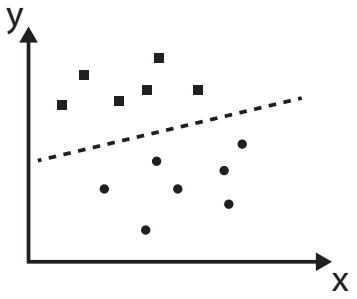

Abbildung 4.7: Eingepasste Ebene (gestrichelte Linie) zur Trennung zweier Vektormengen im zweidimensionalen Raum

4.4.2 Support Vector Machines

Bei Support Vector Machines, kurz SVM, handelt es sich um mathematische Verfahren zur Mustererkennung, die beim maschinellen Lernen eingesetzt werden. Eine solche Support Vector Machine stellt die Trainingsdaten im Vektorraum dar, in dem jedes Objekt durch einen Vektor repräsentiert wird. Die Support Vector Machine versucht beim Lernvorgang eine Hyperebene so in diesen Vektorraum einzupassen, dass diese als Trennfläche fungiert und den Datensatz in zwei Klassen unterteilt (vgl. Abbildung 4.7). Dabei wird der Abstand aller, der Hyperebene am nächsten liegenden, Vektoren maximiert. Diese Vektoren werden "Support Vectors" genannt und sind zentrale Grundlage der Teilung. Weiter entfernte Vektoren haben keinen Einfluss auf die Platzierung der Ebene und müssen nicht betrachtet werden.
Da die eingezogene Ebene den Raum nur linear trennen kann, ist es im Allgemeinen nicht immer möglich eine solche zu finden, da in realen Anwendungsfällen die Objekte oft nicht linear trennbar sind. Um dennoch eine nicht-lineare Klassengrenze finden zu können, bedient sich die Support Vector Machine des Kernel-Tricks. Grundlegende Idee hierbei ist, den Vektorraum in einen höherdimensionalen Raum zu überführen und bei entsprechend hoher Dimensionsanzahl schließlich die lineare Trennung durch eine Hyperebene in diesem Raum zu ermöglichen. Die

entstandene Hyperebene würde dann in den ursprünglichen Raum zurück transformiert. Da die direkte Umsetzung dieses Prinzips mit einem sehr hohen Aufwand verbunden ist, beschreibt man die Trennfläche einfach durch Kernelfunktionen, die die Hyperebene im Hochdimensionalen beschreiben und auf den niedrigdimensionalen Bereich abbilden ohne die Rechnung tatsächlich ausführen zu müssen.
Wir haben uns für die WEKA-Implementierung des "Sequential Minimal Optimization" Algorithmus von John Platt [10], kurz SMO entschieden. Dieser Algorithmus dient primär der Lösung des Problems der quadratischen Programmierung, wie es bei der Lernphase einer SVM entsteht. Die Implementierung in WEKA ist darüber hinaus in der Lage, also Klassifizierungsprobleme mit mehr als zwei Klassen zu verarbeiten, indem die Klassifizierung gemäß Hastie und Tibshirani [6] paarweise vorgenommen wird. Dies wäre mit einer reinen SVM aufgrund der binären Teilung nicht ohne weiteres möglich. Für unseren Anwendungsfall ist dies jedoch essentiell, da mehr als zwei Tabellenarten unterschieden werden müssen.

4.4.3 Baseline-Klassifizierung

Um eine Vergleichsbasis zu haben, wurde ein einfacher, regelbasierter Klassifizierungsalgorithmus entworfen. Dieser beschränkt sich auf den einfachen Anwendungsfall der Unterscheidung von Layout-Tabellen und Inhaltstabellen, da hierfür intuitive Regeln per Hand entworfen werden konnten. Der vereinfachte Java-Quelltext ist in Abbildung 4.8 dargestellt.
So werden Tabellen, deren Spalten- und Zeilenmaße einen festgelegten Wert unterschreiten, als Layout-Tabellen eingestuft. Ebenso Tabellen, die in ihrer Spaltenzahl Unregelmäßigkeiten aufweisen oder deren mittlere Kopfzeilenbeschriftungslänge nicht innerhalb eines festgelegten Zahlenbereichs liegt.
Der Algorithmus kann letztlich nur zwischen Layout- und Inhaltstabelle unterscheiden und kann daher nur in reduzierten Teilproblemen als Vergleichsbasis gegenüber den anderen Lernalgorithmen herangezogen werden.

```
boolean isTableRelation(table) {

        // Tabellen innerhalb von Formularen
        if (isWithinFormTag)
                return false;

        // Tabellen, die Tabellen enthalten
        if (containsTable)
                return false;

        // Tabellen mit weniger als 3 Zeilen
        int rows = countRows();
        if (rows < 3)
                return false;

        // Tabellen, deren häufigste Spaltenzahl
        // weniger als 2 beträgt
        int columns = getMostFrequentColumnCount();
        if (columns < 2)
                return false;

        // Tabellen, deren häufigste Spaltenzahl nicht
        // mit der maximalen übereinstimmt
        int columns_max = getMaximumColumnCount();
        if (columns != columns_max)
                return false;

        // Zu kurze oder zu lange Attributnamen
        header_length = countCharactersFirstRow();
        header_columns = getHeaderColumnCount();
        if ((header_length / header_columns) < 4)
                return false;

        if ((header_length / header_columns) > 20)
                return false;

        return true;
}
```

Abbildung 4.8: Java-Pseudocode für den als Referenz verwendeten Baseline-Algorithmus zur Klassifizierung von Tabellen in Relationale und Layout-Tabellen

5 EVALUATION

Um die eingangs erwähnten Ziele aus 1.2 hinsichtlich der Performanz und Genauigkeit zu erreichen, müssen Lernalgorithmen gewählt und anschließend unter diesen Gesichtspunkten evaluiert werden. Vorraussetzung dafür ist zusätzlich ein Testdatensatz als Eingabedaten für diese Algorithmen. Daher müssen folgende Punkte betrachtet werden:

- Die Wahl und Aufbereitung einer geeigneten Datenquelle zur Erstellung eines Testdatensatzes für die Lernalgorithmen
- Der Vergleich von verschiedenen Lernalgorithmen in Hinblick auf Performanz und Genauigkeit der Klassifizierungsergebnisse
- Die Verkleinerung der benutzten Feature-Menge mittels Feature Subset Selection zur Verringerung des Rechenaufwandes ohne Verschlechterung der Ergebnisse

Alle in diesem Kapitel erwähnten Messwerte für Laufzeiten wurden auf einem System mit den folgenden Spezifikationen durchgeführt: Intel Core i5-3570 3.4 GHz, 8 GB DDR3 1333 MHz RAM, Ubuntu 14.04.1 LTS mit Oracle Java 1.7.0. Für alle Algorithmen wurde die Standardkonfiguration des WEKA-Frameworks in Version 3.6.10 verwendet.

5.1 WAHL DES DATENSATZES

Als Basis zur Erstellung des Trainingsdatensatzes wurde ein Auszug des Webseitenarchivs der Common Crawl Foundation [4] gewählt. Bei der Common Crawl Foundation handelt es sich um eine im Jahre 2007 gegründete Non-Profit-Organisation, die in regelmäßigen Abständen das World Wide Web per Crawling durchsucht und Abbilder von Webseiten abspeichert. Diese gespeicherten Webseitendaten werden von der Common Crawl Foundation in Archiven gebündelt und öffentlich bereitgestellt. Eine solche Sammlung wird von der Foundation mehrmals pro Jahr erstellt und umfasst pro Sammlung etwa 2 bis 5 Milliarden gespeicherte Webseiten.
Es wurden stichprobenartig Einzelarchive aus dem aktuellsten Datenbestand der Common Crawl

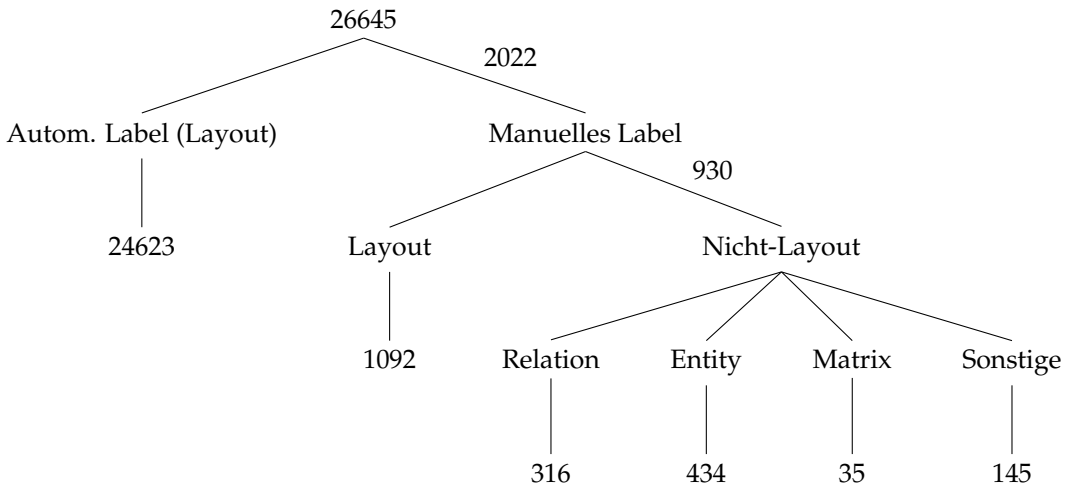

Abbildung 5.1: Übersicht der Verteilung der Labels innerhalb des entstandenen Trainingsdatensatzes

Sammlung bezogen. Dabei wurden insgesamt 8 Archive mit einer Gesamtgröße von 5,2 GB heruntergeladen. Diese enthielten eine Gesamtmenge von 334.380 gespeicherten Webseiten mit insgesamt knapp über 2 Millionen Tabellen.

Um den Trainingsdatensatz erstellen zu können müssen Eingangsdaten vorliegen, die mit einem Label versehen sind, welches die korrekte Klassifizierung der Daten angibt. Da diese Zusatzinformationen in den Archivdateien von Common Crawl nicht enthalten waren, wurden sie manuell hinzugefügt.
Dazu wurden insgesamt 26.645 Tabellen stichprobenartig aus den Archiven von Common Crawl extrahiert. Davon wurden 24.623 bereits durch eine regelbasierte Vorfilterung mit einem Layout-Label gekennzeichnet. Diese Vorfilterung sortierte Tabellen mit weniger als zwei Spalten oder zwei Zeilen, sowie ungültige und nicht darstellbare Tabellen im Vorfeld aus. Die verbleibenden 2.022 Tabellen wurden schließlich manuell klassifiziert. Davon wurden 930 als Inhaltstabellen eingestuft. Die resultierende Verteilung der Tabellen-Label ist in Übersicht 5.1 dargestellt.
Auffällig ist hierbei, dass 25.715 Tabellen insgesamt als Layout-Tabellen identifiziert wurden, was etwa 96,5 Prozent des gesamten Satzes ausmachte.
Ein derart hoher Anteil an Layout-Tabellen findet sich auch in bisherigen Arbeiten. So filterten Crestan et al. in ihrer Arbeit 93% an Layout-Tabellen aus [3]. Cafarella et al. hingegen betrachten zwar nur relationale Tabellen, erkennen aber etwa 90% als klar layoutbezogene Strukturen und weitere 9,5% als nicht-relationale Strukturen in ihren analysierten Webdaten [2].

5.2 AUSWAHL DES LERNALGORITHMUS

Die Menge der zu vergleichenden Lernalgorithmen wurde durch die folgenden 4 aus dem WEKA-Framework gebildet: RandomForest, J48, SimpleCart und SMO.
Als zusätzliche Vergleichsbasis für die binäre Unterscheidung von Layout-Tabellen und Inhaltstabellen wurde ein einfacher, regelbasierter Algorithmus hinzugezogen. Für die Eingabedaten wurden die Werte aller in Kapitel 4 gelisteten Features berechnet. Grundsätzlich wurden die Lernalgorithmen in den folgenden Disziplinen geprüft:

1. **Einstufiges Klassifizierungsproblem** - keine Aufteilung oder Neustrukturierung der Trainingsdaten
2. **Zweistufiges Klassifizierungsproblem**
   (a) **Vorsortierungsphase** - Reduzierung der Eingabedaten auf die Klassen Layout und Nicht-Layout
   (b) **Hauptphase** - Verwurf aller Layout-Tabellen und reine Unterscheidung von Inhaltstabellen

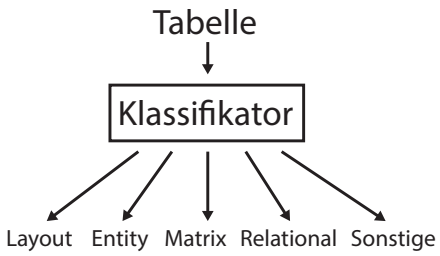

Abbildung 5.2: Schema des einstufigen Klassifizierungsproblems

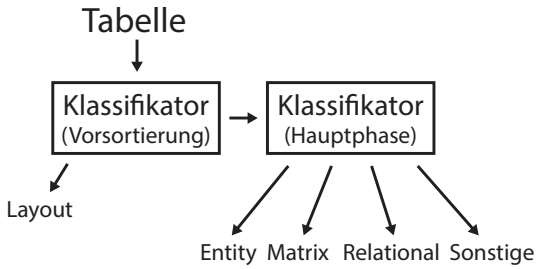

Abbildung 5.3: Schema des zweistufigen Klassifizierungsproblems

Aufgrund der bereits in 5.1 erlangten Erkenntnis, dass Layout-Tabellen einen sehr großen Anteil der Datensätze ausmachen und dem Wissen über die starke Diskrepanz zwischen Layout- und Inhaltstabellen in Hinblick auf Strukturmerkmale, erschien es sinnvoll, die Unterteilung der Klassifizierung in 2 Teilphasen zu prüfen. Ähnlich zu Cafarella et al. [2], die auch einen zweistufigen Ansatz verfolgten. Daher wurde das zweistufige Klassifizierungsproblem definiert und dessen Teilprobleme in die Betrachtung miteinbezogen.

Beim Evaluieren der Lernalgorithmen wurde das folgende, auf dem Kreuzvalidierungsverfahren basierende Prinzip angewandt: Die Trainingsdaten wurden zunächst per Zufallsprinzip in zwei Teile geteilt, sodass ein Teil aus 90% der Ursprungsmenge und der andere aus 10% der Ursprungsmenge bestand. Dann wurde auf dem größeren Teil der Lernalgorithmus trainiert und anschließend auf dem kleineren Teil geprüft. Bei der Prüfung auf dem kleineren Teil wurde der fertig trainierte Lernalgorithmus auf jede Instanz angewendet und gegen das Label geprüft. Die Teilung der Trainingsdaten und das Prüfen des Lernalgorithmus wurden jeweils für jede Disziplin 100 Mal durchgeführt und der Anteil der korrekten Vorhersagen gemittelt, welcher im folgenden prozentual als Genauigkeit angegeben wird. Zudem wurde die Laufzeit, die der Lernalgorithmus für die Klassifizierung eines Datums benötigte, gemessen und über den gesamten Satz gemittelt.

	Genauigkeit in Prozent			Laufzeit in Millisekunden		
	Problem 1	**Problem 2**	**Problem 3**	**Problem 1**	**Problem 2**	**Problem 3**
SimpleCart	77,738	91,744	79,453	0,181	0,102	0,107
J48	76,496	91,035	77,075	0,982	0,519	0,511
RandomForest	80,563	92,718	80,471	2,223	1,636	1,874
SMO	78,493	90,814	77,643	6,310	1,837	3,219
Baseline	-	69,896	-	-	192,290	-

Tabelle 5.1: Vergleich der Lernalgorithmen für die drei Klassifizierungsprobleme unter Nutzung aller Features (Problem 1 = Einstufig, Problem 2 = Zweistufig Vorsortierung, Problem 3 = Zweistufig Hauptphase)

Einstufig	**Layout**	**Relation**	**Entity**	**Matrix**	**Other**	**Gesamt (gewichtet)**
SimpleCart	78,308	83,049	85,528	5,801	58,760	77,739
J48	79,957	74,200	85,226	25,691	62,365	76,496
RandomForest	83,731	85,680	87,991	15,193	56,813	80,563
SMO	83,514	81,525	85,157	23,757	56,669	78,493

Tabelle 5.2: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.1 für das einstufige Klassifizierungsproblem

Zweistufig Phase 1	**Layout**	**Table**	**Gesamt (gewichtet)**
SimpleCart	88,585	94,933	91,744
J48	89,693	92,437	91,035
RandomForest	91,630	93,792	92,718
SMO	88,004	93,642	90,814

Tabelle 5.3: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.1 für die Vorsortierungsphase des zweistufigen Klassifizierungsproblems

Zweistufig Phase 2	**Relation**	**Entity**	**Matrix**	**Other**	**Gesamt (gewichtet)**
SimpleCart	86,710	86,817	10,086	58,514	79,453
J48	77,041	87,050	20,173	61,824	77,075
RandomForest	86,618	89,189	12,680	57,635	80,471
SMO	81,031	86,864	23,055	56,554	77,643

Tabelle 5.4: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.1 für die Hauptphase des zweistufigen Klassifizierungsproblems

5.2.1 Einstufiges Klassifizierungsproblem

Für das einstufige Klassifizierungsproblem wurden keine Änderungen an den Labels der Trainingsdaten vorgenommen. Es wurde lediglich die Menge der als Layout eingestuften Eingabedaten begrenzt, sodass diese im Trainingsdatensatz nicht überwiegen und den Lernalgorithmus negativ beeinflussen: würde der Algorithmus bei einem Testdatensatz, der zu 90% aus Layout-Tabellen besteht, einfach alle Daten blind der Layout-Klasse zuordnen, hätte er immer noch in 90% der Fälle die korrekte Klasse gewählt, was jedoch in Wahrheit keine verlässliche Aussage über die tatsächliche Genauigkeit wäre. Um dies zu verhindern, wurde die Anzahl innerhalb aller Nicht-Layout-Typen gemittelt und als Beschränkung für die Layout-Einträge gesetzt. Es wurden per Zufallsprinzip solange Layout-Einträge aus dem Datensatz entfernt, bis die Beschränkung erfüllt war.

Tabelle 5.1 zeigt, dass in dieser Disziplin der RandomForest Algorithmus mit etwa 80,56% die höchste Genauigkeit bietet. Er liegt jedoch in Hinblick auf die Laufzeit weit hinter den anderen beiden Entscheidungsbaum-Algorithmen zurück. So sind SimpleCart und J48 zwar jeweils um wenige Prozent ungenauer als RandomForest, benötigen jedoch höchstens die Hälfte der Rechenzeit bei der Klassifizierung. SimpleCart ist hierbei zusätzlich um ein Fünffaches schneller als J48 und bietet dazu noch eine etwas höhere Genauigkeit im Vergleich zu J48. Der SMO-Algorithmus liegt in Bezug auf die Genauigkeit knapp unter RandomForest und in etwa auf dem Niveau von SimpleCart, hat jedoch mit Abstand die höchste Laufzeit von allen Algorithmen.

5.2.2 Zweistufiges Klassifizierungsproblem: Vorsortierungsphase

Für die Vorsortierungsphase des zweistufigen Klassifizierungsproblems wurde die Unterteilung auf zwei Klassen beschränkt: Layout und Nicht-Layout. Letztere wurde durch die Ersetzung aller Inhaltstypen-Labels durch Nicht-Layout-Labels innerhalb des Datensatzes gebildet. Zusätzlich wurde auch hier die Menge der Layout-Tabellen ähnlich 5.2.1 beschränkt und an die Menge der Nicht-Layout-Labels angeglichen. Außerdem wurde die Klassifizierung durch den regelbasierten Algorithmus für jede Tabelle aufgezeichnet und ebenfalls in den Vergleich mit einbezogen.

Tabelle 5.1 zeigt durchgehend Werte über 90% für die Genauigkeit aller Lernalgorithmen bei der Klassifizierung. Alle 4 Algorithmen übertreffen damit den regelbasierten Baseline-Algorithmus außerdem um etwa 20%. Der RandomForest-Algorithmus bietet mit etwa 92,72% die höchste Genauigkeit, jedoch die schlechteste Laufzeit nach SMO. SimpleCart ist sowohl genauer als J48 und SMO als auch um ein vielfaches schneller bei der Klassifizierung. Bei der Genauigkeit ist SimpleCart jedoch um etwa 1% schlechter im Vergleich zu RandomForest. Der SMO-Algorithmus hat die höchste Laufzeit und weist die geringste Genauigkeit der vier Algorithmen auf.

5.2.3 Zweistufiges Klassifizierungsproblem: Hauptphase

Bei der Hauptphase des zweistufigen Klassifizierungsproblems wurden alle als Layout gelabelten Daten aus dem Trainingsdatensatz verworfen und lediglich die verbleibenden Inhaltstabellen-Klassen betrachtet.

Ähnlich den vorigen beiden Betrachtungen, bietet der RandomForest Algorithmus erneut die höchste Genauigkeit. Der SimpleCart Algorithmus ist lediglich um 1% schlechter als RandomForest, bietet dafür aber eine Laufzeit die weniger als 6% der von RandomForest beträgt. J48 ist sowohl in der Genauigkeit als auch in der Laufzeit schlechter als SimpleCart. Der SMO-Algorithmus gliedert sich in Bezug auf die Genauigkeit zwischen SimpleCart und J48 ein, weist aber die höchste Laufzeit aller 4 Algorithmen auf.

5.2.4 Interpretation der Ergebnisse

Gesamtgenauigkeiten und Laufzeiten. Innerhalb der verschiedenen Klassifizierungsprobleme und -phasen zeigen alle Lernalgorithmen allgemein eine hohe Genauigkeit, die stets über 75% liegt. Damit erzielen sie alle eine höhere Genauigkeit als simples Raten der Klasse. Außerdem übertreffen sie in der Vorsortierungsphase des zweistufigen Problems den zum Vergleich herangezogenen regelbasierten Baseline-Algorithmus um mindestens 20% im Hinblick auf korrekte Klassifizierungen. Der RandomForest-Algorithmus erzielt in allen 3 Betrachtungen die höchste Genauigkeit, benötigt aber stets ein vielfaches der Laufzeit von J48 und SimpleCart. SimpleCart ist hierbei nur geringfügig schlechter in der Genauigkeit als RandomForest, dafür jedoch am schnellsten in der Laufzeit und überbietet zudem J48 in allen Kategorien. Der SMO-Algorithmus hat in allen Disziplinen die höchste Laufzeit, liegt aber in Bezug auf die Genauigkeit noch hinter RandomForest zurück.

Teilproblemzerlegung. Es wird deutlich, dass die Zerlegung der Klassifizierung in Teilprobleme beim zweistufigen Verfahren einen positiven Einfluss auf die Genauigkeit hat, da sich die Lernalgorithmen stärker spezialisieren können als bei dem einstufigen Klassifizierungsproblem. Zieht man zusätzlich in Betracht, dass etwa 96,5% des Datensatzes aus Layout-Tabellen besteht (vgl. 5.1), fällt die Vorsortierungsphase weit stärker ins Gewicht als die Hauptphase. Damit bietet die Aneinanderkettung der beiden Teilprobleme in 5.2.2 und 5.2.3 einen entscheidenden Vorteil im Gegensatz zu der alleinigen Anwendung des einstufigen Verfahrens. So hat die Vorsortierungphase eine um etwa 10% höhere Genauigkeit als die Hauptphase. Layout-Tabellen, die in der Vorsortierungsphase als solche erkannt werden, können sofort verworfen werden, da sie nicht von inhaltlicher Bedeutung sind. Durch den hohen Anteil an Layout-Tabellen, erzielt der zweistufige Ansatz damit im Gesamten eine höhere Genauigkeit.

Genauigkeitsverteilung auf Klassen. Im Hinblick auf die Tabellen 5.2, 5.3 und 5.4 zur Verteilung der Genauigkeit auf die einzelnen Klassen sind die dort dargestellten Werte für Matrix-Tabellen nicht besonders repräsentativ im Vergleich zu den anderen Klassen, zieht man die Verteilung im Testdatensatz aus Abbildung 5.1 mit in Betracht. Geht man von der beschriebenen 90/10-Teilung des Kreuzvalidierungsverfahrens aus, so verbleiben im 10%-Teil, auf dem die Klassifizierung der Algorithmen geprüft wird, nur etwa drei bis vier Matrix-Tabellen pro Durchlauf und fallen damit deutlich geringer aus als die anderen Typen. Da die Algorithmen jeweils auf die gleichen 90/10-Teilungen geprüft wurden, lassen sich jedoch zwischen den Algorithmen Vergleiche führen. So fällt auf, dass SimpleCart die geringste Erkennungsrate für Matrix-Tabellen aufweist. Auch RandomForest, welcher sonst im Großteil der Fälle die höchste Genauigkeit erzielt, liegt hierbei hinter J48 und SMO zurück.

Die Genauigkeit für "sonstige" Tabellen, liegt deutlich unter den anderen Genauigkeiten, ausgenommen der für Matrix-Tabellen. Da diese Tabellenart all jene Tabellen beschreibt, die nicht einer der anderen Klassen zugeteilt wurden, gibt es für diese "sonstigen" Tabellen keine genaue Definition. Ihre Struktur ist folglich einer größeren Varianz ausgesetzt und die Lernalgorithmen können sich weniger gut auf deren Struktur spezialisieren. Da für unseren Anwendungsfall die Weiterverarbeitung der Tabellen im Vordergrund steht, ist die Klasse der "sonstigen" Tabellen aufgrund ihrer unklaren Definition jedoch weniger bedeutend. Ihre gezielte Weiterverarbeitung würde demnach eine zusätzliche Erkennung ihrer Strukturen erfordern.

Fazit. Die Entscheidung für einen Algorithmus kann nur unter Abwägung verschiedener Faktoren erfolgen, da keiner der Algorithmen in allen Gesichtspunkten stets die beste Wahl ist. Allgemein stehen SimpleCart und RandomForest auf den oberen Plätzen. Für unseren Anwendungsfall ist die Laufzeit bei der Klassifizierung ein wichtiges Kriterium, da diese über einen Datensatz von Millionen Tabellen erfolgen soll. Hierbei wäre SimpleCart die erste Wahl, da dessen Laufzeiten nur ein Bruchteil der anderen beträgt. Es gilt abzuwägen, ob es die geringe Verbesserung in der Genauigkeit wert ist, die vergleichsweise hohen Laufzeiten des RandomForest-Algorithmus in Kauf zu nehmen. Dieser wiederum wäre die erste Wahl, steht die Genauigkeit kompromisslos im Vordergrund. Die anderen beiden Algorithmen J48 und SMO sind lediglich dann von Interesse, wenn die Erkennung von Matrix-Tabellen eine zentrale Rolle einnimmt. Außerdem muss geprüft werden, ob die hier gemessene Laufzeit im Verhältnis zur gesamten Klassifizierungsdauer überhaupt ins Gewicht fällt, da die Feauture-Berechnung keinen unwesentlichen Anteil an der Gesamtlaufzeit hat. Dazu werden im folgenden Abschnitt zusätzlich die Laufzeiten der Features betrachtet.

5.3 FEATURE SUBSET SELECTION

Der folgende Abschnitt bespricht die gewählten Ansätze zur Feature-Selektion sowie deren Einfluss bei der Klassifikation durch die Lernalgorithmen. Dabei werden Selektionsmengen für die einzelnen Teilprobleme erstellt und gegeneinander verglichen, sowie die Ergebnisse gegen die eingangs erwähnten Ansprüche an die Selektion geprüft. Anschließend werden auch die Laufzeiten der einzelnen Features bei deren Berechnung evaluiert.

Zunächst wurde ein einfacher Greedy-Algorithmus zur Feature-Selektion auf Basis des Wrapper-Ansatzes entwickelt. Der Java-Pseudocode für den Algorithmus ist in Abbildung 5.4 gezeigt. Der Algorithmus baut schrittweise eine Menge an Features auf. Dabei beginnt er mit einer leeren Menge an Features und fügt pro Schleifendurchlauf genau das Feature aus der Menge der verfügbaren Features hinzu, welches zusammen mit der bisherigen Auswahl (anfangs leer) die höchste Bewertung erreicht und die Bewertung der bisherigen Auswahl übertrumpft. Zur Bewertung benutzt der Algorithmus den resultierenden Genauigkeitswert des Lernalgorithmus über eine 10-fach Kreuzvalidierung auf dem Trainingsdatensatz. Findet er kein Feature, welches zusammen mit der bisherigen Auswahl eine bessere Bewertung erzielt als die bisherige Auswahl selbst, bricht er ab und gibt die bisherige Auswahl als selektierte Feature-Menge zurück. Der Algorithmus ist also nur in der Lage, das erste lokale Maximum zu finden.

```
public ArrayList<Attribute> selectFeatures() {

        // Alle möglichen Attribute (Features)
        ArrayList<Attribute> availableAttributes = getAvailableAttributes();

        // Attribute, die während der Selektion ausgewählt werden
        ArrayList<Attribute> selectedAttributes = new ArrayList<Attribute>();

        double lastPct = 0.0;

        while (availableAttributes.size() > 0) {
                double bestPct = 0.0;

                // beste Kombination aus bisheriger Selektion und einem neuen
                // Attribut
                ArrayList<Attribute> bestAttributeSelection = new ArrayList<
                    Attribute>();

                // durchsuche die Menge aller verbleibenden Attribute nach
                // demjenigen, welches zusammen mit der bisherigen Selektion
                // die höchste Genauigkeit bei der Kreuzvalidierung erzielt
                for (Attribute a : availableAttributes) {

                        // temporäre Zusammenstellung
                        ArrayList<Attribute> proposals = new ArrayList<
                            Attribute>();

                        // füge Attribut a zu den bisher selektierten
                        // Attributen hinzu und prüfe die Genauigkeit
                        // des Klassifikators unter Nutzung dieser
                        // Menge mittels Kreuzvalidierung
                        proposals.addAll(selectedAttributes);
                        proposals.add(a);
                        double currentPct = crossValidate(proposals);

                        // ist die Genauigkeit besser als die der bisher besten
                        // temporären Zusammenstellung, speichere dies als neue
                        // beste Zusammenstellung für den aktuellen Durchlauf
                        if (currentPct > bestPct) {
                                bestPct = currentPct;
                                bestAttributeSelection.clear();
                                bestAttributeSelection.addAll(proposals);
                        }
                }

                // Prüfe, ob das Hinzufügen des oben ermittelten Attributs eine
                // Verbesserung gegenüber der bisherigen Selektion bringt
                if (bestPct > lastPct) {
                        // wenn ja, füge es der Selektion hinzu
                        for (Attribute a : bestAttributeSelection) {
                                availableAttributes.remove(a);
                                selectedAttributes.add(a);
                        }
                        lastPct = bestPct;
                } else {
                        // andernfalls, brich die Suche ab
                        break;
                }
        }

        return selectedAttributes;
}
```

Abbildung 5.4: Java-Pseudocode für den eigenen Greedy-Algorithmus

5.3.1 Ergebnisse der Selektion

In Tabelle 5.5 sind die Resultate des eigenen Greedy-Selektionsalgorithmus unter Nutzung des RandomForest-Lernalgorithmus dargestellt. Trotz der drastischen Reduzierung der Feature-Mengen, schneidet RandomForest mit der Selektion gut ab. Für das einstufige Problem und die Vorsortierungsphase des zweistufigen liegt hier die Genauigkeit nur 1,5 bis 2 Prozent unter den Werten ohne Selektion. Für die Hauptphase des zweistufigen Problems ist sogar eine, wenn auch vernachlässigbar geringe, Verbesserung von 0,1 Prozent erkennbar.

Da es jedoch Ziel war, keine Verschlechterung der Ergebnisse durch die Feature-Selektion in Kauf zu nehmen, wurde zusätzlich die Correlation Feature Subset Selection, kurz CFS, betrachtet. Es galt zu überprüfen, ob die Aussage, dass CFS keine Verschlechterung der Genauigkeiten ergeben würde [5], für den vorliegenden Fall zutreffend ist.
Daher wurde der CFS-Algorithmus auf die Feature-Mengen der einzelnen Probleme angewandt. Aufgrund der unterschiedlichen Klassenaufteilung der ein- und zweistufigen Klassifizierungsprobleme, ergibt die Selektion per CFS unterschiedliche Selektionsmengen. Tabelle 5.5 zeigt die Klassifizierungsergebnisse für die Selektion durch CFS unter Anwendung des RandomForest-Algorithmus. Es wird deutlich, dass die Selektion durch CFS für alle Problemstellungen eine geringfügige Verbesserung erzielt als ohne Selektion. Zudem werden etwa 75% aller Features verworfen. Somit wird CFS seinen eigenen Anforderungen gerecht: keine Verschlechterung der Ergebnisse und Verwurf von über 50% der Features [5] und genügt damit auch unserem Anspruch in Hinblick auf die Genauigkeit der Klassifizierungsergebnisse.
Um die Werte besser einordnen zu können, wurde der Feature-Satz von Cafarella et al. gemäß Abbildung 5.5 nachgebildet und zusammen mit dem RandomForest-Algorithmus ebenfalls auf den Testdaten geprüft. Die Ergebnisse wurden in Tabelle 5.5 eingegliedert und zeigen durchgängig schlechtere Ergebnisse als die restlichen Selektionen. In der ursprünglichen Arbeit wurde die Feature-Menge als Filtermechanismus für relationale Tabellen benutzt, was in etwa der Vorsortierungsphase unseres zweistufigen Klassifizierungsproblems entspricht. Doch selbst im direkten Vergleich bei diesem Teilproblem, liegt die Selektion von Cafarella et al. hinter den anderen zurück.

Da der CFS-Algorithmus auf dem Filter-Ansatz der Feature Subset Selection beruht, ist er von der Wahl des Lernalgorithmus gänzlich unabhängig. Die Selektion kann daher auf alle Lernalgorithmen gleichermaßen angewendet werden. In den Tabellen 5.6, 5.7, 5.8 und 5.9 sind die Resultate der verschiedenen Lernalgorithmen unter Verwendung der durch CFS reduzierten Feature-Menge analog zu Abschnitt 5.2 dargestellt.
Vergleicht man mit den Ergebnissen für alle Features, fallen keine wesentlichen Unterschiede auf. Der RandomForest-Algorithmus erzielt auch hier die besten Genauigkeiten und bleibt daher die beste Wahl. Auch bei Betrachtung der Aufteilung der Ergebnisse pro Tabellenklasse gibt es durch die CFS-Selektion keine Verschlechterung, außer für Entity-Tabellen beim einstufigen Klassifizierungsproblem um etwa 1,5%. Ein Großteil der Werte liegen wenige Prozent über denen aus Abschnitt 5.2.

	Genauigkeit in Prozent			Anzahl selektierter Features		
	Problem 1	**Problem 2**	**Problem 3**	**Problem 1**	**Problem 2**	**Problem 3**
ohne Selektion	80,563	92,718	80,471	127	127	127
eigener Greedy	78,589	91,269	80,562	9	8	5
CFS	80,809	92,988	82,122	34	26	23
CFS (ohne lokal)	77,764	91,670	76,594	15	11	10
Cafarella 2008	73,018	87,824	73,712	7	7	7

Tabelle 5.5: Vergleich zwischen den resultierenden Genauigkeiten bei RandomForest durch die verschiedenen Feature-Selektionsalgorithmen und der Featureselektion zur relationalen Filterung aus Abbildung 5.5 von Cafarella 2008 (Problem 1 = Einstufig, Problem 2 = Zweistufig Vorsortierung, Problem 3 = Zweistufig Hauptphase)

	Genauigkeit in Prozent			Laufzeit in Millisekunden		
	Problem 1	**Problem 2**	**Problem 3**	**Problem 1**	**Problem 2**	**Problem 3**
SimpleCart	75,721	92,057	79,556	0,189	0,094	0,109
J48	76,631	91,521	78,908	1,172	0,464	0,496
RandomForest	80,809	92,988	82,122	2,239	1,830	2,260
SMO	72,467	89,885	74,466	2,490	0,739	0,939
Baseline	-	70,107	-	-	192,290	-

Tabelle 5.6: Vergleich der Lernalgorithmen für die drei Klassifizierungsprobleme unter Nutzung der Feature Subset Selection des CFS-Algorithmus (Problem 1 = Einstufig, Problem 2 = Zweistufig Vorsortierung, Problem 3 = Zweistufig Hauptphase)

Einstufig	**Layout**	**Relation**	**Entity**	**Matrix**	**Other**	**Gesamt (gewichtet)**
SimpleCart	78,813	79,297	84,687	9,275	52,573	75,721
J48	81,758	75,689	84,544	20,000	60,975	76,631
RandomForest	85,714	86,388	86,388	20,000	57,163	80,809
SMO	83,956	73,852	79,145	0,000	49,0264	72,467

Tabelle 5.7: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.6 für das einstufige Klassifizierungsproblem unter Nutzung der durch CFS reduzierten Feature-Menge

Zweistufig Phase 1	**Layout**	**Table**	**Gesamt (gewichtet)**
SimpleCart	88,972	95,176	92,057
J48	89,684	93,406	91,521
RandomForest	92,015	94,011	92,988
SMO	86,026	93,752	89,885

Tabelle 5.8: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.6 für die Vorsortierungsphase des zweistufigen Klassifizierungsproblems unter Nutzung der durch CFS reduzierten Feature-Menge

Zweistufig Phase 2	**Relation**	**Entity**	**Matrix**	**Other**	**Gesamt (gewichtet)**
SimpleCart	89,338	86,239	5,479	55,882	79,556
J48	80,315	87,687	21,918	63,056	78,908
RandomForest	87,823	90,650	15,616	59,900	82,123
SMO	75,804	87,135	0,000	51,291	74,466

Tabelle 5.9: Genauigkeit der Algorithmen pro Klasse in Prozent sowie die Gesamtgenauigkeit aus Tabelle 5.6 für die Hauptphase des zweistufigen Klassifizierungsproblems unter Nutzung der durch CFS reduzierten Feature-Menge

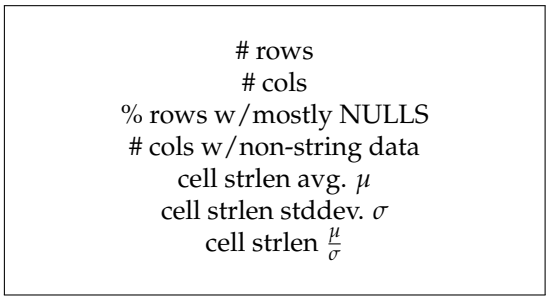

Abbildung 5.5: Satz selektierter Features für relationale Filterung aus Cafarella 2008 [2]

5.3.2 Laufzeiten

Um den Einfluss der Feature-Berechnung auf die Laufzeit des gesamten Klassifizierungsvorgangs beurteilen zu können, wurden die durchschnittlichen Berechnungszeiten für die einzelnen Features gemessen. Hierfür wurden alle Features über insgesamt 40.000 Tabellen eines Common Crawl Archivs berechnet. Dabei hatten die Tabellen im Schnitt eine Breite von 3,28 Spalten und eine Länge von 6,49 Zeilen. Es wurden die Ausführungszeiten der einzelnen Feature-Berechnungen aufgezeichnet und arithmetisch gemittelt. Die Messergebnisse sind in Tabelle 5.10 abgetragen. Da die Berechnung der Inhaltsanteile aus 4.2.4 und 4.3.2 gebündelt erfolgte, sind diese Features in der Tabelle als Gruppe `RATIO_X` beziehungsweise `LOCAL_RATIO_X` gelistet.
Beim Betrachten der Messwerte fallen sofort 2 wichtige Aspekte auf. Zum einen liegen die Laufzeitwerte der Featureberechnungen in völlig anderen Größenordnungen als die Laufzeitwerte der Classifier aus Tabelle 5.1. Die Betrachtung des in 5.2.4 erwähnten Laufzeit-Tradeoffs zwischen RandomForest und SimpleCart ist somit hinfällig, da die Laufzeiten der Klassifikationen im Verhältnis zur Feature-Berechnung vernachlässigbar klein sind.
Zum anderen nimmt die Gruppe der lokalen Inhaltsanteile etwa zwei Drittel der gesamten Berechnungszeit in Anspruch. In Tabelle 5.5 sind zusätzlich die Resultate für eine CFS-Selektion unter Ausschluss der dieser Feature-Gruppe abgetragen. Es wird deutlich, dass diese Feature-Gruppe einen erkennbaren Einfluss auf die Genauigkeit des Lernalgorithmus hat, insbesondere für die Unterscheidung der Inhaltstabellen (Problem 3). Auch der eigene Greedy-Algorithmus für die Selektion nahm Features aus der Gruppe der lokalen Inhaltsanteile in die selektierten Mengen für alle 3 Probleme auf und bildet somit keine Alternative.

Feature	CFS-Selektion	Laufzeit in Millisekunden
MAX_COLS	3	300
AVG_COLS	1, 2, 3	311
AVG_ROWS	1, 2	325
MAX_ROWS	-	379
LOCAL_SPAN_RATIO	1, 2	446
STD_DEV_ROWS	3	1.725
STD_DEV_COLS	3	2.91
HAS_HEADER	1	4.123
AVG_CELL_LENGTH	3	16.078
MAX_CELL_LENGTH	-	16.429
CUMULATIVE_LENGTH_CONSISTENCY	-	17.275
STD_DEV_CELL_LENGTH	-	18.104
LOCAL_AVG_LENGTH	1, 2, 3	33.634
LOCAL_LENGTH_VARIANCE	1, 2, 3	41.398
RATIO_X	1, 2, 3	149.352
CUMULATIVE_CONTENT_CONSISTENCY	1, 3	153.683
LOCAL_RATIO_X	1, 2, 3	803.188

Tabelle 5.10: Mittlere Laufzeit der einzelnen Features über 40.000 Tabellen und Selektionsergebnisse durch CFS nach Teilproblem: 1 = Einstufiges Klassifizierungsproblem, 2 = Zweistufig Vorsortierung, 3 = Zweistufig Hauptphase

6 SCHLUSSBEMERKUNG

6.1 ZUSAMMENFASSUNG

Ziel der Arbeit war es, für den Anwendungsfall der Extraktion von Tabellen aus dem World Wide Web, eine passende Methodik zu finden um die Tabellen ihrer Struktur nach zu klassifizieren und dadurch die gezielte Weiterverarbeitung zu gewährleisten. Dazu wurden bisherige Arbeiten zu dieser Thematik betrachtet und darauf aufbauend für den konkreten Anwendungsfall zu unterscheidende Tabellentypen definiert, eine Sammlung von mathematischen Features zur Beschreibung der Tabellen erarbeitet und ebenfalls der Ansatz des maschinellen Lernens gewählt, um die Problemstellung zu lösen.
Mithilfe von Webseitendaten aus den Archiven der Common Crawl Foundation wurde ein Trainigsdatensatz erstellt, anhand dessen die vorgestellten Algorithmen des maschinellen Lernens evaluiert werden konnten. Es konnte gezeigt werden, dass der RandomForest-Algorithmus, welcher auf Entscheidungsbäumen basiert, die beste Genauigkeit erzielte und somit den geeignetsten Lernalgorithmus darstellt. Es zeichnete sich zwar ab, dass die guten Ergebnisse des RandomForest-Algorithmus im Gegensatz zu den anderen Lernalgorithmen oft einen Tradeoff in der Laufzeit der Klassifizierung mit sich brachten. Dieser konnte jedoch als vernachlässigbar eingestuft werden, da die Laufzeiten der Algorithmen einen Bruchteil im Vergleich zur Berechnungszeit der Features darstellten.
Darüber hinaus konnte gezeigt werden, dass eine Teilung des Klassifizierungsproblems in zwei Stufen bessere Ergebnisse erzielt als ein reines einstufiges Klassifizierungsverfahren. Aufgrund des hohen Anteils von über 90 Prozent an nicht inhaltstragenden Layout-Tabellen im Web, wurde eine binäre Vorklassifikation zwischen Layout- und Inhaltstabellen vorgestellt. An diese wurde eine Hauptklassifikation gegliedert, welche nur die Inhaltstabellen aus der Vorklassifikation betrachtet und diese ihrer Struktur nach klassifiziert. Es wurde ersichtlich, dass die Vorklassifikation durch ihre Spezialisierung eine höhere Genauigkeit aufwies als die Gesamtklassifikation beim einstufigen Verfahren. Da über 90 Prozent der Eingangsdaten die zweite Phase des zweistufigen Verfahrens nicht durchlaufen, verbesserte sich hierdurch die Gesamtgenauigkeit im Vergleich zum einstufigen Verfahren.
Unter der Zielsetzung der Laufzeitverringerung und Erhöhung der Genauigkeit bei der Klassifizierung wurden Ansätze zur Selektion von Features evaluiert um die Menge an zu berech-

nenden Werten zu reduzieren. Dabei zeigte sich, dass die Selektion anhand eines einfachen Greedy-Algorithmus zwar die Feature-Menge drastisch reduzierte, die Ergebnisse jedoch ebenso verschlechterte. Der "Correlation Feature Subset Selection"-Algorithmus von Mark A. Hall hingegen konnte sich behaupten, indem er die Feature-Menge ebenfalls stark reduzierte, dabei aber die Ergebnisse geringfügig verbesserte. Eine drastische Laufzeitverringerung konnte durch die Selektion nicht erreicht werden, da jene Features, welche die größte Berechnungszeit benötigten, von beiden Ansätzen in die Selektion aufgenommen wurden. Messungen unter Ausschluss dieser Features haben gezeigt, dass diese einen nicht unwesentlichen Beitrag zur Zuverlässigkeit der Klassifikation beitragen. Aufgrund der Forderung, durch Selektion keine Verschlechterung der Ergebnisse in Kauf zu nehmen, wurden diese Features letztlich in der Selektion beibehalten.

6.2 AUSBLICK

Die zusammengestellte Methodik der vorliegenden Ausarbeitung erzielt bereits sehr gute Ergebnisse. Für eine gezielte Verarbeitung wäre jedoch eine 100 prozentige Genauigkeit nötig. Falsch klassifizierte Tabellen erschweren die Weiterverarbeitung und müssen abgefangen werden. Des Weiteren sind die potentiellen Informationen innerhalb dieser Tabellen verloren, da sie mangels geeigneter Weiterverarbeitungsvorschrift verworfen werden müssen. Um diese Fehlerquelle und damit den Verlust von Informationen zu reduzieren, ist eine weitere Verbesserung der Erkennungsrate vonnöten. Aufbauend auf dem vorgestellten, zweistufigen Klassifizierungsverfahren lassen sich weitere Optimierungen an den gewählten Methodiken vornehmen.

Aufgrund des geringen Anteils an Matrix-Tabellen und der geringen Größe des Trainingsdatensatzes waren die Messungen in Bezug auf Matrix-Tabellen nicht sehr repräsentativ. Eine Erweiterung der Testdatensatzgröße und ein besserer Ausgleich der Anteile der Tabellentypen, insbesondere der Matrix-Tabellen, wäre ein primärer Ansatz um die Klassifizierung durch die trainierten Lernalgorithmen zuverlässiger und ihre Bewertung repräsentativer zu gestalten.

Für die Reduzierung der Feature-Menge durch Selektion wurde mit dem CFS-Algorithmus ein generischer Filter-Ansatz gewählt, der den Ansprüchen der Problemstellung zwar genügte aber keine Spezialisierung auf den gegebenen Anwendungsfall darstellt, da er die Feature-Werte unabhängig vom Sachverhalt betrachtet. Die Entwicklung eines Selektionsverfahrens, welches die Bedeutung der Features und ihre Relevanz zur konkreten Problemstellung miteinbezieht wäre womöglich in der Lage, eine auf den Anwendungsfall besser abgestimmte Selektion vorzunehmen und damit gegebenenfalls Laufzeit und Genauigkeit zu verbessern.

Nicht zuletzt wäre auch eine genauere Betrachtung der Lernalgorithmen und insbesondere ihrer Parameter ein weiterer Ansatzpunkt, die Ergebnisse weiter zu verbessern, da der Lernalgorithmus das Kernstück des gesamten Ansatzes bildet.

LITERATURVERZEICHNIS

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Michael J Cafarella, Alon Y Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. In *WebDB*. Citeseer, 2008.

[3] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 545–554, New York, NY, USA, 2011. ACM.

[4] Common Crawl Foundation. Common crawl, 2014. [Online; Stand 27. Oktober 2014].

[5] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[6] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.

[7] Larissa R. Lautert, Marcelo M. Scheidt, and Carina F. Dorneles. Web table taxonomy and formalization. *SIGMOD Rec.*, 42(3):28–33, October 2013.

[8] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[9] The University of Waikato. Weka 3: Data mining software in java, 2014. [Online; Stand 21. November 2014].

[10] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[11] Dan Steinberg and Phillip Colla. Cart: classification and regression trees. *The Top Ten Algorithms in Data Mining*, 9:179, 2009.

[12] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 242–250, New York, NY, USA, 2002. ACM.