Automatische Einkategorisierung von Produktdaten in einen Kategorienbaum

Mapping EAN data into a category tree

Bachelor-Thesis von Mansur Iqbal

Juni 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Automatische Einkategorisierung von Produktdaten in einen Kategorienbaum
Mapping EAN data into a category tree

Vorgelegte Bachelor-Thesis von Mansur Iqbal

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten: Christian Wirth

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 23.06.2014

(Mansur Iqbal)

Zusammenfassung

Beim Online-Shop von der Hitmeister GmbH sollten digitale Produktdaten anhand vorliegender Beschreibungen und Eigenschaften automatisch in die jeweils richtige Kategorie eines bereits neu erstellten, hierarchischen Kategorienbaums eingeordnet werden. Diese Arbeit beschäftigt sich mit der Suche nach einem geeigneten Verfahren um die Einkategorisierung maschinell durchzuführen. Es wurde ein Programm geschrieben um die vorhanden, bereits richtig kategorisierten Produktdaten in ein gültiges Format für das Programm *Weka* zu konvertieren. Mit diesem Programm wurden verschiedene gängige Klassifikations-Verfahren aus dem Machine-Learning Bereich getestet um die beste Methode zur automatischen Kategorisierung zu finden. Um die Validierung des kompletten Datensatzes effizient auszuführen, wurde zudem die Effektivität verschiedener Attributreduktions-Verfahren in Weka untersucht. Das beste Reduktions-Verfahren erwies sich als ineffizient, um die Anzahl der Attribute des gesamten Datensatzes reduzieren zu können. Auf der Suche nach alternativen Algorithmen ging der Klassifizierer LIBLINEAR hierbei als ein sehr effizientes Verfahren hervor. Damit wurde es möglich einen großen Datensatz ohne Reduktion der Attribute in sehr kurzer Zeit zu klassifizieren.

Inhaltsverzeichnis

1 Einleitung

Einkaufen im Internet ist heute eine alltägliche Tätigkeit. Auf sogenannten *e-Commerce* Plattformen, wie zum Beispiel Amazon[1], Ebay[2] oder Hitmeister[3] kann man verschiedene Produkte aus den unterschiedlichsten Kategorien erwerben und zu sich nach Hause liefern lassen. Ein potentieller Kunde[4], welcher gezielt nach einem Produkt sucht, kann dazu die Webseiten-Suche verwenden oder sich entlang eines Kategorienbaums (siehe Abschnitt 3.1) zu seinem gesuchten Artikel durchnavigieren. Meistens können auch zusätzliche Einschränkungen von Produkteigenschaften wie zum Beispiel die Breite eines Bettes vorgenommen werden, um das Warenangebot der Webseite gezielt nach einem möglichen Wunschprodukt zu filtern. Die Produkteigenschaften werden dabei als *Attribute* bezeichnet. Der Kunde erhält eine Liste von möglichen Produkten, welche jeweils anhand eines Titels, eines Bildes und dem Preis präsentiert werden. Nach der Auswahl eines Produktes erhält der Kunde eine detailliertere Darstellung der Produkteigenschaften auf der zugehörigen, sich öffnenden Webseite.

Im Rahmen der Umstrukturierung des Produktdatensystems der Hitmeister GmbH, werden unter anderem die alten Produkt-Kategorien verfeinert und in eine übersichtlichere hierarchische Struktur gebracht. Die bereits im Angebot befindlichen Produkte müssen dafür aus den alten Kategorien in den neuen Kategorienbaum einkategorisiert werden. Zur Minimierung des manuellen Aufwands soll diese „Um-Kategorisierung“ mithilfe eines maschinellen Verfahrens durchgeführt werden.

Ziel dieser Arbeit ist es, ein geeignetes Klassifikationsverfahren für das oben beschriebene Vorhaben zu finden. Weil die endgültigen Kategorien bereits festgelegt sind, muss dafür zunächst ein kleiner Anteil aller Produkte manuell in die neuen Kategorien einsortiert werden (Überwachtes Lernen Kap. 2.1). Anhand der textuellen Beschreibungen der Produkte werden anschließend verschiedene Klassifikationsmethoden *trainiert* und getestet. Beim *Trainieren* lernen die Methoden welche Produkt-Beschreibungen in welche Kategorien passen und speichern diese Informationen in einem sogenannten *Modell* ab. In einem gelernten Modell kann dann geprüft werden, wie viele von den manuell zugeordneten Produkten von dem Klassifizierer richtig zugeordnet werden. Auf der Grundlage der Anzahlen der korrekten Zuordnungen und der Effizienz einer Modell-Berechnung wird entschieden, welche Klassifikationsmethode sich für weitere Vorgehen wie eine Optimierung am besten eignet.

[1] `http://www.amazon.de`

[2] `http://www.ebay.de`

[3] `http://www.hitmeister.de`

[4] Werden Personenbezeichnungen aus Gründen der besseren Lesbarkeit lediglich in der männlichen oder weiblichen Form verwendet, so schließt dies das jeweils andere Geschlecht mit ein.

1.1 Hitmeister GmbH

Die e-Commerce Plattform www.hitmeister.de existiert seit 2007 und dient auch gleichzeitig als *Marktplatz*, da verschiedene Händler sich dort anmelden können um ihre Waren zum Verkauf anzubieten. Anfangs wurden dort nur Medienprodukte wie Filme, Spiele und Bücher angeboten. Aufgrund des rasanten Wachstums der Plattform erweiterte sich auch die Produktpalette immer stärker, sodass Hitmeister heute Produkte aus allen Bereichen anbietet. Das Unternehmen hat seinen Sitz in Köln und beschäftigt mittlerweile über 60 Mitarbeiter. Vier feste Mitarbeiter und einige Werkstudenten bereiten die wichtigsten Produktdaten auf und pflegen die Kategorien und Attribute auf der Homepage.

Welche Daten benötigt werden hängt von den Produkten ab, die die Händler anbieten möchten. Die *Produktangebote* werden von den Händlern über verschiedene Wege ins Hitmeister-System importiert und durch die *European Article Number*[5] (EAN) identifiziert. Dabei geben die Händler die verfügbare Anzahl ihrer Angebote und den Preis an. Um diese Angebote auf der Webseite für den Kunden darstellen zu können, werden Produktdaten wie beispielsweise Titel, Beschreibung und Bilder benötigt. Diese werden von professionellen Datenanbietern (hier *Content Provider* genannt), von den Händlern selbst oder auch direkt von Warenherstellern, mit denen Hitmeister zusammen arbeitet, geliefert. Wenn keine Produktdaten zu einer EAN gefunden werden können, kann der entsprechende Artikel demnach nicht angeboten werden und wird auf der Plattform nicht angezeigt. Aktuell hat Hitmeister über fünf Millionen Artikel im Angebot. Von 30% der eingegangen Angeboten von Händlern fehlen die Produktdaten, womit diese Artikel nicht angeboten werden können.

Wie bereits erwähnt, war das ursprüngliche Software-System nur für Medien gedacht. Aufgrund der ansteigenden Produktzahlen skalierte die Plattform nicht mehr gut und es kam zeitweise, unter anderem bedingt durch die hohe Last auf der Datenbank, zu Serverausfällen. Aufgrund dessen wurde entschieden das System grundlegend zu überarbeiten und skalierbarer zu gestalten. Ein weiterer Grund dafür ist, dass die Händler auf Hitmeister viele unterschiedliche und teilweise fehlerhafte Daten über Produkte importieren konnten, worunter die Datenqualität erheblich litt und weshalb ein manueller Prozess benötigt wurde um nur qualitative Daten freizugeben. Dieses Problem sollte mit dem neuen Produktdaten-System mittels einem zum Teil automatisierten Prüfungs- und Freigabe-Prozess behoben werden.

[5] Die EAN ist eine 8 bis 13 Ziffern große Zahl, die ein Handelsartikel international eindeutig kennzeichnet. `https://de.wikipedia.org/wiki/European_Article_Number`

2 Grundlagen

2.1 Maschinelles Lernen

Maschinelles Lernen ist ein typischer Begriff für Verfahren, die in einer Menge von Beispiel-Daten Gesetzmäßigkeiten erlernen und verallgemeinern können. Praktisch gesehen versucht man mit Hilfe von Computern große Datenmengen zu analysieren und daraus Schlussfolgerungen zu ziehen oder zukünftige Verhalten vorherzusagen.

Es gibt zwei grundlegende algorithmische Ansätze: [Nil14, 1.2.1 S.5]

Überwachtes Lernen: Dabei erlernen die Algorithmen eine Abbildung zwischen einer Eingabe X und Ausgabe Y, um dadurch selber Voraussagen treffen zu können. Dazu gehören die *Regression* und die *Klassifikation*. Die Regression versucht aus vorhandenen Zahlenwerten zu lernen und Vorhersagen in Form von neuen Zahlenwerten zu treffen. Bei der Klassifikation hingegen werden auf bereits gekennzeichneten Beispielen Funktionen oder Verknüpfungen zwischen den Daten gelernt, anhand derer später unbekannte Beispiele gekennzeichnet werden können. Die Kennzeichnungen nennt man *Klassen*.

Unüberwachtes Lernen: Bei diesem Ansatz versucht der Algorithmus anhand von Beispielen Muster in den Daten zu erkennen und diese in unterschiedliche Kategorien zu partitionieren. Im Vergleich zum überwachten Lernen sind die Ausgabewerte hierbei nicht bekannt. Das Verfahren *Clusteranalyse* versucht solche Regelmäßigkeiten und Muster in den Eingabedaten zu finden und daraus Cluster oder Gruppierungen zu erstellen.

Da in dieser Arbeit bereits die Klassen in Form von Kategorien im Kategorienbaum vorgegeben sind, wird hier nur das überwachte Lernen beziehungsweise die Klassifikation betrachtet. Die Beispiel-Daten auf denen ein Algorithmus lernt sind immer eine Ansammlung von Schlüssel-Wert-Paaren, die auch *Features* (siehe 2.2) genannt werden. Die Beispiel-Dokumente unterscheiden sich demnach durch die Werte in den Features und können dadurch von den Algorithmen charakterisiert werden.

Als Beispiel werden in Tabelle 2.1 erfundene Daten von einem bestimmten Zeitraum über das Wetter aufgeführt. In der ersten Zeile stehen die möglichen Attribute bzw. *Features*. Die letzte Spalte ist immer die *Klasse*, die unseren Zielwert beschreibt. Jede Zeile ist dabei ein einzelnes *Dokument*. Anhand der Dokumente vom 01.04. bis zum 07.04. weiß man wie das Wetter an den einzelnen Tagen war und in der Klassenspalte ist durch ein „Ja“ oder „Nein“ beschrieben, ob man an diesem Tag Tennis spielen konnte. Ein Algorithmus würde nun versuchen anhand der Daten zu lernen, mit welchen Werten man Tennis spielen kann und mit welchen nicht. Daraus würde ein Modell erstellt werden. Wenn man in ein Programm dann die Werte der letzten beiden Dokumente eingeben würde, würde es durch das Modell spekulativ vorhersagen können (*klassifizieren*), ob man an diesen beiden Tagen Tennis spielen kann oder nicht.

Tabelle 2.1: Maschinelles Lernen - Beispiel Daten

Datum	Temperatur	Aussicht	Windstärke	Tennis spielen?
01.04.	15°C	Sonnig	Stark	Nein
02.04.	16°C	Wolkig	Sehr stark	Nein
03.04.	18°C	Wolkig	Mittel	Ja
04.04.	13°C	Wolkig	Niedrig	Nein
05.04.	15°C	Sonnig	Mittel	Ja
06.04.	20°C	Sonnig	Niedrig	Ja
07.04.	30°C	Sonnig	Kein	Nein
08.04.	26°C	Sonnig	Mittel	?
09.04.	13°C	Wolkig	Stark	?

Es gibt unterschiedliche Methoden im Bereich maschinellen Lernens. Die gängigsten aus der Praxis, die auch hier betrachtet werden, sind:

Entscheidungsbäume: Bei dieser Methode erstellt der Algorithmus anhand der Eigenschaften aus ihm gegebenen Beispielen geordnete, gerichtete und hierarchische Bäume. Die Knoten enthalten Testregeln (IF-THEN-Regeln), anhand derer für ein zu prüfendes Dokument im Baum traversiert wird bis der Algorithmus in einem Blatt angelangt ist. Die Blätter in den Bäumen enthalten dabei die erkannte Klasse. Die gängigsten speziellen Algorithmen aus diesem Bereich sind *Random Forest* (siehe 2.3.5) und *J48* (siehe 2.3.4).

Regelinduktion: Verhält sich ähnlich den Entscheidungsbäumen. Es werden dabei auch verschiedene IF-THEN-Regeln erzeugt. „[...]die Regelinduktion [ist] eine Suche, die sich zuerst in die Tiefe bewegt und einen Pfad (eine Regel) pro Schritt generiert, wohingegen die Bauminduktion sich vornehmlich in die Breite bewegt und alle Pfade simultan generiert.“ [AL08, 9.5, S.197] Der gängige Algorithmus dafür ist *JRip* (siehe 2.3.3).

Support Vector Machines: Eine SVM setzt die Features der Dokumente in einen mathematischen Raum und versucht dort zwischen den Werten eine Trennebene zu errechnen. Dokumente, die später klassifiziert werden sollen, werden dann einem, durch die Ebene getrennten, Bereich zugeordnet und dadurch einer Klasse zugewiesen. Im hier benutzten Tool wird dazu der *SMO* (siehe 2.3.2) Klassifizierer benutzt.

Bayes: Beim Bayes-Klassifikator wird mit Wahrscheinlichkeiten gerechnet. Die Features der Dokumente werden dabei wie bei der SVM in einen mathematischen Raum gesetzt und es wird für jeden Punkt im Raum eine Wahrscheinlichkeit mittels einer Funktion festgesetzt, die eine bestimmte Klasse beschreibt. (siehe 2.3.1)

2.2 Features & Klassen

Wie bereits im Abschnitt 2 erwähnt, bestehen die einzelnen Daten-Dokumente aus Schlüssel-Wert-Paaren, wobei die Schlüssel *Features* genannt werden. Die Werte für die Features sind intern bei den meisten Algorithmen numerisch. Um textuelle Dokumente zu verarbeiten, müssen die Texte zunächst in numerische Werte geparst werden. Man bedient sich der sogenannten *Automatischen Indexierung*[1]. Dazu wird ein Text zunächst mit einem *Tokenizer* verarbeitet, der die Wörter üblicherweise nach Leerzeichen zerlegt. Anschließend können die einzelnen Wörter bzw. *Tokens* zur besseren Analyse „gestemmt“, d.h. auf ihren gemeinsamen Wortstamm zurückgeführt[2] werden. Aus der Häufigkeit des Vorkommens des Tokens wird ein sogenanntes *Tf-idf-Maß*[3] bestimmt.

2.2.1 TF-IDF

Das TF-IDF-Maß wird aus zwei Komponenten bestimmt, die miteinander multipliziert werden. Die erste Komponente (TF) gibt die normalisierte Häufigkeit des Wortes (oder auch *Terms*) im Dokument, dividiert durch die Anzahl aller Wörter im Dokument wieder. Die zweite Komponente (IDF) errechnet sich aus dem Logarithmus vom Verhältnis der Gesamtanzahl der Dokumente zu der Anzahl derjenigen Dokumente, in denen das Wort vorkommt.

TF oder auch **term frequency** misst wie oft ein Wort t in einem Dokument vorkommt. Da das Wort in längeren Dokumenten häufiger vorkommen kann als in kürzeren, wird der Wert normalisiert, indem er durch die Gesamtzahl der Wörter im Dokument geteilt wird. In Weka (siehe 2.5) wird die logarithmisch skalierte Häufigkeit benutzt[4].[MRS08, S.100]

$$tf_{t,d} = \log(f(t,d)+1) \tag{2.1}$$

IDF oder auch **inverse document frequency** misst die Relevanz eines Wortes. Häufige Wörter in der Sprache, die in vielen Dokumenten vorkommen wie beispielsweise der Artikel „das“ erscheinen nach der Komponente TF als wichtig, sind aber nicht aussagekräftig und enthalten keine Hinweise, zu welcher Klasse das Dokument gehören könnte. Um diese Wörter abzustufen und die Relevanz wichtiger Wörter zu erhöhen, wird die folgende Formel angewendet. N ist dabei die Gesamtanzahl der Dokumente und n_t die Anzahl der Dokumente, in denen das Wort t vorkommt.

$$idf_t = \log\frac{N}{n_t} \tag{2.2}$$

1 Siehe `https://de.wikipedia.org/wiki/Automatische_Indexierung`
2 Siehe auch `https://de.wikipedia.org/wiki/Stemming`
3 Siehe auch `https://de.wikipedia.org/wiki/Tf-idf-Ma%C3%9F`
4 Siehe Weka Sourcecode von der StringToWordVector Klasse `http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/StringToWordVector.html`

Das Gewicht w eines Wortes t im Dokument d ist dann nach TF-IDF:

$$w_{t,d} = tf_{t,d} \cdot idf_t \tag{2.3}$$

Quelle: [BYRN99]

2.3 Klassifizierung

Unter Klassifizierung versteht man das Zuordnen von Klassen zu Dokumenten anhand deren Attribute. Sie kann generell in zwei Phasen unterteilt werden: die Trainingsphase und die Testphase.

In der Trainingsphase geht es darum, die Zusammenhänge zwischen unterschiedlichen Attributen zu erlernen und diesen Klassen zuzuweisen. Ein ausgewählter Algorithmus bekommt eine Sammlung von Trainingsdaten mit klassifizierten Dokumenten und generalisiert sie entsprechend zu einem Modell. Für ein neues Dokument kann der Algorithmus mit diesem Modell die wahrscheinlichste Klasse vorhersagen.

In der Testphase wird auf Testdaten mit bereits klassifizierten Dokumenten, die aber nicht in den Trainingsdaten vorkommen, das errechnete Modell geprüft. Dabei wird für jedes Test-Dokument die Klasse berechnet und mit der Klasse aus den ursprünglichen Daten verglichen. Das prozentuale Verhältnis von richtig vorhergesagten Dokumenten zu den falsch vorhergesagten Dokumenten nennt man *Trefferquote* oder auch *accuracy*.

Die unterschiedlichen Algorithmen, die in dieser Arbeit Anwendung finden, werden in den nächsten Abschnitten näher erläutert.

2.3.1 Naive Bayes

Implementierung in Weka nach George H. John und Pat Langley [JL95]. Die Idee bei Naive Bayes ist das Erlernen von Wahrscheinlichkeiten zwischen Attributwerten und den Klassen. Nach dem Satz von Bayes (siehe 2.4) wird die *a-posteriori-Wahrscheinlichkeit* $P(C|x)$ anhand der vorher angelernten Wahrscheinlichkeiten $p(x|C)$ und den Wahrscheinlichkeiten $p(x)$ für die jeweilige Klasse C berechnet. Das geschieht für jedes Attribut x und wird zum Schluss für jede Klasse summiert. Die Klasse C mit der höchsten Wahrscheinlichkeit wird somit als Treffer ausgegeben.

$$P(C|x) = \frac{P(C)p(x|C)}{p(x)} \tag{2.4}$$

2.3.2 SMO

Implementierung in Weka nach J. Platt [Pla98], S.S. Keerthi [KSBM01], Trevor Hastie und Robert Tibshirani [HT98]. *Sequential Minimal Optimization* (SMO) ist eine Verbesserung des üblichen SVM-Algorithmus. Dieses Verfahren ist meist schneller und besser skalierbar. Um die Vorteile von SMO zu beschreiben, wird zunächst SVM betrachtet:

Wie bereits in 2 beschrieben, projiziert der Algorithmus zunächst verschiedene Vektoren in einen n-dimensionalen Raum. In diesen wird zwischen zwei verschiedenen Datenwolken versucht eine trennende Hyperebene einzusetzen, die den maximalen Abstand zu den trennenden Vektoren hat. Bei einem *linearen Klassifizierer* werden die zu klassifizierenden Objekte, die einen n-Dimensionalen Raum besitzen, mit einer $n-1$ dimensionalen Hyperebene getrennt. In Abbildung 2.1 wird dargestellt wie diese Ebenen aussehen können. Zu sehen sind Vektoren in einem 2-dimensionalen Raum getrennt durch 1-dimensionale Hyperebenen. Ebene E1 ist keine wirkliche Trennung, weil sie nicht beide Sätze von einander trennt. Ebene E2 trennt die beiden Sätze, hat aber nicht den maximalen Abstand zu den Vektoren. E3 ist die richtige Hyperebene mit maximalem Abstand zu beiden Datensätzen und wird auch die *optimale Trennebene* genannt.

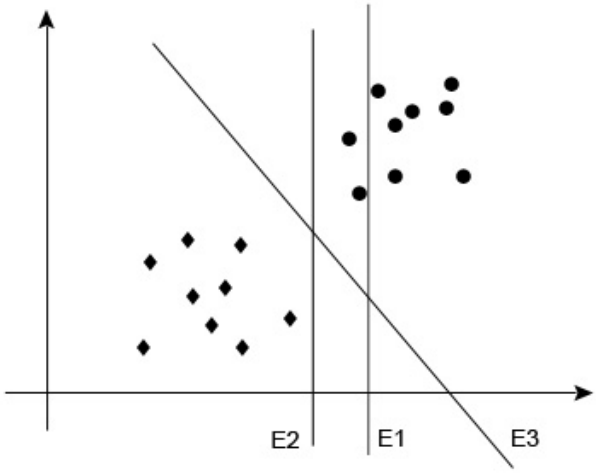

Abbildung 2.1: SVM trennende Hyperebenen Beispiel Skizze

SMO optimiert dabei die Findung des maximalen Abstands zwischen der Ebene und den Vektoren, indem es numerische Ansätze in einer Schleife durch analytische ersetzt. Zusätzlich benötigt es weniger Speicher, weil zwei bisher getrennt gespeicherte Vektoren als eine 2*2 Matrix gespeichert werden können. Detailiertere Ausführungen zu SMO in [Pla99].

2.3.3 JRip

Implementierung in Weka nach William W. Cohen [Coh95]. Der hierbei verwendete Algorithmus heißt *SLIPPER* und basiert auf dem *RIPPER* Algorithmus. RIPPER lernt dabei pro Klasse eine Regel indem es in einer Schleife Bedingungen zur Regel hinzufügt, um ein Maß für den Informationszuwachs zu maximieren, welches in Quinlans (1990) *Foil-Algorithmus* Verwendung findet

[AL08, S. 198]. Eine Regel ist eine Verkettung von mehreren Bedingungen mit boolschen Operatoren. Die Bedingungen werden hinzugefügt bis nur noch positive Beispiele für die jeweilige Klasse von der Regel abgedeckt werden. Danach wird diese Regel zurückgekürzt (*pruning*), um dadurch die Regel zu finden, die die *Regelbewertungsmetrik* maximiert. Die restlichen Beispiele werden demnach anderen Klassen zugeordnet, die wiederum ihre Regeln haben um positive Beispiele für die Klassen zu ermitteln. Die Beispiele, die durch keine Regel abgedeckt sind werden somit der letzten Klasse zugewiesen.

Der Vorteil am SLIPPER Algorithmus ist, dass dieser mittels Boosting [FS97] die Regelinduktion sehr vereinfacht, verständlicher macht und die Trefferquote effektiv verbessert. Der Nachteil ist, dass die erlernten Regeln zugunsten der Trefferquote nicht mehr gut verständlich sind.

2.3.4 J48

Implementierung in Weka nach Ross Quinlan [Qui93]. In Kapitel 2 wird bereits beschrieben wie Entscheidungsbäume grundlegend funktionieren. J48 ist genauer gesagt der Name einer Implementierung des *C4.5* Algorithmus. Dieser ist ähnlich wie der ältere *CART* Algorithmus, nur dass C4.5 eine beliebige Anzahl an Zweigen nutzt und nicht nur binär arbeitet wie CART. Außerdem beschneidet C4.5 im Gegensatz zu CART beim Pruning den Baum ohne auf die Datenbasis zu achten. CART erzeugt einige Bäume und testet diese vorher mit neuen, nicht klassifizierten Daten.

2.3.5 Random Forest

Implementierung in Weka nach Leo Breiman [Bre01]. Random Forest ist ebenfalls ein Algorithmus, der auf Entscheidungsbäumen (siehe 2) basiert. Die Idee bei Random Forest ist es, mehrere Entscheidungsbäume parallel zu erlernen. Für jede Klasse darf demnach jeder Entscheidungsbaum zur Klassifikation ausgewertet werden. Die Klasse mit den meisten Stimmen entscheidet dann über das endgültige Ergebnis. Die Trainingszeit dabei ist sehr kurz und steigt mit der Anzahl der Bäume nur linear an. Durch die Wahl der Parameter ist es möglich den Algorithmus sehr gut auf individuelle Datensätze anzupassen. Aufgrund des Gesetzes der großen Zahlen ist die *Überanpassung*[5] eines Datensatzes ausgeschlossen.

[5] Auch *overfitting* genannt. Siehe `https://de.wikipedia.org/wiki/%C3%9Cberanpassung`

2.4 Kreuzvalidierung (Cross-Validation)

Kreuzvalidierung verfolgt das Ziel möglichst präzise gelernte Modelle zu evaluieren. Dazu braucht man einen geeigneten Datensatz, der homogen zu den Trainingsdaten ist. Die unmittelbare Lösung dabei ist, dass man den ursprünglichen Trainingsdatensatz in zwei Teile aufteilt. Der eine wird zum Trainieren verwendet, der andere zum Testen und umgekehrt.

Bei der Kreuzvalidierung wird der Trainingsdatensatz in mehrere, möglichst gleich große Teile geteilt. Nacheinander werden alle Teile einmal getestet und auf den restlichen wird gelernt. Nach jedem Test wird die Trefferquote gespeichert. Wenn alle Datensatzteile getestet wurden, wird der Durchschnitt aller Trefferquoten gebildet und so die endgültige Quote berechnet.

Um die Varianz der Trefferquoten zu verringern und relativ genaue Aussagen zu gewinnen, bedient man sich dabei der *Stratifizierung*. Die stratifizierte Kreuzvalidierung achtet darauf, dass die Dokumente auf die einzelnen Klassen in allen Datensatzteilen annähernd gleich verteilt sind, damit keine falschen Trefferquoten ermittelt werden.

In dieser Arbeit wird zur Validierung der Modelle stets die 10-fache stratifizierte Kreuzvalidierung verwendet. Abbildung 2.2 stellt die fünf Iterationen einer 5-fachen Kreuzvalidierung dar.

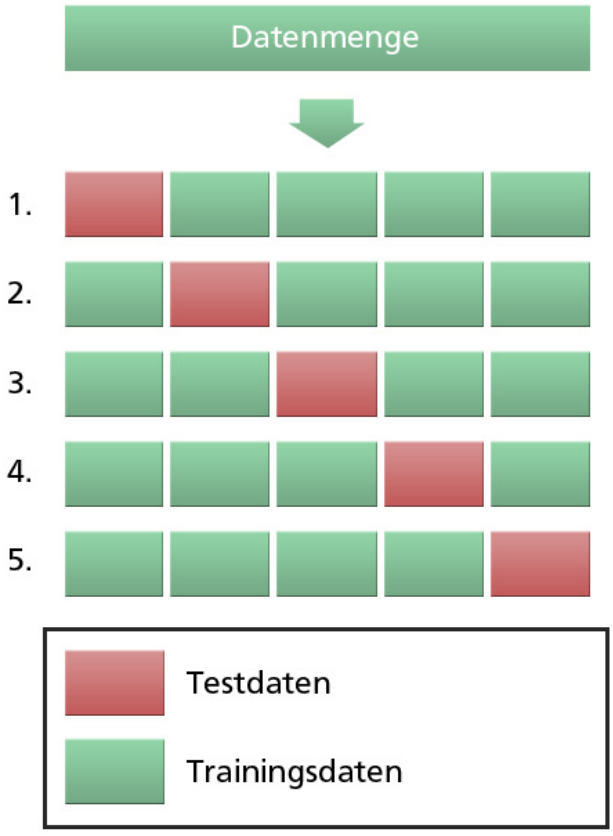

Abbildung 2.2: 5-fache Kreuzvalidierung

2.5 Weka

Die in der Programmiersprache Java geschriebene Software Weka (Waikato Environment for Knowledge Analysis) ist das gängigste Tool im Bereich des maschinellen Lernens. Es stellt viele verschiedene Verfahren zur Verfügung und ist dabei auch mit einer grafischen Oberfläche ausgestattet. Diese ermöglicht eine schnelle, praktische Datenanalyse und Datenverarbeitung. Das Programm liest dabei Dateien vom Typ *Attribute-Relation File Format* (*ARFF*) (siehe 2.5.1) ein und kann auch nach weiteren Verarbeitungen Daten in diesem Format speichern.

2.5.1 ARFF Format

Das Datei-Format wurde speziell für Weka entwickelt. Eine ARFF-Datei besteht aus zwei Bereichen: Einem Kopf-Teil und einem Daten-Teil. Im Kopf-Teil werden der Name via `@RELATION <relations-name>` und alle vorkommenden Attribute via `@attribute <attribut-name> <datentyp>` definiert. Die unterstützten Datentypen sind zurzeit:

- numeric
- <nominal-specification>
- string
- date [<date-format>]

Der Datentyp `numeric` wird für Ganzzahlen und Fließkommazahlen verwendet.

Der Datentyp `string` ist für komplette Texte vorgesehen.

Nominelle Attribut-Werte können mittels der `<nominal-specification>` beschrieben werden. Dabei werden alle möglichen nominellen Werte direkt hinter der Attributdefinition in geschweiften Klammern angegeben: `{<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}`

Im Datums Attribut können Daten in einem bestimmten Format wie folgt angegeben werden: `@attribute <name> date [<date-format>]`

Kommentare können mit einem Prozent-Zeichen eingeleitet werden und gelten bis zum Zeilenende.

Im Listing 2.1 ist ein Beispiel für eine ARFF-Datei.

Listing 2.1: Beispiel ARFF-Datei

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

3 Aufbau

3.1 Kategorienbaum

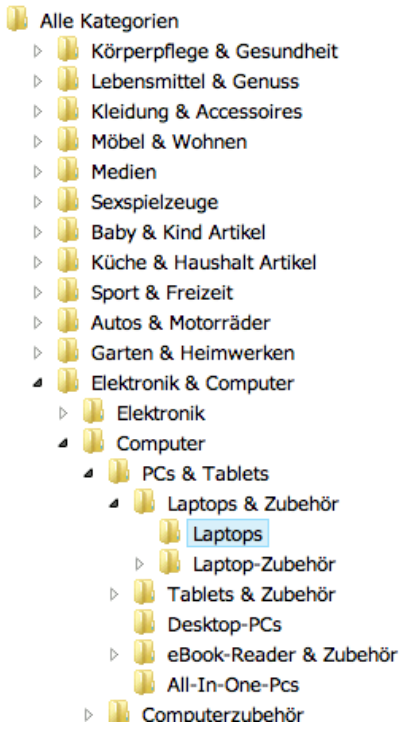

Abbildung 3.1: Kategorienbaum Auszug Laptops

Der neue Kategorienbaum auf Hitmeister hat über 5500 Knoten, davon sind aktuell 4640 Blätter. Zu Beginn wurde das Sortiment von Hitmeister betrachtet und daraus die Oberklassen gebildet. Unter dem Aspekt der *Suchmaschinenoptimierung*[1] (SEO) wurden dann die einzelnen Unterknoten nach und nach herausgebildet.

Um die Komplexität zu verringern, werden die Produkte aktuell nur in Blätter sortiert. Damit ist sichergestellt, dass diese eine eindeutige Zugehörigkeit haben und beim Navigieren auch erwartete und entsprechende Produkte angezeigt werden. Außerdem sind die Produktattribute je nach Kategorie unterschiedlich und können dementsprechend beim Verfeinern nach Attributen in der Navigation besser dargestellt werden. Jeder Knoten kann bestimmte Attribute wie zum Beispiel die Länge eines Bettes enthalten. Diese Attribute werden auch in die Knoten darunter vererbt. Der oberste Knoten „Alle Kategorien“ beinhaltet die wichtigsten Attribute, die jedes Produkt enthalten muss. Dazu gehören Titel, Beschreibung, Kurzbeschreibung, Bilder, EAN, MPN und Listenpreis. Die Angabe zum Hersteller des Produktes wird auch als ein Attribut angesehen, welches aber bei Produkten wie Bücher oder Filmen nicht vorkommt, weshalb es nur eine Ebene darunter angehängt ist. Diese Ebene wird als Welten bezeichnet und somit wird jede Ka-

[1] Siehe auch `https://de.wikipedia.org/wiki/Suchmaschinenoptimierung`

tegorie auf der ersten Ebene im Folgenden *Welt* genannt. Im Bild 3.1 wird der Kategorienbaum exemplarisch für die Produktklasse Laptops dargestellt.

3.2 Datensatz

Als Grundlage für diese Arbeit wurden Basisdatensätze zum Trainieren und Auswerten erstellt, indem Mitarbeiter von Hitmeister initial 25 Produkte pro Kategorie händisch zugeordnet haben. Es konnten jedoch mangels Angebot nicht in jeder Kategorie die vollen 25 klassifizierten Artikel erreicht werden, worunter die Präzision der maschinellen Verfahren litt. Somit entstanden **58432 zugewiesene Instanzen** in insgesamt **3342 Kategorien**. Im Schnitt sind das 17,5 Instanzen pro Kategorie.

Erste Test-Berechnungen auf dieser Menge an Instanzen stellten sich als nahezu unmöglich heraus, da diese zum Speicherüberlauf führten oder schlichtweg viel zu lange dauerten um alle Klassifizierer in einer 10-Fold-Cross-Validation zu validieren. Deshalb wurde der Datensatz auf mehrere Welten aufgeteilt, die wie folgt aussehen:

Tabelle 3.1: Produkt-Datensätze, Welten, ungefiltert

Welt	Instanzen	Klassen	Features
Elektronik & Computer	5602	278	59
Kleidung & Accessoires	3786	215	11
Körperpflege & Gesundheit	2020	162	11
Küche & Haushalt	11416	629	27
Lebensmittel & Genuss	1289	128	6
Möbel & Wohnen	4606	236	10

Die Datensätze wurden mittels einem selbstgeschriebenen Programm aus der Hitmeister-Datenbank in einzelne ARFF-Dateien (siehe 2.5.1) generiert. Die endgültige Datei für alle Kategorien und Attribute hatte die Größe von 48,3 MB. Die Features in den Datensätzen wurden aus den Daten von Hitmeister Attributen generiert. Zu den Grundattributen, die jedes Dokument haben sollte, gehören:

- Titel
- Beschreibung
- Kurzbeschreibung
- Hersteller

Dabei sind Titel, Beschreibung und Kurzbeschreibung Werte vom Typ Text in Weka. Sie wurden vorher in dem ARFF generierenden Programm wie folgt präpariert:

- Möglicher HTML-Code entfernt
- Alle Sonderzeichen entfernt
- Alle Großbuchstaben in Kleinbuchstaben umgewandelt
- Deutsche und englische Stoppwörter[2] entfernt
- Die Wörter mittels einem Stemmer[3] auf ihren Stamm gebracht

Der Text kann jedoch so nicht zur Klassifizierung benutzt werden und muss zunächst automatisch indexiert bzw. in numerische Features transformiert werden. Dies geschieht mit dem unüberwachten Attribut-Filter *StringToWordVector* von Weka, der auf die Text-Attribute angewendet wird. Die Einstellungen, die dort gegenüber den Standard-Einstellungen in dieser Arbeit verändert werden, stehen in der folgenden Tabelle:

Tabelle 3.2: StringToWordVector-Filter Einstellungen

Einstellung	**Wert**	**Beschreibung**
IDFTransform	True	Idf-Wert siehe Abschnitt 2.2.1
TFTransform	True	Tf-Wert siehe Abschnitt 2.2.1
minTermFreq	10	Behält nur die Wörter, die mindestens so häufig in der Klasse vorkommen
outputWordCounts	True	Anzahl der Wörter ausgeben, anstatt 0 oder 1 (was das Vorkommen des Wortes impliziert)

Nach dem Durchlauf des StringToWordVector-Filters verschwinden die gewählten Text-Features und wir haben für jedes Wort ein numerisches Feature mit zugehörigen Werten in Weka. Nach dem Anwenden des Filters auf die bisherigen Datensätze, sehen diese wie folgt aus:

Tabelle 3.3: Produkt-Datensätze, minTermFreq 10

Welt	**Instanzen**	**Klassen**	**Features**
Elektronik & Computer	5602	278	5510
Kleidung & Accessoires	3786	215	1654
Körperpflege & Gesundheit	2020	162	883
Küche & Haushalt	11416	629	4045
Lebensmittel & Genuss	1289	128	715
Möbel & Wohnen	4606	236	1998

[2] `https://de.wikipedia.org/wiki/Stoppwort`
[3] `https://de.wikipedia.org/wiki/Stemming`

3.3 Einstellungen

Es wird das Programm Weka, das in 2.5 vorgestellt wurde, in Version 3.7.10 benutzt. Wie in 2.4 bereits erwähnt wurde, wird zum Validieren eine stratizifierte 10-fache Kreuzvalidierung verwendet. Die einfachen Tests laufen auf einem Macbook Air mit 1,8 GHz Intel Core i7 und 4 Gb DDR3 RAM. Tests, die länger als drei Stunden laufen werden auf einem Server ausgeführt, der mit 32 x 2,0 GHz Intel(R) Xeon(R) CPU E5-2650 und 128 Gb RAM ausgestattet ist. Beides erfolgt in einer Java 1.7 Umgebung und auf einer 64-Bit Architektur.

4 Evaluation & Ergebnisse

4.1 Erste Ergebnisse

In den ersten Schritten wurden die Tests der verschiedenen Welten-Datensätze in jedem Klassifizierer mit Standardparametern durchgeführt. Dazu wurden die Ausgabelogs von Weka gespeichert. In Tabelle 4.1 sind die Ergebnisse mit der Trefferquote und der Dauer zum Trainieren und Testen eines Modells angegeben.

Nach ersten Tests wird sichtbar, dass Naive Bayes und Random Forest relativ schlechte Trefferquoten liefern und somit nicht als Klassifizierer für den gesamten Datensatz in Frage kommen. JRip hat verglichen mit SMO und J48 zwar annähernd ähnliche Quoten, ist aber wegen der langen Trainingsdauer auch unbrauchbar.

Wenn man die Zeiten zum Trainieren in den einzelnen Datensätzen vergleicht, erkennt man, dass die Berechenbarkeit vor allem mit der Anzahl der Attribute zusammenhängt. Zwischen dem Datensatz „Lebensmittel & Genuss“ und „Körperpflege & Gesundheit“ ist die Zahl der Instanzen im letzteren fast doppelt so hoch, wobei die Anzahl der Attribute sich kaum ändert. Demnach verändern sich die Trainings- und Testzeiten auch nur circa um das Zweifache. Wohingegen zwischen den Datensätzen „Kleidung & Accessoires“ und „Elektronik & Computer“ das Ausschlaggebende die Anzahl der Attribute ist, welche beim Letzteren mehr als drei mal so hoch ist, die Zahl der Instanzen jedoch nur 1,5 mal größer ist. Die Trainings- und Testzeiten sind im Datensatz „Elektronik & Computer“ um ein Vielfaches größer.

Um den kompletten Datensatz zu validieren wird im nächsten Abschnitt versucht die Anzahl der Features zu reduzieren.

[1] Speicherüberlauf java.lang.OutOfMemoryError: Requested array size exceeds VM limit
[2] Validierung nach mehr als 48 Stunden Laufzeit abgebrochen

Tabelle 4.1: Erste Ergebnisse nach Welten

Datensatz	Klassifizierer	Dauer Training	Dauer Test	Trefferquote
Elektronik & Computer 5602, 278, 5510	Naive Bayes	19.82 s	3584.55 s	64.9411 %
	SMO	-	-	OOM[1]
	JRip	-	-	>48h[2]
	J48	2019.78 s	13.49 s	63.7272 %
	Random Forest	42.74 s	1.85 s	58.2649 %
Kleidung & Accessoires 3786, 215, 1654	Naive Bayes	2.79 s	554.62 s	65.8214 %
	SMO	74.33 s	147.05 s	85.5784 %
	JRip	1751.86 s	0.32 s	82.5409 %
	J48	226.82 s	3.39 s	83.5446 %
	Random Forest	13.5 s	1.08 s	62.5462 %
Körperpflege & Gesundheit 2020, 162, 883	Naive Bayes	0.8 s	120.4 s	58.5644 %
	SMO	35.98 s	23.28 s	73.9109 %
	JRip	376.03 s	0.16 s	72.3762 %
	J48	42.4 s	0.99 s	74.2079 %
	Random Forest	5.5 s	0.29 s	61.1881 %
Küche & Haushalt 11416, 629, 4045	Naive Bayes	30.65 s	12095.55 s	65.6097 %
	SMO			OOM[1]
	JRip	40739.01 s	2.1 s	82.1566 %
	J48	6705.38 s	134.02 s	83.0851 %
	Random Forest	153.95 s	5.76 s	54.1959 %
Lebensmittel & Genuss 1289, 128, 715	Naive Bayes	0.47 s	49.02 s	61.7533 %
	SMO	23 s	6.96 s	75.3297 %
	JRip	118.34 s	0.11 s	69.2785 %
	J48	11.88 s	0.34 s	74.0109 %
	Random Forest	2.67 s	0.18 s	61.986 %
Möbel & Wohnen 4606, 236, 1998	Naive Bayes	4.33 s	869.58 s	70.647 %
	SMO	94.69 s	229.55 s	88.1676 %
	JRip	2425.12 s	0.38 s	88.0591 %
	J48	409.29 s	5.35 s	90.2736 %
	Random Forest	22.93 s	1.79 s	63.5693 %

4.2 Attribut-Reduktion

Es wäre unter anderem auch möglich die Anzahl der Instanzen zu reduzieren. Wenn jedoch diese Anzahl reduziert wird, gibt es weniger Daten auf denen gelernt werden kann und demnach würde die Trefferquote mit hoher Wahrscheinlichkeit sinken. Außerdem sind es bereits wenige Instanzen pro Klasse und in manchen Klassen ist nur eine Instanz drin.

Auch eine Reduktion der Klassen ist insofern nicht möglich, weil alle Klassen zum Auswerten benötigt werden. Wenn Klassen ausgelassen werden, würden keine Produkte in die entsprechenden Kategorien klassifiziert werden können.

Aufgrund dessen wird nach dem besten Verfahren gesucht, um die Menge der Attribute zu reduzieren. Es gibt in Weka einen *AttributeSelection*-Filter mit dem man mittels einer Evaluations- und einer Suchmethode, die Anzahl der Attribute auf eine manuell festgelegte Menge reduzieren kann. Die Suchmethode, die verwendet wird ist der Ranker. Für die Evaluierung der Tests wird die Menge der Attribute auf 500 reduziert und mit verschiedenen Evaluationsmethoden ausgewertet.

Die folgenden Evaluationsmethoden werden beim AttributeSelection-Filter verwendet. (Quelle ist die Code-Dokumentation[1] der jeweiligen Klassen in Weka.)

CorrelationAttributeEval: Evaluiert die Relevanz eines Attributs nach dem *Korrelationskoeffizient* (Pearson) zwischen der Größe eines Attribut-Werts und der jeweiligen Klasse pro Dokument-Instanz. Der Koeffizient berechnetet sich durch das Verhältnis der Kovarianz zu dem Produkt aus der Standardabweichung zweier Merkmale. Das Ergebnis ist eine Zahl zwischen -1 und 1. Wenn die Zahl näher an der 0 ist, ist es unwahrscheinlich, dass die Werte in einem Zusammenhang stehen. Anderenfalls bei -1 oder 1 sind die Werte mehr an einer Geraden ausgerichtet und zeigen einen möglichen Zusammenhang.

GainRatioAttributeEval: Evaluiert die Relevanz eines Attributs A durch das Messen der Zuwachsrate des Attribut-Wertes im Verhältnis zur Klasse C. Dabei (und im Folgenden) ist $H(X)$ die *Entropie* eines Wertes und $H(X|Y)$ die *bedingte Entropie*. [CT12]

$$GainR(C,A) = \frac{H(C) - H(C|A)}{H(A)} \tag{4.1}$$

InfoGainAttributeEval: Evaluiert die Relevanz eines Attributs A durch das Messen des *Informationszuwachses* im Verhältnis zur Klasse C. Der Informationszuwachs ist dabei die Differenz zwischen der Entropie der Klasse und der bedingten Entropie der Klasse und des Attribut-Wertes.

$$InfoGain(C,A) = H(C) - H(C|A) \tag{4.2}$$

[1] Weka-Dokumentation: `http://weka.sourceforge.net/doc.dev/index.html?weka/classifiers/package-tree.html`

OneRAttributeEval: Evaluiert den Wert eines Attributs mit Hilfe des *OneR*-Klassifizierers. Der OneR-Klassifizierer erstellt für jede Kreuzkombination aus Attributen und den Werten eine Regel mit dem Ergebnis der häufigsten auftretenden Klasse. Die Regel lautet: WENN Attribut = Wert DANN Klasse = häufigste Klasse. Die Regel wird auf alle Dokumente angewandt und dann die Fehlerquote gemessen. Zum Schluss werden die Attribute mit den niedrigsten Fehlerquoten ausgewählt. [BD06]

SymmetricalUncertAttributeEval: Evaluiert die Relevanz eines Attributs *A* mit Hilfe vom *Symmetrischen Unsicherheitskoeffizient*. Dieser ist wie folgt definiert:

$$SymmU(C,A) = \frac{2 * (H(C) - H(C|A))}{H(C) + H(A)} \tag{4.3}$$

In Tabellen 4.2 und 4.3 werden die Ergebnisse der Welten-Datensätze mit reduzierten Attributen veranschaulicht.

[2] Speicherüberlauf java.lang.OutOfMemoryError: Requested array size exceeds VM limit

Tabelle 4.2: Attribut Reduktion nach Welten für SMO

Datensatz	Methode	Dauer Training	Dauer Test	Trefferquote
Elektronik & Computer 5602, 278, 500	Correlation	118.86 s	368.03 s	65.6016 %
	GainRatio	136.91 s	281.04 s	68.3327 %
	InfoGain	120.37 s	272.33 s	68.4577 %
	OneR	123.33 s	308.52 s	66.4763 %
	SymUncert	145.71 s	310.65 s	68.1542 %
Kleidung & Accessoires 3786, 215, 500	Correlation	70.2 s	108.29 s	66.9308 %
	GainRatio	68.99 s	82.84 s	71.0777 %
	InfoGain	68.73 s	85.78 s	70.9456 %
	OneR	64.95 s	69.62 s	82.9107 %
	SymUncert	68.94 s	81.68 s	70.9192 %
Körperpflege & Gesundheit 2020, 162, 500	Correlation	37.11 s	22.35 s	65.7426 %
	GainRatio	36.88 s	17.81 s	69.8515 %
	InfoGain	37.15 s	17.61 s	69.9505 %
	OneR	36.5 s	16.42 s	71.0891 %
	SymUncert	36.88 s	18.59 s	69.9505 %
Küche & Haushalt 11416, 629, 500	Correlation			OOM[2]
	GainRatio	566.16 s	1284.58 s	51.5154 %
	InfoGain	562.87 s	1205.38 s	51.5417 %
	OneR	573.56 s	1920.96 s	66.7397 %
	SymUncert	571.51 s	1183.84 s	51.5417 %
Lebensmittel & Genuss 1289, 128, 500	Correlation	23.58 s	6.79 s	71.6059 %
	GainRatio	23.35 s	5.7 s	71.7611 %
	InfoGain	23.34 s	5.72 s	71.7611 %
	OneR	23.41 s	6.34 s	73.9333 %
	SymUncert	23.32 s	5.7 s	71.6835 %
Möbel & Wohnen 4606, 236, 500	Correlation	88.65 s	185.32 s	66.9779 %
	GainRatio	84.81 s	122.51 s	73.6865 %
	InfoGain	83.93 s	115.72 s	73.4694 %
	OneR	82.16 s	129.13 s	82.7182 %
	SymUncert	85.09 s	119.12 s	73.5345 %

Tabelle 4.3: Attribut Reduktion nach Welten für J48

Datensatz	Methode	Dauer Training	Dauer Test	Trefferquote
Elektronik & Computer 5602, 278, 500	Correlation	51.28 s	2.96 s	55.141 %
	GainRatio	48.82 s	4.46 s	57.3367 %
	InfoGain	57.2 s	5.42 s	57.5509 %
	OneR	62.3 s	5.82 s	55.1767 %
	SymUncert	57.83 s	5.69 s	57.1046 %
Kleidung & Accessoires 3786, 215, 500	Correlation	25.25 s	1.24 s	58.4522 %
	GainRatio	22.66 s	1.45 s	63.7084 %
	InfoGain	23.84 s	1.64 s	63.5763 %
	OneR	34.88 s	1.64 s	78.7903 %
	SymUncert	23.9 s	1.64 s	63.5763 %
Körperpflege & Gesundheit 2020, 162, 500	Correlation	19.14 s	0.72 s	62.9703 %
	GainRatio	17.8 s	0.7 s	68.4158 %
	InfoGain	18.1 s	0.73 s	68.3663 %
	OneR	20.07 s	0.75 s	70.9406 %
	SymUncert	18.1 s	0.74 s	68.3663 %
Küche & Haushalt 11416, 629, 500	Correlation	162.53 s	16.72 s	42.4229 %
	GainRatio	214.65 s	75.67 s	46.6188 %
	InfoGain	226.02 s	79.97 s	46.6801 %
	OneR	254.82 s	100.81 s	51.3577 %
	SymUncert	228.87 s	80.34 s	46.6713 %
Lebensmittel & Genuss 1289, 128, 500	Correlation	7.53 s	0.29 s	69.0458 %
	GainRatio	7.86 s	0.28 s	69.9767 %
	InfoGain	8.06 s	0.3 s	70.287 %
	OneR	7.85 s	0.29 s	72.3817 %
	SymUncert	8.03 s	0.3 s	70.287 %
Möbel & Wohnen 4606, 236, 500	Correlation	42.72 s	2.19 s	57.056 %
	GainRatio	41.49 s	3.05 s	66.6522 %
	InfoGain	43.04 s	3.38 s	67.5423 %
	OneR	57.34 s	3.04 s	79.3313 %
	SymUncert	42.86 s	3.36 s	67.5423 %

Der Vergleich der Trefferquoten von SMO und J48 vor und nach der Reduktion (Tabelle 4.4) zeigt auf, dass SMO nach einer Reduktion immer noch viel präzisier arbeitet als J48.

Tabelle 4.4: Vergleich Trefferquoten vor und nach der Reduktion

Welt	SMO	SMO-OneR	J48	J48-OneR
Elektronik & Computer	-	66.4763 %	63.7272 %	55.1767 %
Kleidung & Accessoires	85.5784 %	82.9107 %	83.5446 %	78.7903 %
Körperpflege & Gesundheit	73.9109 %	71.0891 %	74.2079 %	70.9406 %
Küche & Haushalt	-	66.7397 %	83.0851 %	51.3577 %
Lebensmittel & Genuss	75.3297 %	73.9333 %	74.0109 %	72.3817 %
Möbel & Wohnen	88.1676 %	82.7182 %	90.2736 %	79.3313 %

4.3 Kompletter Datensatz

Zunächst müssen die Textattribute mit dem StringToVector-Filter zu Attributen umgewandelt werden. Daraus werden bei einer minTermFreq-Einstellung von 10 insgesamt 18462 Attribute im Datensatz generiert. Wie schon am Anfang erwähnt ist es nicht möglich diesen Datensatz in einer vernünftigen Zeit zu validieren. Mit der OneR Attribut Reduktion ist eine Klassifikation mit relativ hohen Trefferquoten möglich, aber allein die Reduktion des Datensatzes wurde auch nach drei Wochen Laufzeit nicht fertig.

Somit blieb nur die Option sich nach alternativen, effizienteren Algorithmen umzuschauen. *LIBLINEAR* ist ein linearer SVM-Algorithmus, der speziell für hochskalierte Klassifikationen entwickelt wurde. Dieses Verfahren wird im folgenden Abschnitt näher beschrieben.

4.3.1 LIBLINEAR

LIBLINEAR ist eine Ansammlung von verschiedenen linearen Klassifizierern. Grundsätzlich sind es zwei binäre Klassifizierer: LR und SVM. Der SVM-Klassifizierer allerdings in 2 Varianten: L2-loss und L1-loss. LIBLINEAR ist sehr effizient in der Berechnung von großen Datensätzen. Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang und Chih-Jen Lin geben in ihrem Paper [FCH$^+$08] an, dass ein Datensatz (Reuters Corpus Volume 1) mit über 600.000 Instanzen in nur wenigen Sekunden trainiert ist. Hingegen dauert derselbe Vorgang mit einem üblichen SVM Klassifizierer wie *LIBSVM* mehrere Stunden.

Aus den Vergleichen von [FCH$^+$08, Kap. 4] lässt sich erschließen, dass die LIBLINEAR-L2 Variante am schnellsten und präzisesten ist.

4.3.2 Ergebnisse

Der Wrapper von LIBLINEAR für Weka kann keine Nominal-Attribute und Attribute mit fehlenden Werten interpretieren, weshalb der Datensatz noch mal verarbeitet werden musste. Dazu wurden die nominellen Attribute mittels des *NominalToBinary* Filters in binäre Attribute umgewandelt. Danach wurden mittels des *ReplaceMissingWithUserConstant* Filters alle fehlenden Werte durch eine 0 ersetzt. Dadurch ist die Anzahl der Attribute von 18462 auf 23746 gestiegen.

Die Validierung mit LIBLINEAR auf dem gesamten Datensatz (minTermFreq 10) dauerte ungefähr zwölf Stunden. Das einfache Training dauert hingegen nur 3908,11 Sekunden bzw. 65 Minuten, was im Fall von Hitmeister mit Hinblick darauf, dass zukünftig mehr Produkte und Kategorien eingepflegt werden bzw. auch manuell zugewiesen werden sollen, eine akzeptable Dauer ist.

Die 10-fache Kreuzvalidierung mit LIBLINEAR und Standardeinstellungen erbrachte eine Trefferquote von 82,5754%. Mit Optimierung des Parameters cost (C) auf 0,01 wurde sogar eine Trefferquote von 86,5365% erreicht. Die Trefferquoten von LIBLINEAR sind somit sogar höher als die von J48 und SMO, obwohl der Klassifizierer mehr auf Effizienz ausgelegt ist als auf Präzision.

5 Fazit & Ausblick

5.1 Analyse

Die finale Trefferquote bedeutet letztendlich, dass statistisch gesehen im Schnitt 86,5365% aller Produkte in die richtigen Kategorien zugeordnet werden können. Es bedeutet aber nicht, dass dies abschließend auch bei über fünf Millionen Produkten auch genau so funktionieren wird. Die Produktdaten sind sehr unterschiedlich und viele Produkte haben nur den Titel als Referenz. Es wurde auf einer kleinen Untermenge gelernt und die erlernten Wörter kommen nicht unbedingt in der restlichen, zu klassifizierenden Menge vor. Dadurch würde eines dieser Produkte mit nur fehlenden Wörtern keinen Treffer zu irgendeiner Kategorie bekommen.

Eine weitere Schwierigkeit bereiten die Medienprodukte. Bücher, Filme und Spiele machen den größten Teil der Angebote aus und weil deren Beschreibungen unterschiedliche Wörter aus den übrigen Kategorien beinhalten können, ist es schwerer diese als Medien zu identifizieren. Als Beispiel könnte ein Sachbuch über Bohrmaschinen in die Kategorie der Bohrmaschinen zugeordnet werden.

Bei über fünf Millionen Produkten würden ohne die oben genannten Probleme ca. 700.000 Produkte falsch zugeordnet werden, was sich auf der Plattform sehr bemerkbar machen kann, wenn in Kategorien mit wenigen Produkten einige Falsche erscheinen.

5.2 Fazit

Aus den ersten Ergebnissen geht durch die stark variierenden Trefferquoten in den gesplitteten Datensätzen hervor, dass die Qualität der Daten maßgebend entscheidend ist. Darüber hinaus sind SVM und Entscheidungsbäume gegenüber Regel-Induktion und Bayes die besseren Verfahren zur Klassifizierung von Produktdaten.

Die Ergebnisse aus den Tests mit den reduzierten Attributen deuten eher darauf hin, dass Entscheidungsbäume stärker von der Anzahl der Attribute abhängen, als dies bei SVM der Fall ist. SMO hatte nach der Reduktion der Attribute auf 500 weniger Verluste in der Trefferquote als J48. Die meisten Reduktionsverfahren, OneR ausgenommen, sind ebenfalls abhängig von der Datenqualität. Dies wird dadurch deutlich, dass OneR das einzige Verfahren ist, welches einen Klassifizierer mit Kreuzvalidierung benutzt um die Relevanz eines Attributs festzustellen. Die anderen Methoden haben eine simplere Berechnung der Relevanz. Dadurch hat OneR jedoch im Vergleich zu den anderen Methoden eine sehr viel höhere Rechenzeit.

Eine vielversprechende Alternative sind die LIBLINEAR Klassifizierer. Die Effizienz der Umsetzung reicht aus, um den kompletten bestehenden Produkt-Datensatz zu berechnen und darüber hinaus mit zukünftig größeren Datensätzen rechnen zu können.

5.3 Ausblick

Um die Ergebnisse der Arbeit in die Praxis umzusetzen, muss als nächstes der Trainings-Prozess bei Hitmeister automatisiert werden. Dazu würde jede Nacht ein Programm gestartet werden, um aktuell richtig kategorisierte Daten zu exportieren und auf diesen mittels LIBLINEAR ein Modell zu trainieren. Außerdem muss ein Server implementiert werden, der mittels des berechneten Modells per API die vorgeschlagene Kategorie ausgibt. Der Prozess muss auf alle bisherigen nicht manuell kategorisierten Produkte angewandt werden.

Zusätzlich gibt es noch Klassifizierer, die genau für hierarchische Klassen entwickelt wurden. Dabei geht der Algorithmus Ebene für Ebene durch den Kategorienbaum und sollte somit präzisere Trefferquoten liefern als die Klassen flach zu betrachten.

Eine weitere Optimierungsmöglichkeit bestünde darin, die Produkt-Bilder mit in die Klassifizierung zu nehmen. Durch eine zusätzliche Klassifikation anhand des Bildes ist laut [KTRK11] eine bis zu 16% höhere Trefferquote möglich.

Neben den Ansätzen die Trefferquote bei den Klassifizierern zu verbessern, existiert noch die Möglichkeit, die Generierung der Features aus dem Text besser zu steuern. Statt nur einfache TF-IDF Maße auszurechnen, wäre es von Vorteil zum Beispiel Hersteller Wörter höher zu gewichten, Wort-Zahlen Kombinationen als ein Feature zu generieren oder Produkt-Titel Wörter höher zu werten. Dazu müsste man den Wekas StringToWordVector Filter durch ein eigenes Programm ersetzen bzw. die automatische Indexierung bereits bei der Generierung des Datensatzes für Weka vornehmen.

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Literaturverzeichnis

[AL08] Alpaydin, Ethem ; Linke, Simone: *Maschinelles Lernen*. München : Oldenbourg Verlag, 2008. – ISBN 978–3–486–58114–0

[BD06] Buddhinath, Gaya ; Derry, Damien: A simple enhancement to One Rule Classification. In: *Technique Report at* (2006)

[Bre01] Breiman, Leo: Random Forests. In: *Machine Learning* 45 (2001), Nr. 1, S. 5–32

[BYRN99] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern information retrieval*. Reno : ACM Press, 1999. – ISBN 978–0–201–39829–8. – http://www-nlp.stanford.edu/IR-book/

[Coh95] Cohen, William W.: Fast Effective Rule Induction. In: *Twelfth International Conference on Machine Learning*, Morgan Kaufmann, 1995, S. 115–123

[CT12] Cover, Thomas M. ; Thomas, Joy A.: *Elements of information theory*. John Wiley & Sons, 2012

[FCH+08] Fan, Rong-En ; Chang, Kai-Wei ; Hsieh, Cho-Jui ; Wang, Xiang-Rui ; Lin, Chih-Jen: LIBLINEAR: A library for large linear classification. In: *The Journal of Machine Learning Research* 9 (2008), S. 1871–1874

[FS97] Freund, Yoav ; Schapire, Robert E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Journal of computer and system sciences* 55 (1997), Nr. 1, S. 119–139

[HT98] Hastie, Trevor ; Tibshirani, Robert: Classification by Pairwise Coupling. In: Jordan, Michael I. (Hrsg.) ; Kearns, Michael J. (Hrsg.) ; Solla, Sara A. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 10, MIT Press, 1998

[JL95] John, George H. ; Langley, Pat: Estimating Continuous Distributions in Bayesian Classifiers. In: *Eleventh Conference on Uncertainty in Artificial Intelligence*. San Mateo : Morgan Kaufmann, 1995, S. 338–345

[KSBM01] Keerthi, S.S. ; Shevade, S.K. ; Bhattacharyya, C. ; Murthy, K.R.K.: Improvements to Platt's SMO Algorithm for SVM Classifier Design. In: *Neural Computation* 13 (2001), Nr. 3, S. 637–649

[KTRK11] Kannan, Anitha ; Talukdar, Partha P. ; Rasiwasia, Nikhil ; Ke, Qifa: Improving product classification using images. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on* IEEE, 2011, S. 310–319

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

978–1–139–47210–4

[Nil14] NILSSON, Nils J.: *Introduction to machine learning-an early draft of a proposed textbook*. Website, 2014. – Verfügbar online auf http://robotics.stanford.edu/~nilsson/mlbook.html; besucht am 05.05.2014

[Pla98] PLATT, J.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. Version: 1998. http://research.microsoft.com/~jplatt/smo.html. In: SCHOELKOPF, B. (Hrsg.) ; BURGES, C. (Hrsg.) ; SMOLA, A. (Hrsg.): *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998

[Pla99] PLATT, John C.: Fast training of support vector machines using sequential minimal optimization. (1999)

[Qui93] QUINLAN, Ross: *C4.5: Programs for Machine Learning*. San Mateo, CA : Morgan Kaufmann Publishers, 1993