Abschlussbericht der Projektgruppe 520

Autor1 Autor2 ...

7. November 2007

Inhaltsverzeichnis

1 Einführung **5**
- 1.1 Aufgabenstellung . . . 5
- 1.2 Methoden der Named Entity Recognition . . . 5
  - 1.2.1 Überblick . . . 5
  - 1.2.2 Hidden Markov Model . . . 9
  - 1.2.3 Maximum Entropy Markov Models . . . 19
  - 1.2.4 Conditional Random Fields . . . 19
- 1.3 Methoden der Indexierung und Information Retrieval . . . 19
  - 1.3.1 Indexierung . . . 19
  - 1.3.2 Suchmaschinen . . . 22
  - 1.3.3 Lernende Suchmaschinen . . . 25
- 1.4 Maschinelle Lernverfahren . . . 30
  - 1.4.1 Klassifikation . . . 30
  - 1.4.2 Clustering . . . 38
- 1.5 Anwendungsbereich . . . 40
- 1.6 Corpus-Erstellung und Beispielmengen . . . 40
- 1.7 Systementwurf . . . 40

2 Informationsextraktion **41**

3 Fragebeantwortung **43**

4 Evaluation **45**

Kapitel 1

Einführung

1.1 Aufgabenstellung

1.2 Methoden der Named Entity Recognition

1.2.1 Überblick

Definition und Zielsetzung

Zielsetzung der Named Entity Recognition ist es, bestimmte Bestandteile eines natürlich-sprachlich verfassten Text, also Zeitungsartikel, Dossiers oder auch E-Mail, zu erkennen und einer vorgegebenen Kategorie zuzuordnen. Diese Bestandteile nennt man auch **Named Entities**. Die Named Entities dienen in der Regel dazu wichtige Fragen, die an einen natürlich-sprachlichen Text gestellt werden, zu beantworten. Was? Wann? Wo? und Wer? sind dabei typische Fragen, die mit der Named Entity Recognition geklärt werden können.
Die Besonderheit dieser Named Entities ist ihre Einmaligkeit, denn üblicherweise handelt es sich dabei um eine Person, Organisation oder einen Ort. Angaben also, die in der Regel definit sind und nur einmal vorkommen. Beispielsweise könnte dem Begriff *Angela Merkel* eindeutig die Klasse Person zugewiesen werden - es handelt sich also um ein Named Entity oder auch Eigenname, während der Begriff *Bundeskanzler/in* nicht auf eine bestimmte Person hinweist, somit also nicht klassifiziert wird. Wichtig ist also, dass nur relevante Informationen klassifiziert werden, wobei sich die semantische Relevanz auf die Domäne des Texts bezieht. Die Relevanz kann durch die Auswahl passender Trainingsdaten sowie den Tags bestimmt werden. Neben den oben aufgeführten Kategorien Person, Organisation und Ort gehören auch Zeitangaben sowie quantitative Aussagen zu den möglichen Named Entities.
Hier nun ein kurzes Beispiel für eine mögliche Ein- und Ausgabe eines NER-Systems:

Eingabe: Auch die widersprüchlichen Angaben darüber, wie viel Geld Bohlen tatsächlich am 11. Dezember 2006 gestohlen wurde, wollte das Landgericht Bochum klären.

Ausgabe: Auch die widersprüchlichen Angaben darüber, wie viel Geld **<Person>Bohlen</Person>** tatsächlich am **<Datum> 11. Dezember 2006 </Datum>** gestohlen wurde, wollte das **<Organisation> Landgericht Bochum </Organisation>** klären.

Geschichte der NER

Named Entity Recognition wurde durch die von der DARPA instituierten Message Understanding Conferences (kurz: MUC) bekannt. Die Message Understanding Conferences fanden erstmals 1987 statt und hatten die Entwicklung besserer Information Extraction-Methoden zum Ziel. Dabei gibt es eine Reihe von Teams, die alle an einem genau definierten Ziel, unter Vorgabe der zu untersuchunden Texte, arbeiten. Die einzelnen Ergebnisse der Gruppen werden an den Konferenztagen vorgestellt und ermöglichen so einen umfassenden Blick über den aktuellen Stand der Technik.
In diesem Zusammenhang wurde Named Entity Recognition / Koreferenz erstmals 1995 in der sechsten MUC als Ziel definiert. Das Forschungsgebiet ist in der Informatik also relativ neu.

Evaluation

Um gute Programme zu entwickeln, benötigt man ein Scoring-verfahren um die Qualität des Programms bemessen zu können. In der Named Entity Recognition haben sich dabei drei immer wieder verwendete Evaluationsmaße etabliert. Alle Werte gehen davon aus, dass wir den gesamten Korpus einer Datei oder eines Texts nach Named Entities durchsuchen.

Precision Unter Precision, oder zu deutsch Präzision, versteht man die Anzahl der korrekt klassifizierten Named Entities im Verhältnis zu der Anzahl der vom Recognizer gefundenen Named Entities.

$$Precision = \frac{Anzahl korrekt klassifizierter NEs}{Anzahl NEs gefunden} \tag{1.1}$$

Recall Recall ist die Ausbeute an Named Entities aus dem gesamten Dokument. Hier wird also das Verhältnis der Anzahl an korrekt klassifizierten Named Entities zu den insgesamt im Text vorhandenen Named Entities betrachtet.

$$Recall = \frac{Anzahl korrekt klassifizierter NEs}{Anzahl vorhandener NEs im Korpus} \tag{1.2}$$

F-Measure Um beide Größen, also Precision und Recall, zusammen betrachten zu können, bildet man den ungewichteten harmonischen Mittelwert beider Werte und erhält den so genannten F-Measure [13].

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{1.3}$$

Interne / Externe Evidenz

Jedes Merkmal zur Klassifikation eines Named Entity kann ganz grob in Interne und Externe Evidenz aufgeteilt werden. So kann jedes Analyseverfahren in [1.2.1] eine der beiden Gruppen aufgeteilt werden.

Interne Evidenz Verfahren der Internen Evidenz ziehen ihr Wissen nur aus dem betrachteten Wort. Dies kann ein Eintrag in einem Lexikon (zum gesuchten Wort) sein [1.2.1], bestimmte Bestandteile eines Wortes oder auch die Großschreibung einzelner Buchstaben. Wenn wir beispielsweise das Wort DARPA betrachten, ist es wahrscheinlich, dass es sich hierbei um eine Organisation handelt, da alle Buchstaben groß geschrieben sind, folglich es sichum eine Abkürzung handeln könnte.

Externe Evidenz Viele Verfahren der Internen Evidenz gelangen rasch an ihre Grenzen. So unterscheidet sich rein syntaktisch eine Bank mit der Bedeutung einer Finanzeinrichtung nicht mit der einer (Sitz-) Bank. Auch ein Lexikon würde an dieser Stelle nur viele Möglichkeiten anzeigen, aber kein Aufschluss darüber geben, worum es sich nun wirklich handelt.
Hier muss man weitergehen und den Kontext des Wortes betrachten. Der Kontext kann einerseits der umschlossene Satz, aber auch ein ganzer Textabsatz darstellen. So ist der Kontext *Die französische Stadt...* ein eindeutiger Hinweis auf ein Named Entity der Kategorie *Ort.*Zum Kontext gehören aber auch die vorher bestimmten Named Entities, die oftmals ebenfalls ein hilfreiches Indiz für eine mögliche Einordnung liefern.

Mögliche Analyseverfahren

Tokenisierung Bevor man einen Text bearbeiten und maschinell auswerten kann, muss man ihn in seine einzelnen Bestandteile, den Token, zerlegen. Diese Aufsplittung gestaltet sich bei den meisten Sprachen, beispielsweise englisch, deutsch oder spanisch, relativ einfach. Man sucht nach Leerzeichen, die üblicherweise jedes Wort abgrenzen und erhält damit einen einzelnen Token. Wichtig bei der Tokenisierung ist allerdings auch, dass dies nicht in allen Sprachen so einfach sein muss.
Im nächsten Schritt kann man anhand von Satzzeichen ('.', ',', '¿) die Struktur des Texts erfassen und bekommt damit wichtige Informationen wenn es darum geht den Kontext eines Worts zu untersuchen. Ausruf- oder Fragezeichen kann man hierbei als deutlichen Hinweis für ein Satzende ansehen. Bei einem '.'kann man sich dagegen nicht immer sicher sein, dass es sich um das Ende eines Satzes handelt. Hier ist beispielsweise auch eine Abkürzung möglich (bzw. , e.V.,usw.) und man muss anhand des Kontexts bzw. des syntaktischen Aufbaus das Satzende bestimmen.

Morphologische Analyse Die morphologische Analyse bedient sich sprachlicher Mittel um die im ersten Schritt erfassten Token für die weitere Bearbeitung zu vereinfachen. Ein wichtiger Bestandteil der morphologischen Analyse ist das **Stemming-Verfahren**. Hierbei werden Verben in ihre Stammform überführt

um die Anzahl der Regeln, die in den nachfolgenden Schritten zur Erkennung der Kategorie dienen, klein zu halten. So wird aus *'schläft' 'schlafen'* und aus *'sprach' 'sprechen'*. Auch die Anzahl der Lexikoneinträge kann mit diesem Verfahren reduziert werden.
Ganz ähnlich kann man sich die Motivation für die Präfix- und Suffix-Erkennung vorstellen. Auch hier geht es darum komplex zusammengesetzte Wörter zu vereinfachen und auf ihr Lemma zurückzuführen.
Wichtig zu erwähnen ist, dass dieser Schritt bei manchen Sprachen sehr wichtig sein kann. Im Deutschen gibt es eine Vielzahl von morphologischen Veränderungen eines Wortes. Viele verschiedene Zusammensetzungen eines Wortes sind denkbar und müssen hier berücksichtigt werden. Gleiches gilt beispielsweise auch in der französischen Sprache. Im Gegensatz dazu kann man bei der Betrachtung eines englischen Textes einen Großteil dieser Verfahren wegfallen lassen und allenfalls noch ein Stemming von Verben durchführen. Die maschinelle Erfassung der Semantik eines Textes kann also in den verschiedenen Sprachen deutlich variieren.

Lexikalische Analyse Die Lexikalische Analyse in der Named Entity Recognition ist ein einfacher LookUp in einem vorbereiteten Lexikon. In den meisten Fällen kann an dieser Stelle schon eine eindeutige für das Named Entity gefunden werden. So wird man unter dem Stichwort *Angela Merkel* sicherlich nur eine mögliche Kategorie-Zuordnung zulassen, nämlich *Person*.
Da es sich hierbei allerdings um ein Verfahren der Internen Evidenz (1.2.1) handelt, sprich es wird nur die Struktur des Wortes / Token betrachtet und nicht der Kontext, können Mehrdeutigkeiten auftreten. Diese Mehrdeutigkeiten können nur aufgehoben werden, wenn man sich den kompletten Satz, bzw. den ganzen Abschnitt ansieht und nach möglichen Relationen zur gesuchten Entität Ausschau hält. Dies wird in der Syntaktischen Analyse gemacht.
Hier einige Beispiele für Mehrdeutigkeiten:

Wort	Mögliche Deutung
Essen	Stadt in NRW oder Mahlzeit?
Bank	Finanzeinrichtung oder Sitzmöbel?
Buchen	Baum oder Imperativ von *buchen*.Bsp.: **Buchen** Sie mir einen Flug!
Hamburger	Burger oder Einwohner von Hamburg?

Tabelle 1.1: Beispiele von Mehrdeutigkeiten

Syntaktische Analyse Um Mehrdeutigkeiten in der Struktur der Wörter aufzuheben, kommt man nicht umher den Kontext eines Wortes zu betrachten. Kleinräumig ist dies ein Satz, im größeren Maßstab kann man hier aber auch einen Textabschnitt betrachten. Bei der Betrachtung des Kontexts geht es darum den Satz in syntaktische Bestandteile zu zerlegen. Es geht dabei also um die Erkennung von Nomen, Verben, Präpositionen, usw.. Auch bereits erkannte und klassifizierte Named Entities gehören zum Kontext und können hilfreich sein, beispielsweise wenn es darum geht zwei gleiche Referenzen auf ein Objekt aufzulösen. So steht die Abkürzung *IBM* für *International Business Machine*. Beides sind Named Entities, die aber die gleiche Firma beschreiben. Anhand der Abkürzung könnte man dies erkennnen.
Für die syntaktische Analyse eines Satzes werden in der Named Entity Recognition **Part-of-Speech Tagger** eingesetzt. Diese weisen jedem Wort eines Satzes eine Wortklasse zu. Anhand von Kontextinformationen können hier Mehrdeutigkeiten aufgehoben werden, wobei auch unbekannten Wortphrasen eine Wortklasse zugeteilt wird. TnT ist ein relativ bekannter POS-Tagger und klassifiziert englische Wörter mit bis zu 86 Prozent und deutsche mit 89 Prozent Wahrscheinlichkeit zur richtigen Wortgattung.
Eine Alternative zum POS-Tagger wäre ein **Full-Parsing**, wo die Analyse einer Satzkonstruktion über einen Parsebaum geschieht, der an kontextfreie Grammatiken angelehnt ist und Wortgattungen aufgrund von Position und Vorkommen im Satz ableitet. Allerdings ist diese Verfahren sehr fehlerbehaftet und benötigt viel Rechenzeit, weswegen man es in der Regel nicht anwendet [1].

Domänenspezifische Analyse In der Domänenspezifischen Analyse geht es darum die Klassifizierung durch Einbezug des Text-Themas - daher Domänenspezifisch - zu verbessern. Hierbei kommt insbesondere die Koreferenz-Auflösung zum Einsatz. Die Aufgabe der **Koreferenz-Auflösung** ist die Erkennung von gleichen Referenzen innerhalb eines Texts. Falls also am Anfang eines Texts von G.W. Bush die Reder ist und in späteren Abschnitten nur von Er die Rede ist, muss erkannt werden, dass es sich hierbei um die anfangs genannte Person handelt. Auch bei Organisationen/Firmen ist die Erkennung dieser Personalpronomen wichtig um den Inhalt des Texts verfolgen zu können. Auch temporale Referenzen müssen allerdings aufgelöst werden. So bedeutet die Phrase ‘um Viertel vor Vier ‘genausoviel wie 15:45 Uhr.

Beim **Merging** geht es hingegen darum gegebene Named Entities zu verschmelzen, falls es sich dabei um ein und dasselbe Objekt handelt. Hier einige Merging-Beispiele:

Abkürzung	Referenz
IBM	International Business Machine
Deutsche Bahn AG	Die Bahn entlässt Mitarbeiter
USA	United States of America
Hamburger	Burger oder Einwohner von Hamburg?

Tabelle 1.2: Beispiele für Merging-Operationen

Systemarchitekturen

Listenbasierte Systeme Beim Entwurf einer Systemarchitektur für ein NER-System kann man sich ganz einfach eine riesige Liste mit allen Named Entities vorstellen, die es gibt. In dieser Liste müssten dann aber auch immer alle neuen Wortschöpfungen ergänzt, sowie morpholgische Varianten eines Wortes abgespeichert werden. (aus [4])
Während diese beiden Punkte noch relativ gut realisierbar sind, führen die möglichen Mehrdeutigkeiten dazu, dass ein Listenbasiertes Verfahren keine gute Option für die Named Entity Recognition darstellt. Mann kann zwar nachschlagen, was Essen alles bedeuten kann, aber ohne Betrachtung des Kontexts wird es nicht möglich sein, die passende Kategorie der Entität zu finden. Rein Listenbasierte Verfahren scheiden also aufgrund der zu hohen Fehlerrate aus.

Regelbasierte Systeme Bei Regelbasierten Systemen definiert man anhand der Merkmale der einzelnen Token Regeln, um das Einsortieren in Kategorien zu ermöglichen. Dabei nutzt man morpholgisches, lexikalisches, syntaktisches und domänenspezifisches Wissen aus, also alle Verfahren, die zur Bestimmung von Named Entities in Frage kommen. Hat man eine Reihe von Regeln entwickelt kann man daraus eine Grammatik (kontextfrei) entwickeln, die in Texten nach Named Entities sucht.
Im Gegensatz zu lernbasierten Verfahren benötigt man hier allerdings spezielle Linguisten für die Entwicklung der einzelnen Regeln. Diese Regeln werden dann an relativ kleinen Trainings-Datensätzen getestet und falls erforderlich korrigiert und wieder getestet. Nach einigen Iterationen hat man so eine qualitativ gute Regelmenge zur Erkennung von Named Entities zusammen. Insgesamt ist die Entwicklung von Regelbasierten Verfahren zeitaufwändiger, da man hier sehr umfassende Grammatiken entwickeln muss. Und auch hinsichtlich der Flexibilität zeigen sich Regelbasierte Systeme eher nachteilig im Vergleich zu Lernbasierten Verfahren. Hier kann man die Grammatik in der Regel nicht schnell erweitern oder gar einige Teile verändern, da viele Regeln aufeinander aufbauen und transitiv abhängig sind. Auch die Anpassung an eine neue Domäne ist damit wesentlich schwieriger. (siehe [6])

Nachfolgend einige Beispiele wie solche Regeln aussehen können:

- Aufeinanderfolgende Phrasen der Form <Wort><Wort> GmbH deuten mit hoher Wahrscheinlichkeit auf eine Firma / Organisation hin
- Großgeschriebene Worte können Hinweise auf eine Firma oder Organisation sein: NASA, ADAC, UNICEF
- ‘denken‘ist, unabhängig vom Tempus der Verbform, in der dritten Person immer ein starker Hinweis für ein menschliches Subjekt
- Vorkommen von ‘-burg‘, ‘-dorf‘, ‘-stadt‘deuten auf eine Ortsangabe aus dem deutschsprachigen Raum hin

Lernende / Automatische Systeme Lernende oder Automatische Systeme nutzen statistische Verfahren oder Methoden des Maschinellen Lernens um in einem Textkorpus nach Named Entities zu suchen. Im Gegensatz zu Regelbasierten Verfahren werden dabei qualitativ und quantitativ hohe Anforderungen an Trainingstexte gestellt. Douglas E. Appelt und David J. Israel beschreiben in ihrem Paper [1], dass man für jede Verdopplung der Trainingsdatensätze den F-Measure um 1,5 Punkte steigern kann. Dabei ist die Auswahl der Trainingsdatensätze entscheiden für das spätere Verhalten des Recognizers und kann mitunter schwieriger sein als das Entwerfen von Regeln für ein Regelbasiertes System.
Zu den wichtigsten Verfahren des Maschinellen Lernens gehören Hidden Markov Models 1.2.2, Maximum

Entropy Markov Models 1.2.3, Conditional Random Fields 1.2.4 und Support Vextor Machines 1.4.1. Jedes Verfahren wird mit den möglichen Erweiterungen weiter unten beschrieben.
Eine Gefahr bei Maschinellen Lernverfahren ist das so genannte **Overfitting**. Dabei wird dergleiche Trainingstext immer und immer wieder gelernt, so dass eine Überanpassung des Systems an den Datensatz vorkommen kann. Wird nun bei der Erkennung ein thematisch anderer Datensatz verwendet, ist eine hohe Fehlerrate bei der Klassifikation von Named Entities wahrscheinlich.

Vor- und Nachteile der Systemarchitekturen

Regelbasierte Systeme benötigen aufgrund der einfachen Erstellung eines Parsebaums relativ wenig Zeit zur Auswertung, wobei die Auswertungsergebnisse auf gleicher Höhe oder etwas besser im Vergleich zu Lernbasierten Systemen sind. So ergab die Auswertung eines Wall Street Journals in der siebten Message Understanding Conference einen Vorteil der Regelbasierten Systeme von 3,3 Prozentpunkten (93,7% vs. 90,4% gemessen an der Zahl der korrekt klassifizierten NE, [6]).
Lernende Systeme sind hingegen wesentlich flexibler, wenn es darum geht, ein System zur Laufzeit zu verändern. Während wir bei Regelbasierten Systemen viele Regeln einzeln anpassen müssen, genügt es die Parameter zu verändern und das System mit neuen Trainingstexten zu füttern. Diese Flexibilität zeigt sich auch hinsichtlich der Anpassung an verschiedene Sprachen.

1.2.2 Hidden Markov Model

Vorwort

Namensgeber des Modells ist sein Entwickler Andrei Andrejewitsch Markov (1856 - 1922), der wesentliche Beiträge zur Wahrscheinlichkeitstheorie und Analysis beisteuerte. Er berechnete 1913 die Buchstabensequenzen in russischer Literatur um die Notwendigkeit der Unabhängigkeit für das Gesetz der grossen Zahlen nachzuweisen. Die Berechnungen konnten zudem als Aussage über die Wohlgeformtheit der Orthographie von Buchstabenketten interpretiert werden. Aus diesem Ansatz entwickelte sich ein allgemeines statistisches Werkzeug, der sogenannte stochastische Markov-Prozess, basiert auf dieser Methode findet heute sich eine Anwendung für unsere Projektgruppe sogenannter Hidden Markov-Models bzw. in der NER [1].

Markov-Prozess

Der Markov-Prozess wird allgemein synonym zur Markov-Kette verwendet, allgemeiner da sich die Verteilungsfunktion zu einem beliebigen, auch nicht- diskretem Zeitpunkt bestimmen lässt. In der Computertechnik kommen die spezielleren Markov-Ketten zum Einsatz.

Markov-kette

Eine Markow-Kette ist eine spezielle Klasse von stochastischen Prozessen. Man unterscheidet eine Markow-Kette in diskreter und in stetiger Zeit. Markow-Ketten in stetiger Zeit werden meistens als Markow-Prozess bezeichnet. Das System springt bei jedem Zeitschritt von einem Zustand in den nächsten, dem zeitlichen Fortschreiten liegt eine Übergangswahrscheinlichkeit zugrunde. Das Spezielle einer Markow-Kette ist die Eigenschaft, dass durch Kenntnis einer begrenzten Vorgeschichte ebensogute Prognosen über die zukünftige Entwicklung möglich sind wie bei Kenntnis der gesamten Vorgeschichte des Prozesses. Im Falle einer Markow-Kette erster Ordnung heißt das: Die Zukunft des Systems hängt nur von der Gegenwart (dem aktuellen Zustand) und nicht von der Vergangenheit ab. Um diese Eigenschaft anschaulich zu beschreiben sagt man auch, dass eine Markovkette „Gedächtnislos“ ist. Ausserdem besagt die Ordnungszahl einer Markov-Kette, von wie viel vorherigen Zuständen der aktuelle Zustand abhängt. Bei der Markov-Ketten 1. Ordnung hängt der aktuelle Zustand und seine Ausgabe nur vom seinem vorherigen Zustand ab, mathematisch ausgedruckt:

$$\begin{aligned} P(q_t &= S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots) \\ &= P(q_t = S_j | q_{t-1} = S_i) \end{aligned}$$

Um Markov-Kette zu modellieren, müssen sie aus die folgenden Elemente bestehen:

- die Menge der Zustände $S = \{s_1, s_2, \ldots, s_N\}$

[1] Named Entity Recognition

Abbildung 1.1: Wettervorhersage mit dem Markov Modell

- a_{ij} ist die Wahrscheinlichkeit, dass das System in den Zustand s_j übergeht unter der Vorraussetzung, dass das System sich im Zustand s_i befindet, wobei die Summe aller Übergangswahrscheinlichkeiten in einem Zustand bzw. einer Zeile der Matrix zusammen 1 ergeben müssen.

$$a_{ij} = P(q_{t+1} = s_j | q_t = s_i), 1 \leq i, j \leq N \tag{1.4}$$

$$a_{ij} \geq 0, \sum_{i=1}^{N} a_{ij} = 1 \tag{1.5}$$

- Anfangswahrscheinlichkeitenvektor $pi = \{\pi_i\}$, der den Anfangszustand des Systems beschreibt.

$$\pi_i = P(q_1 = s_i), 1 \leq i \leq N \tag{1.6}$$

Ein Markov-Modell ist eine Datenstruktur, die die nötigen Werte für die Steuerung einer Markov-Kette bereitstellt. hier beschreibt das ursprüngliche, einfache Modell zu einem Modell zur Vorhersage auf Grundlage indirekter Evidenz (Hidden Markov-Modell).

Hier benutze ich das populäre Beispiel vom Markov-Modell, nämlich ist das System der Wettervorhersage, welches einfach mit Zwei Zustände aufgebaut ist: Regen und Sonne.
Übergangswahrscheinlichkeiten:

$$P(Regen|Regen) = 0.3, P(Sonne|Regen) = 0.7$$

$$P(Regen|Sonne) = 0.2, P(Sonne|Sonne) = 0.8$$

$$A = \{a_{ij}\} = \begin{pmatrix} 0.3 & 0.2 \\ 0.7 & 0.8 \end{pmatrix}$$

Anfangswahrscheinlichkeiten:

$$P(Regen) = 0.4, P(Sonne) = 0.6$$

Nehmen wir an, dass wir eine Beobachtungsfolge der Zustände in unserem Beispiel errechnen möchten: {Sonne, Sonne, Regen, Regen}.

$$\begin{aligned} P(Sonne, Regen) &= P(Sonne)P(Sonne|Sonne)P(Regen|Sonne)P(Regen|Regen) \\ &= 0.6 * 0.8 * 0.2 * 0.3 \\ &= 0.0288 \end{aligned}$$

Hidden Markov-Model

Heute finden wir HMM-Technik überall vor, wo (Zeit)seriendaten zu modellieren, interpretieren, segmentieren sind und wo einen Erwerb umfangreicher Trainingsdatensammlungen mit passendem Aufwand zu vollziehen ist. Sie machen auf die vielfalt von technischen Anwendungen, einige bekannten Anwendungsgebieten sind:

- Gen-Vorhersage
- Spracherkennung und Mustererkennung
- Signal-Verarbeitung
- Neurobiologie
- ...

mit Hilfe von Hidden Markov-Modellen kann man nach bestimmten Sequenzen oder auch Mustern in Daten suchen. Bei allen diesen Anwendungsmöglichkeiten werden bestimmte Muster in grossen Datenmengen gesucht, die Mustersuche in Datenbanken wird als Data-Mining bezeichnet.

Das Hidden Markov-Modell, ist aus allgemeinem Markov-Modell ein erweitertes stochastisches Modell, das sich durch zwei Zufallsprozesse beschreiben lässt. Der erste Zufallsprozess entspricht dabei einer Markov-Kette, die durch Zustände und Übergangswahrscheinlichkeiten gekennzeichnet ist. Die Zustände der Kette sind von außen jedoch nicht direkt sichtbar (sie sind verborgen). Stattdessen erzeugt ein zweiter Zufallsprozess zu jedem Zeitpunkt beobachtbare Ausgangssymbole gemäß einer zustandsabhängigen Wahrscheinlichkeitsverteilung. Die Aufgabe besteht häufig darin, aus der Sequenz der Ausgabesymbole auf die Sequenz der verborgenen Zustände zu schließen.

Definition [9] ein Hidden Markov-Modell wird formal als Fünftupel $\lambda = (N, M, A, B, \pi)$ definiert:

- N ist die Menge aller Zustände $S = \{s_1, s_2, \ldots, s_N\}$ im HMM die Zustandsvariable $Q = q_1 q_2 \ldots q_T$ annehmen kann.
- M ist Merkmalsraum (oft als Ausgabealphabet bezeichnet), also der Definitionsbereich von b_i. Dieser kann diskret, oder kontinuierlich sein. Je nach dem ist b_i eine Wahrscheinlichkeit oder eine Dichte. Wahrscheinlichkeit, dass q_i der Startzustand ist.
- $A = \{a_{ij}\}$ Zustandsübergangsmatrix, wobei a_{ij} die Wahrscheinlichkeit angibt, dass nach Zustand q_i in Zustand q_j gewechselt wird.
- $B = \{b_1, \ldots, b_n\}$ Menge der Emissionswahrscheinlichkeitsverteilungen bzw. Dichten
- $b_i(k)$ Wahrscheinlichkeit im Zustand q_i die Beobachtung k zu machen
- $\pi = \{\pi_i\}$ Anfangswahrscheinlichkeitsverteilung mit

$$\pi_i = P(q_1 = s_i), 1 \leq i \leq N \tag{1.7}$$

Gegeben seien die passenden Werte von N, von M, von A, von B und von π das HMM kann als Generator verwendet werden, um eine Beobachtung Reihenfolge anzugeben, in der jede Beobachtung O_t eins der Symbole von V ist und T die Zahl Beobachtungen in der Folge wie folgen ist. Das oben genannte Verfahren kann verwendet werden als Generator von Beobachtungen und als Modell für, wie eine gegebene Beobachtung Reihenfolge durch ein passendes HMM erzeugt wurde. Genauer diese Schrittfolge[9] sieht man direkt unten ein.

1. wähle Anfangszustand $q_i = s_i$ entsprechend Anfangszustandverteilung π
2. Setze $t = 1$
3. Wähle $O_t = v_k$ entsprechend Wahrscheinlichkeitsverteilung der Beobachtungssymbole im Zustand s_i, d.h. $b_i(k)$
4. Wechsle in den nächsten Zustand $q_{t+1} = s_i$ entsprechend übergangswahrscheinlichkeitsverteilung a_{ij} für Zustand s_i

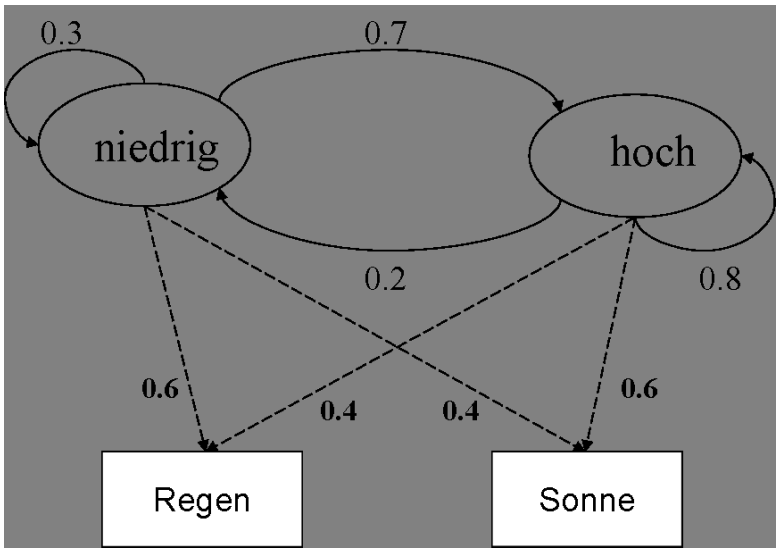


Abbildung 1.2: Wettervorhersage mit dem Hidden Markov-Modell

5. setze $t = t + 1$, wiederhole Schritt 3, falls $t < T$, sonst endet dies Verfahren

In der vordere Vorstellung haben wir schon gesehen, wie das System der Wettervorhersage mit Hilfe des allgemeinen Markov-Modells aussieht. und jetzt kucken wir nochmal das erweiterte System, welches aus dem Markov-Modell evolviert ist, an.

Zwei Zustände vom Luftdruck:

niedrig oder hoch

Zwei Beobachtungen:

Regen und Sonne

Übergangswahrscheinlichkeiten:

$$P(niedrig|niedrig) = 0.3, P(hoch|niedrig) = 0.7$$

$$P(niedrig|hoch) = 0.2, P(hoch|hoch) = 0.8$$

$$A = a_{ij} = \begin{pmatrix} 0.3 & 0.2 \\ 0.7 & 0.8 \end{pmatrix}$$

Ausgabewahrscheinlichkeiten:

$$P(Regen|niedrig) = 0.6, P(Sonne|niedrig) = 0.4$$

$$P(Regen|hoch) = 0.4, P(Sonne|hoch) = 0.3$$

Anfangswahrscheinlichkeiten:

$$P(niedrig) = 0.4, P(hoch) = 0.6$$

hier als Beispiel können wir versuchen, alle mögliche Reihenfolge der versteckten Zustände anzunehmen:

$$\begin{aligned} P(Sonne, Regen) &= P(Sonne, Regen, niedrig, niedrig) \\ &+ P(Sonne, Regen, niedrig, hoch) \\ &+ P(Sonne, Regen, hoch, niedrig) \\ &+ P(Sonne, Regen, hoch, hoch) \end{aligned}$$

anschaulich sollte ein Gedanke nach unserer Probe daher erzeugt werden. Die Berechnung von $P(O|\lambda)$ gemäß originaler Definiton kann ziemlich groß sein, man betracht im diesem Fall den Rechenaufwand auf $O(N^T)$. Gibt es Methode, um diesen Schlechte Fall zu verbessern? In den folgenden Abschnitten werden wir konkrete darauf eingehen.

Drei klassische Problemstellungen bei den Anwendungen

Es gibt drei grundlegende Probleme, die gelöst werden müssen, damit das Hidden Markov-Modell in der Real-Welt bei verschiedenen Anwendungen nützlich ist. hier finde ich noch ein sinnvolles Online-Tutorium [2] für alle Interessanten vor.

1. Gegeben sei die Beobachtungsfolge $O = O_1 O_2 \ldots O_T$ und ein Modell $\lambda = (A, B, \pi)$, wie berrechnet man die Wahrscheinlichkeit $P(O|\lambda)$?

2. Wie bestimmt man eine Zustandsfolge $Q = q_1 q_2 \ldots q_T$, die eine gegebene Beobachtungsfolge $O = O_1 O_2 \ldots O_T$ zu gegebenem Modell $\lambda = (A, B, \pi)$ am besten erklärt?

3. Wie paßt man die Modellparameter $\lambda = (A, B, \pi)$ an, wenn nur die Beobachtungsfolge bekanntgegeben ist, um die Wahrscheinlichkeit $P(O|\lambda)$ zu maximieren?

Die Wahrscheinlichkeit einer Reihenfolge von Übergängen herauszufinden ist eine nützliche Sache, da wir bald sehen, aber es ist auf keinen Fall die einzige Sache, die wir tun möchten. Um die Wahrscheinlichkeit einer Reihenfolge von Beobachtungen zu finden,wird gekennzeichnet als ***Evaluation*** und kann in der Vorwärts- und Rückwärts- Richtung gerechnet werden (d.h. vorwärts von der ersten Beobachtung arbeiten oder von der letzten Beobachtung zurück zu arbeiten).

Nachdem das oben genannte Problem gelöst worden war, im allgemeinen können wir noch nicht absichern, welche Reihenfolge der Zustände für eine gegebene Reihenfolge von Beobachtungen vorgenommen wird, Im allgemeinen ist das beste, dass wir tun können, die wahrscheinlichste Zustandflugbahn für eine gegebene Reihenfolge von Beobachtungen zu schätzen. Dieses Problem wird häufig als ***Decodieren*** gekennzeichnet.

Eine andere allgemeine Anforderung ist, die Wahrscheinlichkeiten zu erlernen, die mit Übergängen im System verbunden sind, durch eine Repräsentativtraining seine passende Reihenfolge gegeben werden anstatt seine Übergangswahrscheinlichkeiten direkt gegeben werden. Die Idee ist, das Raum von Übergangswahrscheinlichkeiten zu finden, der die Wahrscheinlichkeit der Training-Reihenfolge maximiert. Dieses ist das ***Lernproblem***.

Forward- und Backward-Algorithmus

Die Wahrscheinlichkeit der Beobachtungen O für eine spezifische Zustandreihenfolge Q ist:

$$\begin{aligned} P(O|Q,\lambda) &= \prod_{t=1}^{T} P(O_t|Q_t,\lambda) \\ &= b_{q_1}(O_1)(\times)b_{q_2}(O_2)\ldots b_{q_T}(O_T) \end{aligned} \tag{1.8}$$

und die Wahrscheinlichkeit der Zustandreihenfolge ist:

$$P(O|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \ldots a_{q_{T-1} q_T} \tag{1.9}$$

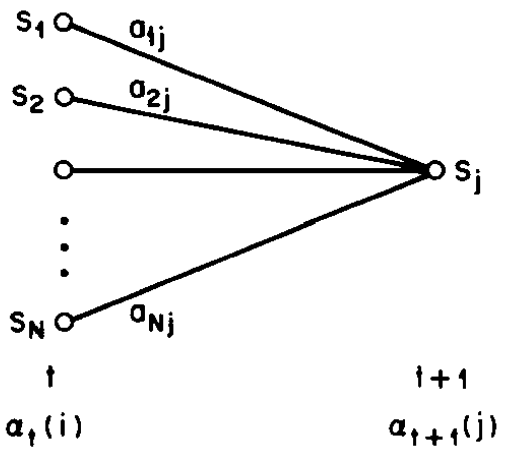


Abbildung 1.3: Forward-Rekursion

die Wahrscheinlichkeit einer Beobachtungssequenz bei vorgegebenem Modell ist die Wahrscheinlichkeit für das Auftreten einer Beobachtungssequenz bei allen moglichen Hintergrundsequenzen, die sich zur 1.8 Gleichung expandieren lässt.

$$P(O|\lambda) = \sum_{alleQ} P(O|Q,\lambda)P(Q|\lambda) \tag{1.10}$$

$$= \sum_{q_1,q_2,\ldots,q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \ldots a_{q_{T-1} q_T} b_{q_T}(O_T) \tag{1.11}$$

also die Forward-Variable wird so definiert:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = s_i|\lambda) \tag{1.12}$$

Forward Rekursion

1) Initialisierung:

$$\alpha_1(i) = P(O_1, q_1 = s_i|\lambda) = \pi_i b_i(O_1), 1 \leq i \leq N \tag{1.13}$$

Die Gesamtwahrscheinlichkeit, Zustand s_i im Zeitpunkt 1 zu erreichen, ist die Anfangswahrscheinlichkeit für s_i multipliziert mit der Wahrscheinlichkeit, dass O_1 ausgegeben wird.

2) Induktion:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}), 1 \leq t \leq T-1, 1 \leq j \leq N \tag{1.14}$$

Die Gesamtwahrscheinlichkeit, Zustand sj im Zeitpunkt t+1 zu erreichen, ist die Summe der Wahrscheinlichkeiten von einem beliebigen Zustand s_i in Zustand s_j zu kommen unter Ausgabe des nächsten Symbols O_{t+1}.

3) Terminierung:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{1.15}$$

Die Gesamtwahrscheinlichkeit der Realisierung ist die Summe der Wahscheinlichkeiten, zum Zeitpunkt T in einen beliebigen Zustand zu kommen. Daher ist der Rechenaufwand auf N^2T reduzierbar. also wir können die Backward-Rekursion analog machen. Die Backward Rekursion wird in der Lösung zum Problem 3 verwendet und ist nicht für das Problem 1 notwendig.

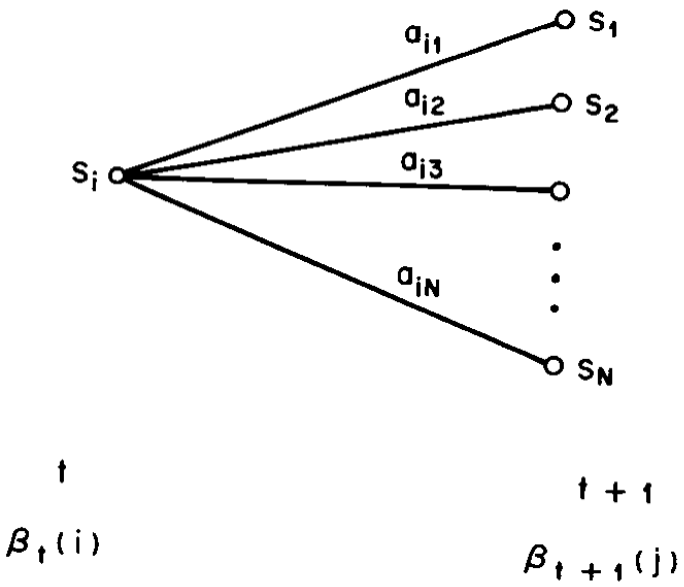


Abbildung 1.4: Backward-Rekursion

Backward Rekursion
Wir definieren zuerst die Backward-Variable:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T | q_t = s_i, \lambda) \tag{1.16}$$

1) Initialisierung:

$$\beta_T(i) = 1, 1 \leq i \leq N \tag{1.17}$$

2) Induktion:

$$\beta_t(j) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_i(O_{t+1}), 1 \leq i \leq N, t = T-1, T-2, \ldots, 1 \tag{1.18}$$

3) Terminierung:

$$P(O_1, O_2, \ldots, O_T) = \sum_{i=1}^{N} \beta_1(i) \pi_i b_i(O_1) \tag{1.19}$$

Viterbi-Algorithmus

Die Frage welche Hintergrundsequenz optimal bei gegebener Beobachtungssequenz und Modell ist, beschäftigt sich damit, die korrekte Zustandsabfolge zu finden. Zum zweiten Problem ist mit Hilfe des Viterbi-Algorithmus lösbar, wobei der $P(Q, O|\lambda)$ maximiert.

Der Viterbi-Algorithmus ist eine Methode aus dem Gebiet der Dynamischen Programmierung. Die Idee war, dass $\delta_t(i)$ die hochste Wahrscheinlichkeit durch einen Suchpfad zum Zeitpunkt n beschreibt. Dabei werden die ersten n Beobachtungen gezahlt und $\delta_t(i)$ landet schliesslich im Zustand s_i.

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_t = i, O_1, O_2, \ldots, O_t | \lambda \tag{1.20}$$

Durch Induktion hat man die Beziehung $\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_i(O_{t+1})$, Um die Zustandsfolge wieder zu gewinnen, speichert man auch gleichzeitig das Argument, das die $\delta_{t+1}(j)$ für jedes t und j jeweils maximiert, in ein Feld $\Psi_t(j)$

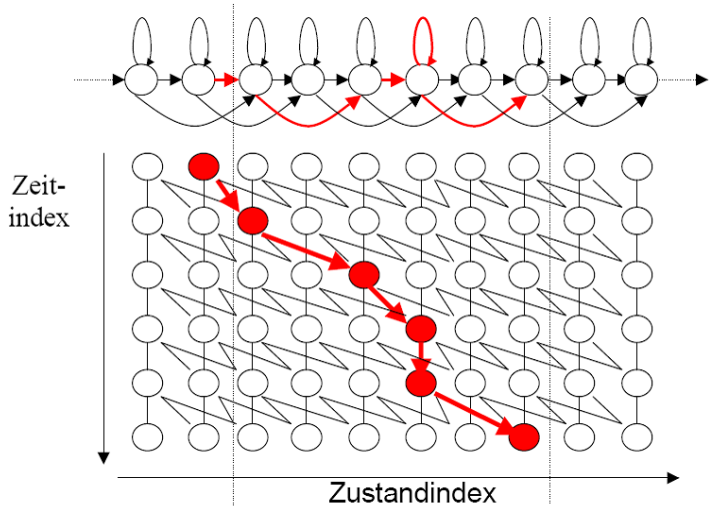


Abbildung 1.5: Viterbi-Decoder

Der gesamte Optimierungsvorgang
1) Initialisierung:

$$\delta_1(i) = \pi_i b_i(O_i), 1 \leq i \leq N \tag{1.21}$$
$$\Psi_t(i) = 0 \tag{1.22}$$

2) Induktion:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), 2 \leq t \leq T, 1 \leq j \leq N \tag{1.23}$$
$$\Psi_t(i) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], 2 \leq t \leq T, 1 \leq j \leq N \tag{1.24}$$

3) Terminierung:

$$P^\star = \max_{1 \leq i \leq N} [\delta_T(i)] \tag{1.25}$$
$$q_t^\star = \arg \max_{1 \leq i \leq N} [\delta_T(i)] \tag{1.26}$$

4) Optimale Zurückverfolgerung der Zustandsfolge:

$$q_t^\star = \Psi_{t+1}(q_{t+1}^\star), t = T-1, T-2, \ldots, 1 \tag{1.27}$$

EM-Algorithmus

Der EM-Algorithmus [2] ist ein allgemeines Verfahren zur Bestimmung von Maximum-Likelihood Schätzwerten von probabilistischen Modellen. Die Bezeichnung Älgorithmusïst eigentlich irreführend, da in der allgemeinen Definition des Verfahrens keine genauen Anweisungen für Rechenschritte gegeben werden, wie es der Begriff des Algorithmus fordert, sondern nur eine allgemeine Beschreibung des Verfahrens und seiner mathematischen Eigenschaften. Für jedes Anwendungsgebiet musst daher ein EM-Algorithmus erfunden werden. Für Hidden Markov-Models heißt dieser Algorithmus „Baum-Welch Algorithmus“.

[2] Expectation-Maximization Algorithmus

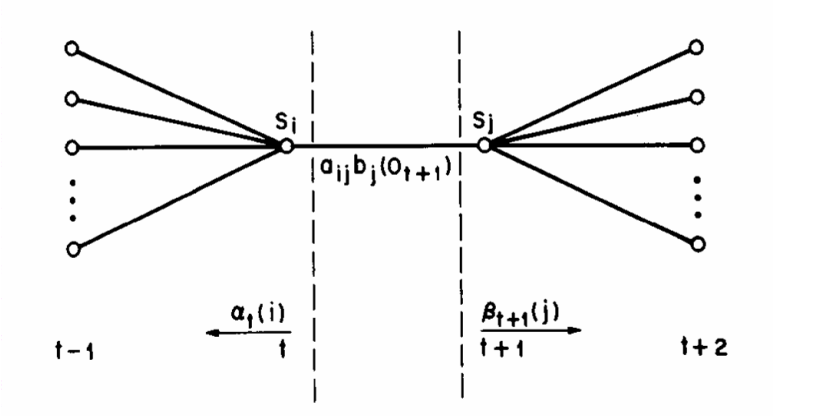


Abbildung 1.6: Baum-Welch-Algorithmus

$\xi_t(i,j), 1 \leq t \leq T; 1 \leq i,j \leq N$ sei die Wahrscheinlichkeit, dass der Übergang zwischen Zustand s_i und Zustand s_j zur Zeit t (mit Forward-und Backward-Variable geschrieben) bei einer gegebenen Realisation O genommen wird:

$$\begin{aligned} \xi_t(i,j) &= P(q_t = s_i, q_{t+1} = s_j | O, \lambda) && (1.28) \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} && (1.29) \end{aligned}$$

Weiterhin ist $\gamma_i(t) = \sum_{j=1}^{N} \xi_t(i,j)$ die Wahrscheinlichkeit, dass Zustand i zum Zeitpunkt t bezüglich O erreicht wird.

Im Estimation-Schritt werden nun die Erwartungswerte für die Anzahl der Übergänge pro Kante berechnet:

- $\sum_{t=1}^{T-1} \gamma_i(t)$ ist die erwartete Anzahl von Übergängen aus Zustand s_i bei Realisierung O.
- $\sum_{t=1}^{T-1} \xi_t(i,j)$ ist die erwartete Anzahl von Übergängen von Zustand s_i nach s_j bei Realisierung O.

Im Maximization-Schritt werden nun die Modellparameter neu berechnet anhand der Erwartungswerte:

- $$\overline{\pi_i} = \gamma_i(1) \tag{1.30}$$

  Die neuen Anfangswahrscheinlichkeiten sind die Wahrscheinlichkeiten, dass Zustand s_i zum Anfang der Zustandssequenz bezüglich O auftritt.

- $$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{1.31}$$

  Die neue Wahrscheinlichkeit für einen Übergang von Zustand s_i nach Zustand s_j ist die erwartete Anzahl der Übergänge von s_i nach s_j dividiert durch die erwartete Gesamtzahl der Übergänge aus s_i durch die Normierung entsteht hier wieder eine Wahrscheinlichkeit $0 \leq a_{ij} \leq 1$.

- $$\overline{b_i k} = \frac{\sum_{t=1}^{T} \gamma_i(t) \delta_{k,O_t}}{\sum_{t=1}^{T} \gamma_i(t)} \tag{1.32}$$

  δ_{k,O_t} ist wie folgt definiert:

  $$\delta_{k,O_t} = \begin{cases} 1; & k = O_t \\ 0; & sonst \end{cases} \tag{1.33}$$

  Die neue Wahrscheinlichkeit für die Ausgabe eines Symbols k im Zustand s_i ist die erwartete Anzahl im Zustand s_i zu sein und dabei das Symbol $k = O_t$ auszugeben, dividiert durch die erwartete Anzahl, überhaupt im Zustand s_i zu sein.

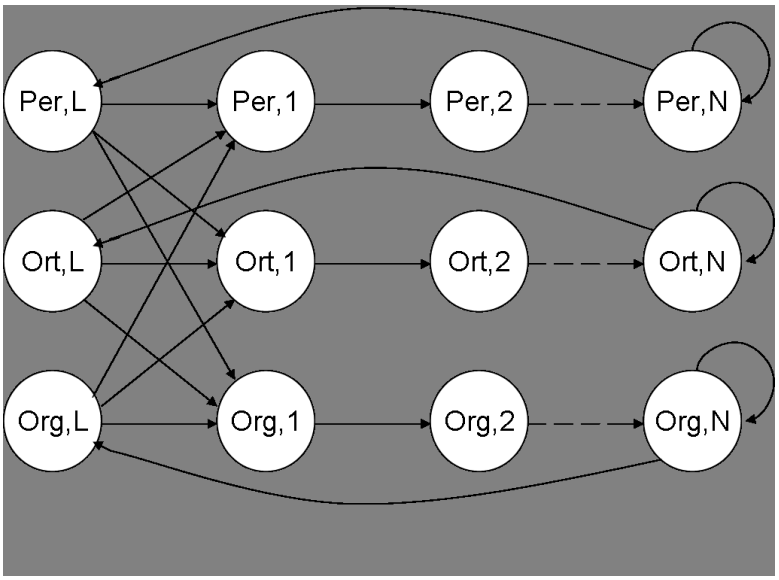
mit hilfe des hmms.png mit hilfe des hmms.png 

Abbildung 1.7: NER mit Hilfe des HMMs

Damit wurde ein neues Modell $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$ berechnet. Wie für den EM-Algorithmus allgemein gezeigt, gilt $P(O|\overline{\lambda}) \geq P(O|\lambda)$.

Zeichenbasiertes NER mit Hidden Markov-Modellen

In einem Zeichenstrom kann mittels eines HMMs Named Entity erkennt und klassifiziert werden. Interessant ist der Ansatz vorallem deswegen, weil er aufzeigt, wie wertvoll bereits Teilsequenzen von Zeichen sein können für NER.

Das Ziel ist, zu einem gegebenen Text automatisch alle Wörter mit den zugehörigen Wortarten zu annotieren. Wie beschrieben kann ein HMM diese Aufgabe lösen, wenn seine Parameter (A, B, π) richtig gesetzt sind. Ein HMM muss also trainiert werden, indem man mithilfe eines vorgetaggten Textes die Parameter so anpaßt, dass das HMM für die Wortfolgen in diesem Text die richtigen Wortartfolgen mit möglichst geringer Fehlerrate ausgibt.

Als Beispiel nehmen wir an, dass man die Wörtern eines Textes ihre Wortarten zuweisen möchte. Alles was vorliegt, ist ein Text als eine Folge von Wörtern. Einen solchen Text kann man in Annäherung als eine Folge von zufälligen Ereignissen interpretieren, zwar dass auf eine Wortart mit einer bestimmten Wahrscheinlichkeit eine andere Wortart folgt, und für jede Wortart dann mit einer bestimmtenWahrscheinlichkeit ein konkretes Wort generiert wird. Betrachtet man den Text als Ergebnis des gesamten Zufallsexperiments, so ist die Abfolge der Wörter erkennbar, nicht mehr die Folge der Wortarten, die die Wörter generierte.

Ein Text kann also als ein Zufallsexperiment mit einer Menge von Zufallsvariablen $O_1, \ldots, O_T, q_1, \ldots, q_T$ gesehen werden. Die Werte der O_i ist die Folge der beobachtbaren Daten (die Realisierung), hier sind die einzelnen Wörter. Die Werte der q_i sind die versteckten Daten, nämlich die zu den Wörtern gehörigen Wortarten.

Die zugrundeliegende Verteilung ist also eine gemeinsame Verteilung der Menge von Zufallsvariablen $\{O_1, \ldots, O_T, q_1, \ldots, q_T\}$. Mit Hilfe des Hidden Markov-Modells können solche Verteilungen modellieren.

Der Aufbau des HMM's wird schon erläutert. Pro Zustand, welcher vom HMM durchlaufen wird, wird ein Zeichen generiert bzw. ausgegeben. Jeder Zustand wird durch ein Tupel bestehend aus Klasse (Person, Ort, Organisation), welche den Offset im aktuellen Wort angibt, identifiziert. Es gibt z.B. einen Zustand (Ort, 2) in welchem die Emission des dritten Zeichens aller NE vom Typ 'Ort' vorgenommen wird. Ein spezieller Endzustand (Ort, L) wird für das Leerzeichen (Wortgrenze) zwischen zwei Wörtern verwendet. Ferner wird eine Outputmodell der Form $P(v_k|v_{k-1}\ldots v_{k-N+1}, s_k)$ benutzt, wobei ck das aktuelle Zeichen und s_k der aktuelle Zustand ist. Die Anzahl Zustände pro Klasse ist beschränkt auf N 'reguläre' Zustände und den Endzustand.

Statistische Modelle jenseits des HMMs

Local-Berechnung-Methoden(Forward- und Backward- , Viterbi-Algorithmus) erleichtern die Analyse von großen Datenmengen, wenn eine Schätzung von Emissionswahrscheinlichkeiten vorhand ist. Der EM-Algorithmus ist eine gute Wahl, wenn schnelle und nicht so präzise Ergebnisse gefordert sind.

- HMM sind für die Klassifikation und Modellbildung insbesondere von Zeitreihen gut geeignet.
- Durch das Modell erhält man viele Informationen über die modellierte Signalquelle.

Trotz erfolgreicher Einsätze der einschlägigen HMM-Modellierungswerkzeuge in etlichen symbolverarbeitenden KI-Anwendungen unterliegt das traditionelle Hidden Markov-Model einer Reihe offenkundiger Schwachstellen und Einschränkungen:

- das endliche Zustandsinventar führt zu dem endlichen Gedächtnis
- die fixe Abhängigkeitsstruktur der Zustandsvariablen
- die eindimensionale Organisation der Ausgabedaten
- insbesondere seine implizit exponentielle Zustandsverweildauer

welche in den nachkommden Abschnitte zur Vorstellung einiger Variationen des Themas HMM gab: Maximum Entropy Markov Models, Conditional Random Fields.

1.2.3 Maximum Entropy Markov Models

1.2.4 Conditional Random Fields

1.3 Methoden der Indexierung und Information Retrieval

1.3.1 Indexierung

Einführung

Mit der Popularität des Internets sind auch die Quellen der Dokumentensammlungen (Zeitungsartikel, Webseiten, Bibliographien, usw) gestiegen, die für die Informationsgewinnung den Nutzern zur Verfügung stehen. Somit ist auch die Frage entstanden, wie man unter dieser großen Informationsmenge, auf effiziente Art und Weise, Dokumente fündig machen kann, die einem relevant erscheinen. Bewährt haben sich dafür Web-Suchmaschinen, welches den Nutzern als Hilfsmittel dient, um bestimmte Dokumente in Sammlungen zu finden. Diese liefern einen leichten und effizienten Zugang zu Informationen. Neben der effizienten Suchfunktion sind Suchmaschinen auch fähig Indexierung vorzunehmen. Viele gängige Suchmaschinen sind Indexbasiert. Man stelle sich den Index wie das Schlagwortverzeichnis eines Buches vor. Durch stöbern im Index nach dem gesuchten Wort, findet man schliesslich den passenden Verweis auf die Seite und schlägt diese dann auf. Vorteil hierbei ist mit Sicherheit die verkürzte Suchzeit. In Kapitel 2 möchte ich vorstellen wie man den Index als Datenstruktur konstruiert und verwalten kann. Zusätzlich will ich in die Indexstruktur, dem Inverted File näher eingehen. Da im Internet eine erheblich große Menge an Dokumenten zur Verfügung stehen, können dementsprechend auch die Indizes groß ausfallen. In Kapitel 3 werde ich zeigen, wie man mit geeigneten Kompressionsverfahren die Indexgröße reduzieren kann. Zum Abschluß in Kapitel 4 will ich auf weitere Indexierungsverfahren eingehen, dem Signature File, dem Suffix Array, sowie dem Suffix Tree.

Index

Inverted Files Ein Index ist eine Datenstruktur, welches Ausdrücke auf Dokumente abbildet, die sie enthalten. Die effizienteste Indexstruktur, die zum Indizieren verwendet wird, ist das Inverted File. Das Inverted File ist Sammlung von Listen indem alle Ausdrücke abgespeichert werden die in Dokumenten vorkommen. Dabei unterscheidet man 2 Varianten des Index. Zum einen das word-level Index, indem angezeigt wird, an welcher Position ein Ausdruck im Dokument vorkommt. Die zweite Variante ist das document-level Index, indem nur Information vorliegt, ob ein Ausdruck in einem Dokument vorkommt, ohne den Hinweis dafür zu liefern an welcher Stelle der jeweilige Ausdruck erscheint. Üblich ist die Zuordnung jedes einzelnen Dokumentes einer Menge von Ausdrücken. Doch wie der Name Inverted File verrät, findet hier eine Zuordnung jedes Terms auf eine Liste von Informationen über das Vorkommen des Terms in Dokumenten statt. Ein Inverted File besteht im wesentlichen aus einem Wörterbuch, indem alle vorkommenden Terme eines Dokuments abgelegt werden. Das Wörterbuch

bildet dann auf ein Posting File ab, indem die doc-id, also Termvorkommen, eventuell auch Wortpositionen abgespeichert sind. Der Vorteil einer solchen Datenstruktur ist der schnelle Zugriff auf Terme und Dokumente. Daraus resultiert natürlich eine kurze Suchzeit. Der Nachteil bei einer großen Anzahl an Wörtern die in Listen abgespeichert werden müssen, ist der hohe Speicheraufwand. Durch Erweiterungen des Index wird beabsichtigt eine Suche möglichst effizient zu machen. Methoden wie Stemming oder Stopping beispielsweise sorgen dafür den Index klein zu halten. Mit Stemming werden Wörter auf ihre Stammforme reduziert. Man betrachtet dabei nicht die Worte sondern die Wortstämme und trennt die Endungen eines Wortes und speichert die Wörter in ihrer Grundform im Index ab. Mit Stopping werden Präpositionen wie (die,von,bzw.,...) weggelassen, die wenig Aussagekraft haben. Dies ermöglicht die Einsparung von Platz bei der Speicherung des Indexes. Daraus resultiert eine effiziente Suche da der Index kleiner ist. Andere Techniken, die verwendet werden um die Größe des Index klein zu halten ist der Thesaurus. Ein Lexikon, indem unterschiedliche Wörter gleicher Bedeutung zusammengefasst werden.

Indexkonstruktion Die Konstruktion eines Indexes, auch Invertierung genannt, beinhaltet die folgenden 3 Schritte: Im ersten Schritt, dem Parsen der Dokumente, werden die Dokumente die zu indizieren sind von vorne bis hinten durchlaufen und eine Liste der Wörter mit der Dokumenten-id erstellt.Dabei können schon die Stopwörter, die den Index unnötig vergrößern, ausgelassen werden. Nachdem man diese Liste alphabetisch mit Hilfe von Sortieralgorithmen sortiert hat, werden Wörter die in der Liste doppelt vorkommen, zusammengefasst. Durch die in der Liste angegebenen Dokumenten-id ist das Vorkommen der Wörter eindeutig bestimmt. Beim Erstellen des Index können auch mögliche Probleme auftreten. Durch das Sortieren des Index müssten alte umsortierte oder neue sortierte Daten auf die Platte abgelegt werden. Diese würde kurzzeitig zu einem erhöhten Speicherverbrauch führen.

Indexverwaltung Da die Aktualisierung eines Index meistens Neu Einfügungen von Wörtern ist, und selten gelöscht wird, wächst natürlich auch die Größe des Index enorm. Der Index wird meist in Speicher verwaltet. Dabei werden kurze invertierte Listen in Speicher fester Größe zugeordnet, lange Listen werden in zusammenhängende Speicherbereiche variabler Größe abgespeichert. Jede invertierte Liste wird in einem Bucket hinterlegt, da sie zu Beginn noch als kurze Liste hineinpasst. Doch mit dem Hinzufügen neuer Terme und dem Wachstum der Liste, droht ein Überlauf des Buckets. Ist der Überlauf geschehen, wird die älteste darin enthaltene Liste zu einer langen Liste. Als Datenstruktur fir einen Index werden meist Verkettete Listen genutzt. Diese ermöglichen eine dynamische Speicherbelegung, und ein einfaches Zufügen von Ausdrücken in Dokumenten.

Indexoptimierung Wie schon erwähnt kann der Index durch zuviele Einträge stetig wachsen. In Kapitel 2 wurden Verfahren erläutert, die die Größe des Index reduzieren. Das Stemming kürzt Wörter auf ihre Stammform ab, und fasst diese in einem Wort zusammen. Bsp. neues, neuem, neuesten wird abgekürzt zu neu. Wenn man in der Suchanfrage den Begriff "neu" eingibt, werden auch all die Dokumente ausgegeben, in denen das Wort neu mit verschiedenen Endungen auftritt. Durch das Auslassen von Präpositionen, Artikel, also sogenannten Stopwörtern, verringert sich der Speicherbedarf des Index.

Kompressionsverfahren

Es gibt aber auch andere Ansätze mit denen man einen Index klein halten kann. Und zwar sind dies Komprimierungsverfahren, die mittels geeigneter Kodierung zu einem erwünschten Ergebnis führen. Diese Kodierungsmethoden ermöglichen eine Reduzierung der Textgröße auf fast 50 Prozent. Daraus resultiert eine geringere Auwertungszeit von Fragen. Zusätzlich kann in kurzer Rechenzeit ein Index erstellt werden.

Golomb-Code Der Golomb-Code wurde 1966 von Solomon W. Golomb vorgestellt. Dabei handelt es sich um eine Darstellungsform für alle positiven Zahlen, einschliesslich der Null. Ein großer Vorteil an diesem Kodierungsverfahren ist die Abspeicherung von Daten unbekannter Größe. Es ermöglicht eine schnelle Kodierung und eine sparsame Nutzung des Speichers. Die Idee des Golomb Ansatzes besteht darin, die darzustellende Zahl n durch den Quotienten q, einem frei wählbarem Parameter b und seinem Rest r zu beschreiben. Dazu wird im ersten Schritt der Quotient der Zahl n berechnet mit der simplen Formel

$$q = \left\lfloor \frac{n-1}{b} \right\rfloor$$

. Das Ergebnis des Quotienten wird unär kodiert ausgegeben, und an das Ende der Bitfolge wird eine logische 0 angehängt. Im zweiten Schritt bestimmt man den Rest r der Zahl n mit Hilfe der Formel:

$$r = n * (q * b) - 1.$$

Das Ergebnis wird nun binär ausgegeben. Zum Abschluss hängt man den zweiten Teil des Code-Wortes an das Ergebnis aus dem ersten Schritt und man erhält das fertige Code Wort. Zusätzlich zum Golomb-Code gibt es noch den Rice-Code. Der Rice-Code ist ein Spezialfall des Golomb-Code. Hierbei ist der Paramater b nicht frei wählbar, sondern eine zweier Potenz.

Weitere Indexierungsverfahren

Selbstverständlich gibt es auch weitere Ansätze um Indexierung vorzunehmen. So sind das Suffix Array, der Suffix Tree und die Signature Files Verfahren mit denen man einen Index erzeugen kann.

Signature Files In einem Signature File können Einträge einer Datenbasis, welche eine bestimmte Bedingung erfüllen, effizient herausgefiltert werden. Vor allem große Textbestände sind, innerhalb kurzer Zeit nach Schlüsselwörtern durchsuchbar. Die Signaturen selbst werden durch die Abbildung von Wörtern auf Bitstrings mittels Hash-Funktion erzeugt. Dabei werden die Dokumente in einer Textdatei, die Signaturen in einer separaten Datei abgespeichert. Ein Vorteil gegenüber dem Inverted File ist der geringe Speicherplatzbedarf, jedoch ist dieses Verfahren ineffizient für große Datenbanken und ein Nachteil sind die falschen Einträge, sogenannte False Drops, die herausgefiltert werden. Die Konstruktion eines Signature File besteht darin, das Dokument in logische Blöcke zu unterteilen. Für jeden Block werden die einzelnen Signaturen gebildet, wobei jeder Block eine konstante Anzahl an Wörtern hat.

Suffix Array Ein Suffixarray ist ein Array, welches die Suffixe, in lexikographischer Reihenfolge angeben kann. Der Text ist als eine lange Zeichenkette abgespeichert. Jedes Teilstück des Textes ist eindeutig durch seine Position bestimmt, wobei die Positionen im Text frei wählbar sind. Der Vorteil des Suffixarrays als Index zu nutzen liegt darin, möglichst schnell jedes Teilwort innerhalb des Wortes zu lokalisieren. Durch die lexikographische Sortierung können die Suffixe effizient mit Hilfe der binären Suche gesucht werden.

Suffix Tree Ein Suffix-Tree dient dazu, die Suffixe in einer baumartigen Struktur darzustellen. Die Suffixe enden als Blatt im Baum. Also hat ein Suffix-Tree soviele Blätter wie eine Zeichenkette Suffixe besitzt. Aus den inneren Knoten gehen mindestens zwei Kinder aus. Jede Kante des Suffix-Trees definiert ein Teilwort der Zeichenkette.

Beispiel : *acacb*

1. Suffix= cacb
2. Suffix= acb
3. Suffix= cb
4. Suffix= b

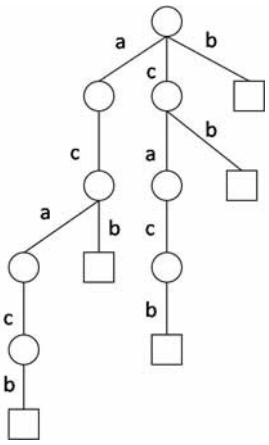


Abbildung 1.8: Suffix Baum

1.3.2 Suchmaschinen

Neben den im vorherigen Abschnitt beschriebenen Indizierungstechniken zum Zugriff auf die gespeicherten Dokumente, benutzen moderne Suchmaschinen Ranking Verfahren um die Ergebnisse einer Suchanfrage zu sortieren. Diese Verfahren werden als Page-Ranking Verfahren bezeichnet, da sie die Linkliste der Ergebnisse einer Suchmaschine absteigend sortieren. Die Relevanz einer Page dient hierbei als Sortierkriterium. Dieses Kriterium soll möglichst objektiv und maschinell berechenbar sein und das menschliche Interesse sowie Aufmerksamkeit widerspiegeln.

Page-Ranking Algorithmen können in zwei Kategorien aufgeteilt werden. Semantisch unabhängige, vor dem Indizierungsschritt ausführbare Algorithmen, welche die Dokumente a priori mit einem statischen Rank versehen und semantisch abhängige Verfahren welche nach der Indizierung die Dokumente aufwändig verarbeiten, teilweise sogar zur Query Zeit ausgeführt werden.

Die statischen Ranking Verfahren erzeugen ein globales Ranking von Webseiten durch Analyse der Verbindungsstruktur des Netzes. Sie betrachten nur die Topologie des Webgraphen und ignorieren die Inhalte und den semantischen Kontext einer Webseite. Die Verweise auf andere Webseiten sind einzige Informationsträger.

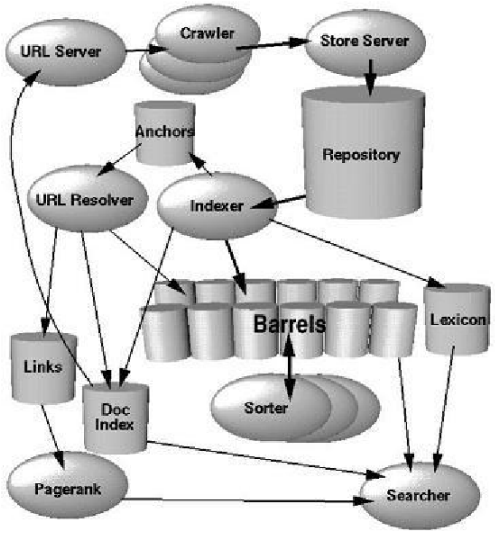
Die zweite Kategorie, der semantischen Page-Ranking Verfahren, nutzt Informationen der mit Meta-Daten angereicherten Dokumente. Durch aufwändiges Preprozessing extrahierte Meta-Daten der Inhalte sind hier die Informationsträger.

In diesem Abschnitt werden beide Ranking Verfahren anhand zwei Suchmaschinen erläutert.

Google Architektur

Google ist zur Zeit die mit Abstand bekannteste und am häufigsten benutzte Suchmaschine. Diese positive Entwicklung resultiert aus dem ausgewogenem Mix aus Performance, Benutzerfreundlichkeit und Qualität der Suchergebnisse. Lawrence Page und Sergey Brin, die beiden Gründer von Google beschreiben in "Anatomy of a Large-Scale Hypertextual Web Search Engine" [3] den Aufbau und Einsatzmöglichkeiten Ihrer Suchmaschine, sowie in "The PageRank citation ranking: Bringing order to the Web" [3] das PageRank Verfahren zur Analyse der Webtopologie.

Die System-Architektur von Google besteht aus mehreren miteinander kooperierenden Komponenten:



Die Crawler, ausgelegt auf größtmögliche Paralelität, wandeln URLs über Domain Name Service (DNS) Requests in IP-Adressen um. Sie stellen mittels HTTP simultan, mehrere Verbindungen mit den Server her, um die erreichten Pages herunterzuladen.

Im Repository wird der komplette HTML-Code jeder heruntergeladenen Page mit zlib komprimiert gespeichert.

Die Aufgabe des Indexers besteht im dekomprimieren und analysieren der gespeicherten Pages. Jede heruntergeladene Page bekommt eine DocID. Zusätzlich werden Hits als Wortvorkommen in so genannten Hit Listen

registriert. Die Hit Listen beinhalten Informationen über das Auftreten eines bestimmten Wortes in einem bestimmten Dokument, einschließlich Position, Schriftkegel und Hervorhebungen. Eine weitere wichtige Funktion des Indexers ist das Abspeichern aller Links Page in einer Anchor Datei. Die Sonderbehandlung von Link in Dokumenten ermöglicht Indexierung von Seiten, die aus Bildern, Programmen oder Datenbanken bestehen.

Für jedes Dokument werden zwei Indizes erstellt, der Forward Index ist bereits teilweise sortiert und in einer Reihe von Barrels gespeichert. Die Barrels ermöglichen einen hohen Grad an Parallelität, der Index wird auf viele relativ kleine Barrels aufgeteilt. Jedes Barrel beinhaltet also nur eine kleine Anzahl an WordIDs, enthält ein Dokument eine WordID, so wird seine DocID in dem entsprechenden Barrel gespeichert. Der Datensatz enthält zusätzlich eine Liste von WordIDs mit Hitlisten. Der Invertierte Index wird in den gleichen Barrels wie der Forward Index gespeichert, allerdings ist er bereits durch den Sorter verarbeitet worden. Für jede gültige WordID enthält das Lexicon einen Zeiger auf das entsprechende Barrel, dort befindet sich ein Zeiger auf eine DocList die alle Vorkommen des Wortes in allen Dokumenten zusammenfasst.

Im Document-Index werden alle zusätzlichen Informationen über ein Dokument gespeichert. Dabei enthält jeder Eintrag den Dokumentstatus, einen Zeiger ins Repository, eine Dokumentprüfsumme und verschiedene Statistiken.

Die Sorter arbeiten kontinuierlich daran, den nach der DocID sortierten Forward Index in einen nach WordID sortierten Inverted Index umzuwandeln.

Während das gesamte übrige System kontinuierlich daran arbeitet, die Datenbasis zu aktualisieren und zu erweitern, ist der Searcher die einzige Anfrageverarbeitende Komponente. Der Searcher verarbeitet Suchanfragen, erzeugt Ergebnislisten und sortiert sie nach Relevanz.

PageRank

Lawrence Page und Sergey Brin beschreiben in ”The PageRank citation ranking: Bringing order to the Web” [3] das von Google verwendete PageRank Verfahren zur Analyse der Webtopologie. Dieses Verfahren basiert auf der Idee eines globales Rankings von Webseiten, das durch Analyse der Verbindungsstruktur des Netzes erzeugt wird. Die Autoren nehmen an, das wichtige Seiten besonders oft von anderen Seiten verlinkt werden, ein Webautor signalisiert durch einen Link auf eine andere Seite die Bereitschaft eine andere Quelle in den eigenen Inhalten zu zitieren.

Das Web ist ein mittels HTTP Protokoll realisiertes, virtuelles Netz aus HTML-Dokumenten, die durch Hyperlinks miteinander verbunden sind. Das Verfahren Nutzt diese Struktur in dem es das Web als Graph betrachtet.

Webgraph - Es sei $G = (V,E)$ ein gerichteter Graph mit:

- V der Knotenmenge statischer Webseiten
- E der Kantenmenge aus Hyperlinks

Jeder Knoten des Webgraphen ist also ein HTML-Dokument, verbunden mit anderen Dokumenten durch Anchor Tags. Eingehende Kanten eines Dokuments bezeichnen wir als B Backlinks oder ”inedges”, ausgehende Kanten als F Forward Links oder ”outedges”.

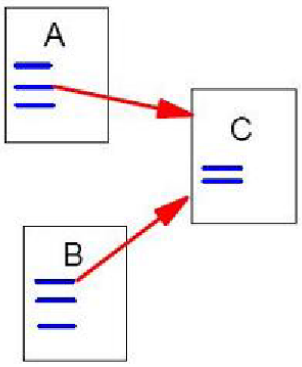


Abbildung 1.9: A und B sind Backlinks - C erreichbar durch Forward Links

Der PageRank Algorithmus betrachtet die Backlinks einer Webseite, allerdings werden nicht alle eingehenden Links gleich bewertet. Der Rank einer Seite ist die Summe der Ranks ihrer Backlinks.

PageRank Sei u eine Webseite und ein Knoten in G

- F_u die Menge aller Forward Links von u,
- B_u die Menge aller Backlinks von u,
- $N_u = |F_u|$ die Anzahl aller Nachfolger von u
- d soll Damping Faktor zur Normierung des Ranks sein
- E sei eine Initialbelegung des statischen PageRanks über alle Seiten.

R ist eine Rankingfunktion von u und ihrer Vorgängerin v, der Form:

$$R(u) = d \sum_{v \in B_u} \frac{R(v)}{N_v} + dE(u)$$

Der PageRank einer Seite u wird also rekursiv aus der Summe der PageRanks derjenigen Seiten die auf u verweisen. Allerdings wird nur eine einzige Verbindung zu u, anteilig zur der Menge aller Links betrachtet. Für die Analyse des Algorithmus betrachten wir Adjazenzmatrix A des Webgraphen G mit:

- $a_{v,u} = \frac{1}{N_v}$ falls eine Kante von v nach u existiert.
- $a_{v,u} = 0$ sonst.

PageRank Starte in $R_0 = E$ der Initialbelegung und laufe durch die folgende Schleife bis R_j konvergiert.

- $R_{i+1} \leftarrow AR_i$
- $d \leftarrow ||R_i||_1 - ||R_{i+1}||_1$
- $R_{i+1} \leftarrow R_{i+1} + dE$

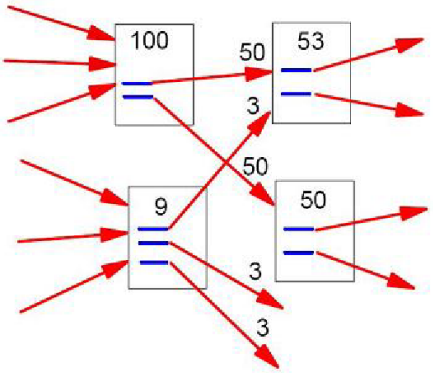


Abbildung 1.10: Eine Iteration der vereinfachten Rankingfunktion

Ohne die Initialbelegung und den Damping Faktor würde der PageRank bei jeder Iteration zu den Senken fliessen. Also zu Seiten die auf keine anderen Seiten verlinken (Dangling Links) oder auf dem Webgraph Kreise bilden (Rank Sinks). Um den negativen Effekt der Dangling Links entgegenzuwirken, werden die PageRanks dieser Seiten erst berechnet, wenn die eigentliche Berechnung bereits abgeschlossen ist. Seiten die durch gemeinsame Verlinkung auf dem Webgraph Kreise bilden und nur auf sich selbst verweisen, erzeugen eine Schleife in der der Rank steigt aber nicht weiter verteilt wird, da dieser Kreis keine ausgehenden Kanten besitzt.

Der Damping Faktor, der stets zwischen 0 und 1 liegt wirkt diesem Problem entgegen und verringert den Ausmaß der PageRank Weitergabe.

Das statische PageRank Verfahren von Google ist Unabhängig von den Inhalten einer Webseite. Der Algorithmus konvergiert schnell und liefert selbst nach wenigen Iterationen gute Ergebnisse. Nach 45 Iterationen ist die PageRank Berechnung von 161 Mio. Links abgeschlossen. Bei Verdoppelung der Anzahl von Links konvergiert konvergiert der Algorithmus bereits nach 52,5 Iterationen. Der Algortihmus wird der Datenaufbereitung ausgeführt und ist schont Ressourcen die zu Query Zeit benötigt werden.

Zusätzlich lässt sich das Verfahren, wie in "A Unified Probabilistic Framework for Web Page Scoring Systems" von Diligenti, Gori etal. [5] beschrieben, auch probabilistisch als Rando Surfer Modell interpretieren.

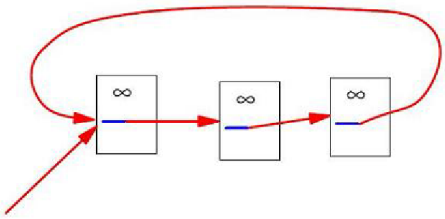


Abbildung 1.11: Ein Kreis im Webgraph: Rank Sink

Alvis

Alvis Superpeer Semantic Searchengine

1.3.3 Lernende Suchmaschinen

Vorwort

Der vorliegende Abschnitt dient zur Lernenden Suchmaschinen. Und er basiert auf einem Paper von Thorsten Joachims [7]. Das klassische Anwendungsgebiet der Lernenden Suchmaschinen sind Literaturdatenbanken, die heute in Form Digitaler Bibliotheken zunehmend an Bedeutung gewinnen. Lernenden Suchmaschinen sind besonders populär geworden durch die Anwendung in Internet-Suchmaschinen; dadurch kommt jeder Internet-Nutzer mit IR-Methoden in Berührung. Jeder, der eine dieser Anwendungen wiederholt genutzt hat, wird die wesentlichen Unterschiede zwischen Suchmaschinen-Anwendungen und denen klassischer Datenbanksysteme leicht erkennen:

- Meistens durchläuft der Prozess der Anfrageformulierung mehrere Iterationen, bis passende Antworten gefunden werden.
- Anfragen liefern potentiell sehr viele Antworten, aber nur wenige davon sind für den Nutzer interessant.
- Das vorgenannte Problem entschärft sich durch die vom System bereitgestellte Rangordnung der Antworten, wodurch potentiell relevante Antworten gehäuft am Anfang der Rangliste auftauchen (z.B. betrachten bei Internet-Suchmaschinen mehr als 90

Was ist eine lernende Suchmaschine?

Zur Definition des Gebietes legen wir hier die Beschreibung der Aufgaben und Ziele. Eine lernende Suchmaschine ist eine Suchmaschine mit künstlicher Intelligenz, die selbständig lernt, für wen etwas interessant sein könnte, indem sie die Benutzer befragt.

Trainingsdaten

Lernende Suchmaschine braucht Trainingsdaten, damit sie lernen kann. Die Trainingsdaten können durch die Befrage der Besuchern automatisch erzeugen. Sie werden durch eine Datengruppe $Trio(q,r,c)$ dargestellt.

- q: Anfrage
- r: Die Rangliste der Suchergebnisse
- c: URL-Liste, die der Benutzer gewählt hat.

Um die Definition von $Trio(q,r,c)$ besser zu verstehen, sehen wir ein Beispiel mit Anfrage 'support vector machine'. Die Anfrage 'support vector machine' liefert 10 Antworten, aber nur $1,3,7$ sind für den Nutzer interessant (siehe Abb.1.12).
Bei dem Beispiel sind:

- q: 'support vector machine'

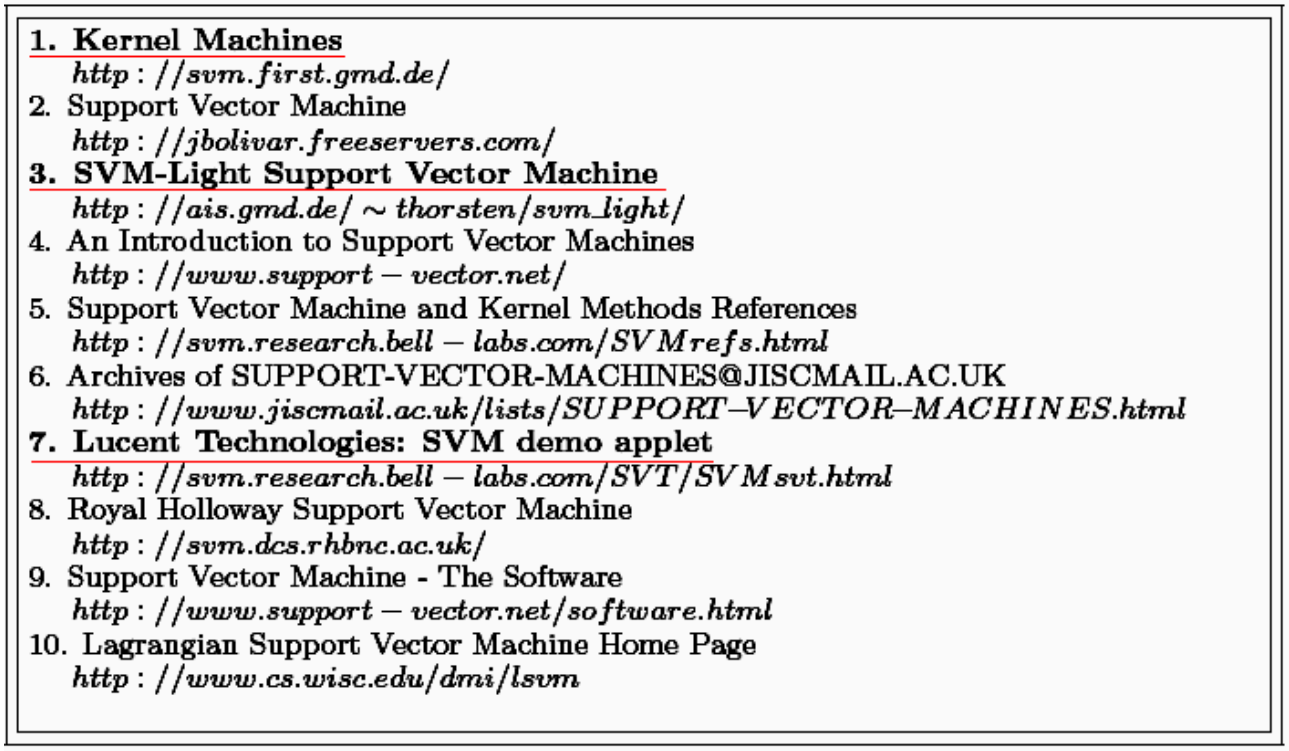


Abbildung 1.12: die Suchergebnisse mit der Anfrage 'support vector machine'

- r: Liste (1,2,3,4,5,6,7,8,9,10)
- c: (1,3,7)

Welche Informationen sind relevant?

Was können die lernenden Suchmaschinen durch die Trainingsdaten lernen. Bei dem Beispiel (Abb.1.12) hat der Benutzer $1,3,7$ gewählt. Es ist aber nicht sicher, dass $1,3,7$ absolut relevant Sind. Aber man kann die relativ relevanten Dokumente bestimmen. Der benutzter hat nach dem Rang2 den Rang3 gelesen. Ja, der Rang3 ist wahrscheinlich relevanter als dem Rang2. man kann ihm mit einem Relation r^* darstellen link3 $<r^*$ link2. Analog kann man die ganz Relation r^* bei dem Beispiel (Abb.1.12) bestemmen:

link3 $<r^*$ link2 link7 $<r^*$ link2
link7 $<r^*$ link4
link7 $<r^*$ link5
link7 $<r^*$ link6.
Die Relation r^* hat also partielle Information der gesuchte Sortierung der Suchergebnisse.

Darstellung der Relation $<r^*$

Für eine Suchergebnisliste gibt es m Dokumente $D=\{d_1,\ldots,d_m\}$. Die Relation r^* kann man mit einer Matrix M darstellen.
$((M \subset D \times D))$
$(d_i <r^* \ d_j) \Longrightarrow (M_{ij}=1 \text{ und } M_{ji}=0)$
Bei dem Beispiel (Abb.1.12)

$$M=\begin{pmatrix} - & * & * & * & * & * & * & * & * \\ * & - & 1 & * & * & * & 1 & * & * & * \\ * & 0 & - & * & * & * & * & * & * \\ * & * & * & - & * & * & 1 & * & * \\ * & * & * & * & - & * & 1 & * & * \\ * & * & * & * & * & - & 1 & * & * \\ * & 0 & * & 0 & 0 & 0 & - & * & * \\ * & * & * & * & * & * & * & - & * \\ * & * & * & * & * & * & * & * & - \end{pmatrix}.$$

(* unbekannt)

Die Funktion für lernende Suchmaschinen

Mit den Trainingsdaten müssen die lernenden Suchmaschinen eine Funktion finden, damit die Dokumente gut sortiert werden können. Um die Suchergebnisse zu verbessern, muss die Qualität der Funktion zu beurteilen. In Statistik ist Kendall's τ sehr oft benutzt, um 2 Range zu vergleichen.
Für 2 Range ($r_a \subset D \times D$) und ($r_b \subset D \times D$),
P ist die Anzahl der übereinstimmenden Paare,
Q ist die Anzahl der nicht übereinstimmenden Paare, ist dann Kendall's τ definiert durch:

$$\tau = \frac{P-Q}{P+Q} = 1 - \frac{2Q}{\binom{m}{2}} \tag{1.34}$$

Um die Kendall's τ besser zu verstehen, sehen wir folgendes Beispiel für 2 Rangen r_a und r_b:
$r_a : d_1 <_{r_a} d_2 <_{r_a} d_3 <_{r_a} d_4 <_{r_a} d_5$
$r_b : d_3 <_{r_b} d_2 <_{r_b} d_1 <_{r_b} d_4 <_{r_b} d_5$.
Die Anzahl der nicht übereinstimmenden Paare Q ist 3 ($\{d_2,d_3\},\{d_1,d_2\},\{d_1,d_3\}$). Und die Anzahl der übereinstimmenden Paare P ist 7. Damit kann man τ durch die Definition errechnen:

$$\tau = \frac{P-Q}{P+Q} = \frac{7-3}{7+3} = 0,4. \tag{1.35}$$

Wenn Q=0, hat τ die Maximum 1. Und umgekehrt hat τ die Minimum -1. $r_{f(q)}$ ist das System mit der Frage q liefert Rangliste. $\tau(r_{f(q)}, r*)$ ist dann die Qualität der Suchergebnisse mit einer Funktion f nach die Anfrage q. Man kann die $\tau(r_{f(q)}, r*)$ als die Präzision der Suchergebnisse betrachten. Ist $\tau(r_{f(q)}, r*)$ groß, ist die Präzision der Suchergebnisse dann groß.
Wir sind jetzt in einer Position das Problem der zu definieren. Das Ziel ist die Funktion für die zu erwartenden Kendall's τ

$$\tau_{p(f)} = \int \tau(r_{f(q)}, r^*) d_{Pr(q,r^*)} \tag{1.36}$$

für eine feste, aber unbekannte Verteilung $Pr(q,r^*)$ von Abfragen und Zielgruppen Rangliste auf ein Dokument Sammlung D mit m Dokumenten maximal. Während das Ziel der Lerner ist nun definiert, bleibt die Frage, ob es möglich ist, um die Funktion zu optimieren?

Der Algorithmus für lernende Suchmaschinen

Die meiste Arbeit der lernenden Suchmaschinen im Information Retrieval ist nicht die Formulierung von Dokumente, aber die vereinfachte Aufgabe, eine binäre Klassifikation Problem mit den beiden Klassen relevanteünd nicht relevant". Eine solche Vereinfachung hat mehrere Nachteile.
Die nicht relevanten"Dokumente haben eine starke Mehrheit. Ein Lernender muss eine Regel haben, um die Dokumente maximale automatische genau zu klassifizieren. Aber es ist noch wichtig, und gezeigte, dass solche absoluten Relevanz Urteile nicht existiert ist, so dass ist sie einfach nicht verfügbar. Deshalb sollen wir einen Algorithmus finden.
Für identische q und ihre r^*:

$$(q_1, r_1^*), \cdots, (q_n, r_n^*) \tag{1.37}$$

wählt Lerner L eine Funktion f, so dass

$$\tau_s(f) = \frac{1}{n} \sum_{i=1}^{n} \tau(r_{f(q_i)}, r_i^*) \tag{1.38}$$

maximiert wird. Ist es möglich, einen Algorithmus und eine Familie von Rang Funktionen F zu entwerfen, so dass eine Funktion $f \in F$ Maximierung effiziente gefunden wird, und diese Funktion weit über den Trainingsdaten verallgemeinert werden kann. Betrachten Sie die linearen Funktionen

$$(d_i, d_j) \in f_{\bar{\omega}}(q) \Leftrightarrow \bar{\omega}\phi(q,d_i) > \bar{\omega}\phi(q,d_j). \tag{1.39}$$

ω ist ein Gewichtsvektor, und $\phi(q,d)$ ist ein Diagramm mit der Dimension, die die Eigenschaft zwischen Anfrage q und Dokument d beschreibt.
Mit unterschiedlichen Gewichtsvektoren ω werden die Dokument unterschiedlich sortiert (siehe Abb.1.13). Jetzt

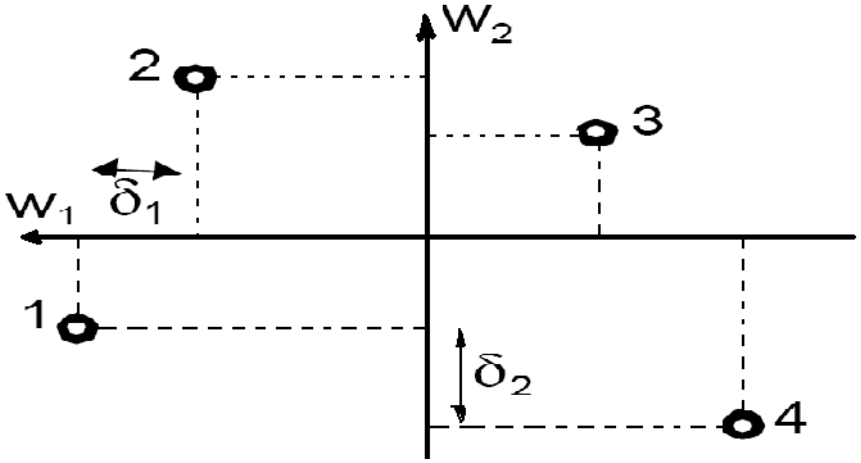


Abbildung 1.13:

t_i	ω_i	d_{1i}	d_{2i}	d_{3i}	d_{4i}
side effect	2	1	0,5	1	1
drugs	2	1	1	1	1
memory	1	1		1	
cognitive ability	1		1	1	0,5
not aging	-2		1		
Retrivalgewicht		5	2	6	4,5

haben wir sofort eine Frage, wie kann man die Dokumenten als Punkte auf einem Diagramm zeichnen? Das folgende Beispiel illustriert die Anwendung der Gewichtsvektoren bei Suchergebnissen.
Beispiel-Frage: 'side effect of drugs on memory and cognitive abilities, not aging'
Entsprechend den Retrivalgewichten werden die Dokumente in der Reihenfolge $d_3;d_1;d_4;d_2$ ausgegeben.
Die Maximierung (Formel 1.38) ist direkt äquivalent zu minimieren die Zahl der Q der nicht übereinstimmenden Paare in der Formel (1.34). Für die Klasse der linearen Rangfunktionen (1.39), dies ist äquivalent das Gewicht Vektor zu finden, so dass die maximale Anzahl der folgenden Ungleichungen erfüllt ist.
$\forall(d_i,d_j) \in r_i^* : \bar{\omega}\phi(q_1,d_i) > \bar{\omega}\phi(q_1,d_j)$
...
$\forall(d_i,d_j) \in r_n^* : \bar{\omega}\phi(q_n,d_i) > \bar{\omega}\phi(q_n,d_j)$
Es ist leid NP-Schwer Problem. Aber genau wie bei der Klassifizierung SVMs ist es möglich durch die Einführung (nicht negativ) Schlupfvariablen $\xi_{i,j,k}$ und Minimierung $\sum\xi_{i,j,k}$ der zu Lösen.

(RANKING SVM)

$$\text{Min.}V(\bar{\omega},\bar{\xi}) = \frac{1}{2}\bar{\omega}\cdot\bar{\omega} + C\sum\xi_{i,j,k} \tag{1.40}$$

mit Nebenbedingung:

$$\forall(d_i,d_j) \in r_i^* : \bar{\omega}\phi(q_1,d_i) \geq \bar{\omega}\phi(q_1,d_j) + 1 - \xi_{i,j,1} \tag{1.41}$$
$$\dots \tag{1.42}$$
$$\forall(d_i,d_j) \in r_n^* : \bar{\omega}\phi(q_n,d_i) \geq \bar{\omega}\phi(q_n,d_j) + 1 - \xi_{i,j,n} \tag{1.43}$$
$$\forall i,j,k : \xi_{i,j,k} \geq 0 \tag{1.44}$$

C ist positiv Konstante, die den Ausgleich zwischen der Minimierung von $\frac{1}{2}\bar{\omega}\cdot\bar{\omega}$ und der korrekten Klassifizierung der Trainingsbeispiele regelt. Durch die Neuordnung der Ungleichung (1.41) als

$$\bar{\omega}\phi(q_1,d_i) - \bar{\omega}\phi(q_1,d_j) \geq 1 - \xi_{i,j,1} \tag{1.45}$$

wird deutlich, dass die Optimierung Problem ist äquivalent zu einer Klassifikation SVM auf paarweise Unterschied Vektoren $\bar{\omega}\phi(q_1,d_i) - \bar{\omega}\phi(q_1,d_j)$. Es kann gezeigt werden, dass die erlernten Retrieval Funktion $f_{\bar{\omega}^*}$ immer

als eine lineare Kombination der Vektoren kann.

$$(d_i, d_j) \in f_{\bar{\omega}^*}(q) \tag{1.46}$$
$$\Longleftrightarrow \bar{\omega}^* \phi(q, d_i) > \bar{\omega}^* \phi(q, d_j) \tag{1.47}$$
$$\Longleftrightarrow \sum \alpha^*_{k,l} \phi(q_k, d_l) \phi(q, d_i) > \sum \alpha^*_{k,l} \phi(q_k, d_l) \phi(q, d_j) \tag{1.48}$$

Der Vektorraum und die zugehörigen Trainingsdaten sind durch eine Kernelfunktion $\sum \alpha^*_{k,l} \phi(q_k, d_l) \phi(q, d_i)$ in einen Raum mit so hoher Dimension zu überführen, dass sich die Trainingsdaten dort linear trennen lassen.

Experiment

Um in der Lage die Qualität der verschiedenen Retrieval Funktionen zu vergleichen, wird die Methode beschrieben in [T. Joachims 2002] verwendet. Die zentrale Idee ist es, zwei Rankings in der gleichen Zeit zu vergleichen. Man kombiniert zwei Rangliste A und B zu C. Es muss die folgende Bedingung gelten. Die oben L Links der kombinierten Rang C enthalten die oben K_a Links von A und der oben K_b Links von B mit $|K_a - K_b| \leq 1$. Mit anderen Worten, wenn der Benutzer die Links von C von oben nach unten scannt, hat er an einem beliebigen Punkt fast ebenso viele Links von der Spitze von A wie von der Spitze des B gesehen.
Ein Beispiel ist in Abb.1.14. Die von zwei Retrieval Funktionen kombiniert Ergebnisse C wird an den Benutzer.

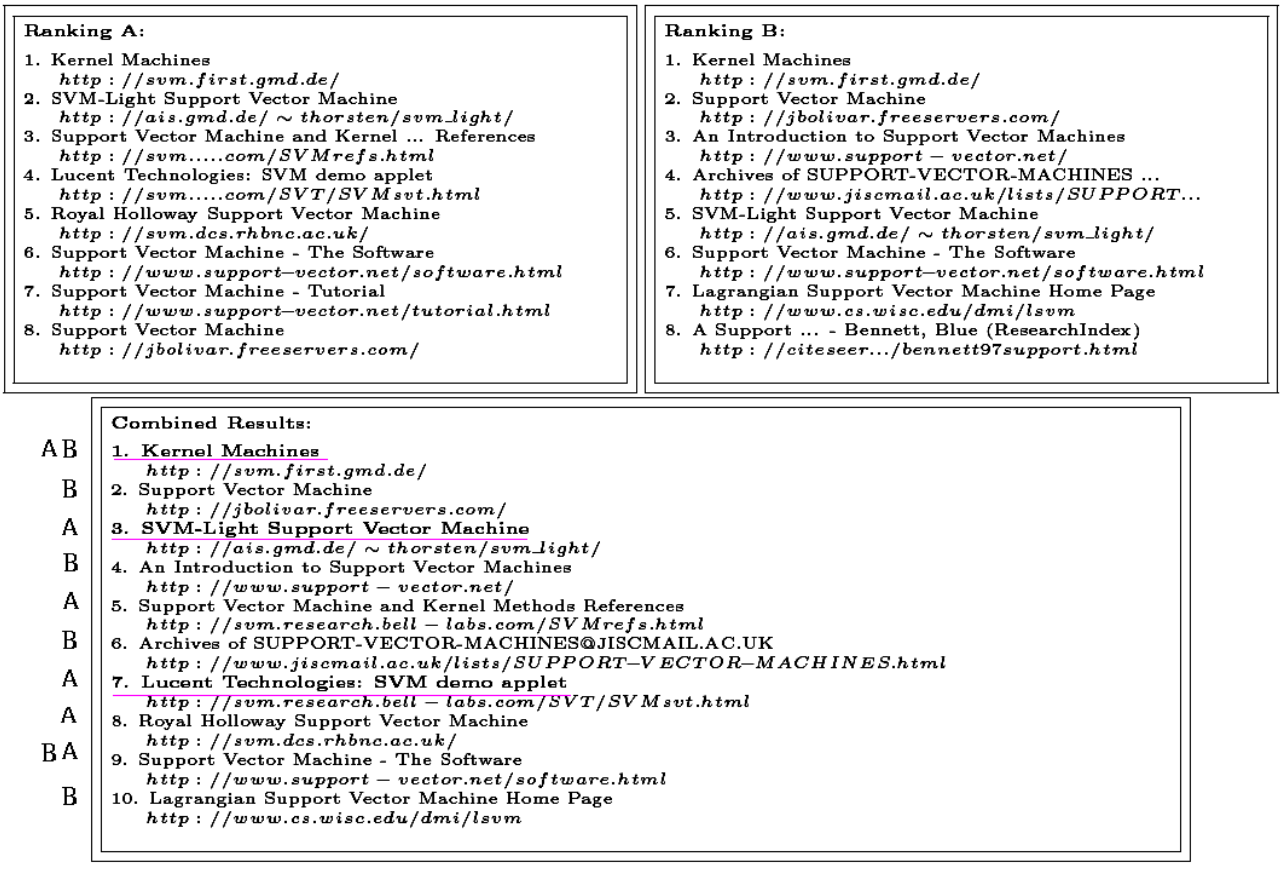


Abbildung 1.14:

In diesem Beispiel hat der Benutzer auf den Links 1, 3 und 7 angeklickt. In dem Beispiel muss der Benutzer die oben 4 Links aus einzelnen Rankings gesehen haben, da er auf den Link 7 in der kombinierte Ranglist C angeklickt hat. Er hat auf 3 Links in der oberen 4 im Ranking A (1, 2 und 4) angeklickt, sondern nur auf 1 Link in Rang B (1). Es ist deshalb zu dem Schluss, dass die oben 4 Links von A besser als jene von B für diese Anfrage sind.
Am Paper von Thorsten Joachims [7] wird die lernende Funktion von SVM mit Google, MSNSearch und Toprank vergleicht. Die Ergebnisse siehe man im Tab.1.3.

Comparison	more clicks on learned	less clicks on learned
Learned vs. Google	29	13
Learned vs. MSNSearch	18	4
Learned vs. Toprank	21	9

Comparison	tie(with clicks)	no clicks	total
Learned vs. Google	27	19	88
Learned vs. MSNSearch	7	11	40
Learned vs. Toprank	11	11	52

Tabelle 1.3:

1.4 Maschinelle Lernverfahren

1.4.1 Klassifikation

SVM

Eine Support Vector Machine löst ein binäres Klassifierungsproblem. Sie lernt auf einer bereits klassifizierten Menge von Trainigsbeispielen und entscheidet dann für Test-Elemente, zu welcher der zwei Klasse sie gehören. Das eigentliche Lernen besteht darin, eine Hyperebene H zu berechnen, die die Beispiele beider Klassen trennt. Danach kann anhand der Lage der Test-Elemente zur Hyperebene eine Klassifizierung erfolgen. Zunächst gilt es aber unter den vielen möglichen trennden Ebenen die optimale zu finden. Optimal wird dabei so verstanden, dass der Abstand von der Ebene zum nahesten Beispiel beider Klassen maximiert wird. Es liegt also ein Optimierungsproblem vor, welches nachfolgend formalisiert werden soll.

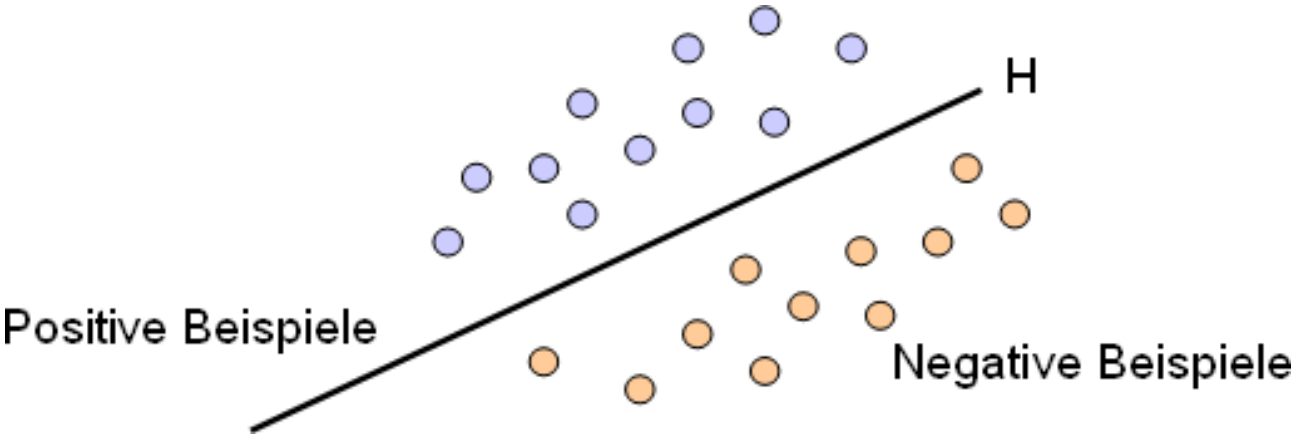


Abbildung 1.15: trennende Hyperebene

Es sei o.B.d.A. angenommen, dass die Beispiele in positive und negative unterteilt seien. Vorerst wird vorrausgestzt, dass die Trainingsdaten linear trennbar sind, es also eine trennende Hyperebene gibt. Der Fall linear nicht trennbarer Daten wird nach aufbauend auf diesen Erkenntnissen betrachtet.

(x_i, y_i) $(i = 1, ..., m)$ seien m Trainingsbeispiele, bestehend aus einem Punkt $x_i \in RR^d$ und der Klassifizierung[3] $y_i \in \{-1, 1\}$. $w \in RR^d$ sei der zu bestimmende Normalenvektor der Hyperebene (siehe Abbildung 1.16), $b \in RR$ ihr Bias. Als Ebenengleichung wird die Variante mit dem Skalarprodukt gewählt, so dass alle Punkte x der Ebene die Gleichung $w \cdot x + b = 0$ erfüllen. Per Definition sollen die nahesten positiven Beispiele auf der Ebene H_1: $w \cdot x_i + b = 1$ und die nahesten negativen Beispiele auf der Ebene H_2: $w \cdot x_i + b = -1$ liegen (siehe Abbildung 1.17). Dies kann durch eine entsprechende Skalierung von w und b sichergestellt werden. Aus diesen Definitionen lassen sich direkt zwei Nebenbedingungen des Optimierungsproblems formulieren:

$$w \cdot x_i + b \geq 1, \qquad \forall i : y_i = 1 \tag{1.49}$$

$$w \cdot x_i + b \leq -1, \qquad \forall i : y_i = -1 \tag{1.50}$$

[3] Man verwendet als Klassifizierung die -1 und 1 statt z.B. 0 und 1, weil sich damit, wie wir sehen werden, besser rechnen lässt.

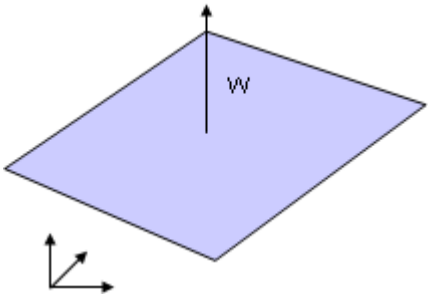


Abbildung 1.16: Hyperebene

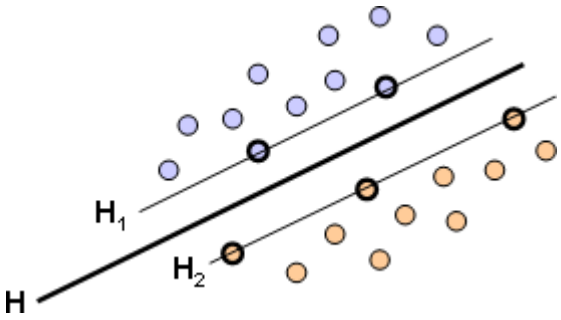


Abbildung 1.17: Breite der Hyperebene

Die lässt sich zusammenfassen zu:

$$y_i(w \cdot x_i + b) - 1 = 0, \qquad \forall i \tag{1.51}$$

Ziel des Optimierungproblems ist die erwähnte Maximierung des Abstandes zu den nahesten Beispielen. Man spricht hier auch von der Breite (*margin*) der Ebene H, die bildlich den Abstand der Ebenen H_1 und H_2 aus Abbildung 1.17 repräsentiert. Dieser Abstand beträgt $\frac{2}{\|w\|}$, wobei $\|w\|$ die euklidische Länge von w darstellt. Zur Maximierung der Breite muss also $\|w\|$ minimiert werden. Eine SVM lässt sich dann durch folgendes Optimierungsproblem beschreiben:

$$\begin{aligned} \|w\|^2 &\longrightarrow Min! \\ y_i(w \cdot x_i + b) - 1 = 0, &\qquad \forall i \end{aligned}$$

Nun wird ersichtlich, warum die SVM *Stützvektor*-Maschine heißt: Die Hyperebene H wird durch Maximierung der Breite bestimmt. Die Breite hängt nur von den zur Ebene am nächsten gelegenen Beispielen ab, die ja die Ebenen H_1 und H_2 bestimmen. Die Bespiele, die auf einer dieser Ebenen liegen, stützen die Ebene H, da sich ohne sie die Lage (die Breite) von H verändern würde, und werden daher Stützvektoren genannt.

Um den Fall nicht nicht linear separierbarer Daten besser handhaben zu können, ist eine Transformation des Problems in die duale Lagrange-Form nötig. Dazu werden die üblichen Lagrange-Multiplikatoren $\alpha_i \geq 0$ eingeführt und die Nebenbedingung in die Zielfunktion aufgenommen. Man erhält zunächst die Zielfunktion:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{m} \alpha_i \tag{1.52}$$

Jedes Optimum der Zielfunktion muss auch ein Minimum der Funktion sein. Dies ist die eine notwendige Bedinung für die Eigenschaft als Optimum. Die Karush-Kuhn-Tucker (KKT) Bedingungen setzten Tatsache in Gleicheungen um. Damit ein Minimum vorliegt, müssen die ersten partiellen Ableitungen in Richtung der Entscheidungsvariablen w und b null sein. Die KKT Bedingungen für das Optimierungsproblem der SVM mit

der Lagrange-Zielfunktion (1.52) lauten:

$$\begin{aligned}
\frac{\partial}{\partial w}L(w,b,\alpha) = w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \\
\frac{\partial}{\partial b}L(w,b,\alpha) = -\sum_{i=1}^{m} \alpha_i y_i = 0 \\
y_i(w \cdot x_i + b) - 1 = 0, \qquad \forall i \\
\alpha_i \geq 0, \qquad \forall i \\
\alpha_i(y_i(w \cdot x_i + b) - 1) = 0, \qquad \forall i
\end{aligned}$$

Durch Umstellen der ersten beiden Bedingungen erhält man:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{1.53}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{1.54}$$

Einsetzen von (1.53) und (1.54) in (1.52) liefert die Zielfunktion L_D des dualen Optimierungsproblems:

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{1.55}$$

Die Zielfunktion hängt damit nur noch von α ab und verwendet als einzige nicht elementare Rechenoperation das Skalarprodukt. Nun erhalten wir das (einfachere) duale Optimierungsproblem für eine SVM auf Basis linear trennbarer Trainingsdaten:

$$\begin{aligned}
\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \longrightarrow Max! \\
\sum_{i=1}^{m} \alpha_i y_i = 0, \qquad \forall i \\
\alpha_i \geq 0, \qquad \forall i
\end{aligned}$$

Nach Lösung des Problems sind alle α_i bestimmt und der Normalenvektor w kann mit (1.53) berechnet werden. Den Bias erhält man, in dem man einen beliebigen Stützvektor ein die Ebenengleichung von H einsetzt und nach b umformt. Normalenvektor, Bias und damit die Hyperebene sind also durch Linearkombinationen aus gerade den Beispielen x_i, für die $\alpha_i > 0$ ist, bestimmt. Diese Beispiele sind, wie wir schon gesehen haben, die Stützvektoren. Also gilt:

- $\alpha_i > 0$: x_i ist Stützvektor
- $\alpha_i = 0$: x_i ist kein Stützvektor

Nun können die Elemente der Test-Menge klassifiziert werden. Dazu genügt die Auswertung der Funktion

$$f(x) = w \cdot x + b \tag{1.56}$$

für ein Testelement x. Das Vorzeichen von $f(x)$ gibt dabei die Klassifizierung an, da nur folgende Fälle zu unterscheiden sind:

- $f(x) > 0$: x leigt im positiven Halbraum
- $f(x) < 0$: x leigt im negativen Halbraum
- $f(x) = 0$: x leigt auf H

Betrachten wir jetzt den Fall nicht linear trennbarer Daten, den Abbildung 1.18 schematisch zeigt. Was kann man tun, um dieses Problem zu lösen?

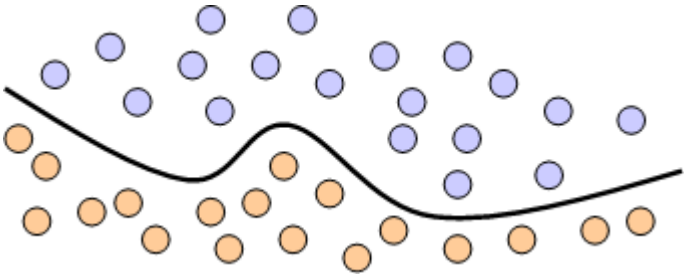


Abbildung 1.18: Nicht linear trennbare Trainingsdaten

- Fehler zulassen
  Dies kann mit der Einführung einer *weich trennenden* Ebene gelöst werden, die fehlerhafte Klassifizierungen von Trainingsbeispielen bis zu einem gewissen Grad zulässt.

- Das Problem und die Daten so transformieren, dass sie linear trennbar werden.

Die eine SVM mit weich trennender Hyperebene erweitert die bekannte SVM um die 'Fehlerkosten' $\xi_i \geq 0$ für jedes Beispiel. Die Nebenbedingungen 1.49 und 1.50 ändern sich dann zu:

$$w \cdot x_i + b \geq 1 - \xi_i, \qquad \forall i : y_i = 1 \tag{1.57}$$

$$w \cdot x_i + b \leq -1 + \xi_i, \qquad \forall i : y_i = -1 \tag{1.58}$$

Die lässt sich wieder zusammenfassen zu:

$$y_i(w \cdot x_i + b) - 1 + \xi_i = 0, \qquad \forall i \tag{1.59}$$

Zu der ursprünglichen Zielfunktion werden die 'Fehlerkosten' einfach hinzu addiert. Weiterhin wird noch eine vom Benutzer zu wählende Konstante C eingeführt, die die Auswirkung der Fehlklassifikationen beeinflusst. Eine weich trenndende SVM wird dann durch folgendes Problem repräsentiert:

$$\begin{aligned} \|w\|^2 + C\sum_{i=1}^{m} \xi_i &\longrightarrow Min! \\ y_i(w \cdot x_i + b) - 1 = 0, &\qquad \forall i \\ 0 \leq \alpha_i \leq C, &\qquad \forall i \end{aligned}$$

Die Zielfunktion des dualen Problem ist wieder (1.52), da die ξ_i beim bilden des dualen Problems verschwinden. Das bekannte duale Problem für linear trennbare Daten wird für die weich trennende SVM also nur um die Nebenbedingungen $0 \leq \alpha_i \leq C$ erweitert. Abbildung 1.19 verdeutlicht noch einmal die Bedeutung der α_i und ξ_i.

Die Transformation der linear nicht trennbaren Daten in linear trennbare gelingt ohne große Mühen, da in den entwickleten Formeln das Skalarprodukt die einzige nicht elementare Rechenoperation ist. Die Idee ist, die Beispiele durch eine Abbildung $\Phi : RR^m \longrightarrow RR^h$ mit $h > m$ in einen höherdimensionalen Raum zu transformieren, wo sie dann linear trennbar sind. Die Zielfunktion (1.55) wird dann einfach angepasst:

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)) \tag{1.60}$$

Sei $K(x,y) = \Phi(x) \cdot \Phi(y)$ eine Kernfunktion, die die das Skalarprodukt zweier transformierter Trainingsdaten direkt berechne. Eine solche Funktion muss die Abbildung Φ garnicht kennen. Abbildung 1.20 verdeutlicht diesen Gedanken, den man auch den Kern-Trick nennt.
Ein Beispiel:
Seien $x, y \in RR^2$ und $K(x,y) = (x \cdot y)^2$ eine Kernfunktion. Um zu zeigen, dass K das Skalarprodukt direkt berechnet, muss eine Abbildung Φ mit $\Phi(x) \cdot \Phi(y) = (x+y)^2$ gefunden werden. Wir nehmen an, dass die Funktion

$$\Phi(x) = \begin{pmatrix} [x]_1^2 \\ \sqrt{2}[x]_1[x]_2 \\ [x]_2^2 \end{pmatrix} \tag{1.61}$$

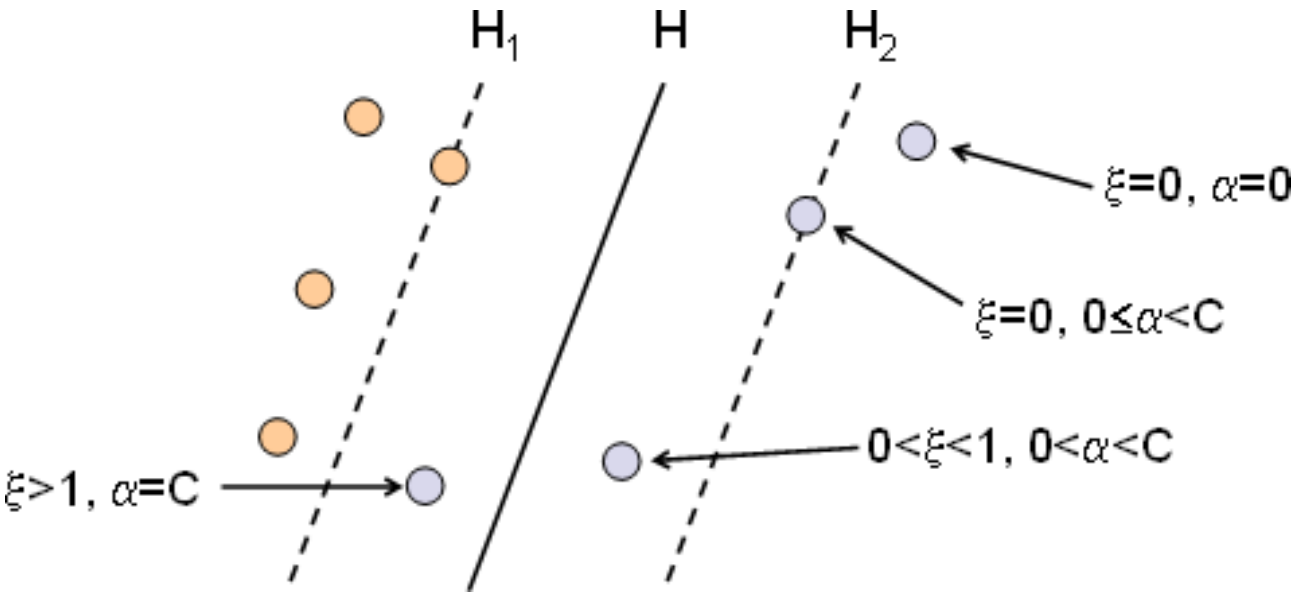


Abbildung 1.19: Beispiele für α-ξ Kombinationen

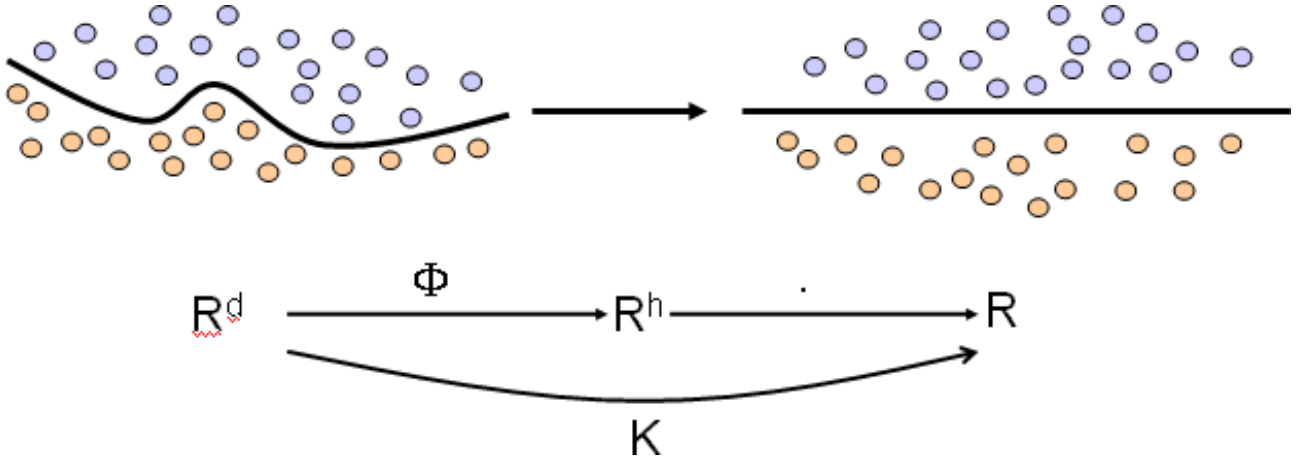


Abbildung 1.20: Der Kern-Trick

dieses tut. $[x]_i$ bezeichnet dabei die i-te Komponente von x. Beweis:

$$\begin{aligned}
\Phi(x)\cdot\Phi(y) &= \begin{pmatrix} [x]_1^2 \\ \sqrt{2}[x]_1[x]_2 \\ [x]_2^2 \end{pmatrix} \cdot \begin{pmatrix} [y]_1^2 \\ \sqrt{2}[y]_1[y]_2 \\ [y]_2^2 \end{pmatrix} \\
&= [x]_1^2[y]_1^2 + 2[x]_1[x]_2[y]_1[y]_2 + [x]_2^2[y]_2^2 \\
&= ([x]_1[y]_1)^2 + 2[x]_1[y]_1[x]_2[y]_2 + ([x]_2[y]_2)^2 \\
&= ([x]_1[y]_1 + [x]_2[y]_2)^2 \\
&= (x\cdot y)^2
\end{aligned}$$

Wir sehen also, dass K das Skalarprodukt von x und y direkt berechnet, ohne dabei erst in den höherdimensionalen Raum abzubilden.

Setzt man die Kernfunktion in in die Zielfunktion (1.60) ein, erhält man zusammen mit den bekannten

Nebenbedingungen das Optimierungsproblem für eine SVM mit nicht linear trennbaren Daten:

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \longrightarrow Max!$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \qquad \forall i$$

$$0 \leq \alpha_i \leq C, \qquad \forall i$$

Mit der *Mercer's Condition* lässt sich überprüfen, ob eine Funktion $K(x,y)$ eine Kernfunktion ist:
Es gibt Abbildungen K und Φ mit $K(x,y) = \Phi(x) \cdot \Phi(y) = \sum_i [\Phi(x)]_i [\Phi(y)]_i \Leftrightarrow$ für jedes $g(x)$ mit finitem $\int g(x)^2 dx$ gilt

$$\int K(x,y) g(x) g(y) dx dy \geq 0$$

Die Lösung des Optimierungsproblems gestaltet sich als schwerig. Zwar liegt ein quadratisches, konvexes Optimierungsproblem vor, welches in $O(m^3)$ gelöst werden kann, jedoch ist dies in praktischen Anwendungen für eine sehr große Menge an Trainingsdaten langwierig. Erschwerend kommt hinzu, dass die Ungleichungen für große m nur mit numerischen Methoden gelöst werden können. Ein numerischer Solver muss als Unterprogramm aufgerufen werden. Im Folgenden werden zwei Lösungsalgorithmen vorgestellt, die das Optimierungsproblem heuristisch lösen.
Ein Verfahren ist das *Chunking*, bei dem der zentrale Gedanke darin liegt, Trainingsbeispiele, die keine Stützvektoren sind, aus der Berechnung auszuschließen, da sie, wir wir gesehen haben, die Lage der Hyperebene nicht beeinflussen. Eine Iteration läuft wie folgt ab:

1. Löse das Problem auf Basis einer Teilmenge der Trainingsdaten
2. Betrachte die berechnten α_i-Werte:
   Ist $\alpha_i = 0$ ist x_i kein Stütuvektor und wird von der weiteren Berechnung ausgeschlossen.
3. Nimm M Trainiungsbeispiele in die Berechnungsmenge auf, die die KTT-Bedingungen am schlechtesten erfüllen.

Über die Iterationen wird also versucht, die Stützvektoren zu identifizieren. In der letzten Iteration sind dann nur noch Stützvektoren in der Berechnungsmenge, so dass dann das gesamte Probmlem gelöst wird. Damit reduziert Chunking den Aufwand auf die quadrierte Anzahl der Stützvektoren. Dies kann erhebliche Vorteile bringen, aber auch genauso schlecht wie die Lösung ohne die Heuristik sein, wenn nahezu alle Trainingsdaten Stützvektoren sind. Definitv nachteilig ist aber, dass Chunking weiterhin nicht ohne Numerik-Solver auskommt. Genau diesem Nachteil nimmt sich das zweite Verfahren an: Die *Sequential Minimal Optimization* ([8]), kurz *SMO*.

Während das Chunking immer mehrere α_i optimiert, beschränkt sich die SMO auf die Optimierung von genau zwei α_i in jeder Iteration. Der Vorteil ist, dass sich dies analytisch lösen lässt. Somit kann auf das Unterprogramm zur numerischen Lösung einer Gleichung verzichtet werden. Insgesamt sind natürlich weit mehr Optimierungen nötig als beim Chunking, da ja alle α_i paarweise optimiert werden müssen. Trotzdem ist die Laufzeit besser als die des Chunking, da die Numerik-Solver Aufrufe schwerer wiegen als das Mehr an Optimierungsiterationen (die alle in $O(1)$ gelöst werden können).
Der SMO-Algorithmus verwendet eine äußere und eine innere Schleife zur Auswahl der α_i-Paare. In der äußeren Schleife werden zunächst alle Beispiele durchlaufen und alle nicht gebundenen (non bound, d.h. α_i ist weder 0 noch C) Beispiele gesucht, die die KKT-Bedingungen verletzten. Das dazugehörige Alpha sei jeweils α_I, welches für die innere Schleife fest ist. Die innere Schleife findet nun unter den übrigen Beispielen ein α_{II} als Optimierungspartner. Als Auswahlkriterium wird der momentane Fehler

$$E_i = f(x_i) - y_i \tag{1.62}$$

genutzt. E_I sei der Fehler des Beispiels aus der äußeren Schleife. Die Fehler der übrigen Beispiele werden berechnet und das Beispiel, welches α_{II} liefert, wie folgt gewählt:

- $E_I > 0$: Wähle Beispiel mit kleinstem E_i
- $E_I < 0$: Wähle Beispiel mit größtem E_i

Für die genaue Vorgehensweise zur Optimierung des Paares (α_I, α_{II}) sei auf [8] verwiesen. Nach der Optimierung wird die Ebene neu berechnet. Damit ändern sich auch die Ergebnisse von $f(x_i)$ bei der Fehlerberechnung aus (1.62). Die äußere Schleife läuft, bis kein Beispiel mehr die KKT-Bedingungen verletzt. Danach werden noch einmal alle Beispiele durchlaufen und paarweise optimiert. Die Suche in der äußeren Schleife beginnt immer an einer Zufallsposition, damit sich die SVM nicht den ersten Trainingsdaten annähert.

SVM struct

Dieser Abschnitt basiert auf dem Artikel, welcher unter [12] zu finden ist.

Einleitung SVMstruct soll auch komplexe Ausgaben miteinbeziehen können. Es soll eine Zuordnung von einer Eingabe $x \in \mathcal{X}$ zu einer diskreten Ausgabe $y \in \mathcal{Y}$ gefunden werden, welche auf einer Trainingsmenge von Eingabe-Ausgabe Paaren $(x_1,y_1),...,(x_n,y_n) \in \mathcal{X} \times \mathcal{Y}$ basiert. $\mathcal{Y}$ ist dabei ein strukturierter Ausgaberaum. $y \in \mathcal{Y}$ können z.B. Sequenzen, Strings, Bäume, Gitter und Graphen sein. Derartige Probleme gibt es bei einer Vielzahl von Anwendungen. Durch eine Verallgemeinerung der Large Margin Methode (Standard Multiclass-SVM) sollen nun strukturierte Antworten gefunden werden.

Diskriminanten und Verlustfunktionen Das Funktionenlernen $f : \mathcal{X} \to \mathcal{Y}$, basiert auf einer Trainingsmenge von Eingabe-Ausgabe Paaren. Es soll eine Diskriminanzfunktion $F : \mathcal{X} \times \mathcal{Y} \to \Re$, über die Eingabe/Ausgabe Paare gelernt werden, von der Vorhersagen abgeleitet werden können, indem F über die Rückgabevariable für die Eingabe x maximiert wird. Formal ist dies wie folgt zu notieren: f(x;w)= $\arg\max_{y \in \mathcal{Y}}$ F(x,y;w), mit w als ein Parameter-Vektor. F soll linear in seiner kombinierten Merkmalsrepräsentation von Ein- und Ausgaben $\Psi(x,y)$ sein, also soll gelten F(x,y;w) = $\langle w, \Psi(x,y) \rangle$. Die spezifische Form von Ψ hängt allerdings von der Art des Problems ab.

Beispiel 1 (natürlichsprachliche Syntaxanalyse) *F soll so gewählt werden, dass ein Modell entsteht, dass einer kontextfreien Grammatik entspricht. Jeder Knoten, im Syntaxbaum y für einen Satz x, entspricht einer Grammatikregel g_j mit den Kosten w_j. Alle gültigen Syntaxbäume y für den Satz x bekommen eine Punktzahl zugewiesen, die sich aus der Summe der w_j seiner Knoten berechnen lässt, also F(x,y;w) = $\langle w, \Psi(x,y) \rangle$. $\Psi(x,y)$ ist ein Histogrammvektor, der die Vorkommen jeder Grammatikregel g_j im Baum y zählt. f(x;w) kann effizient berechnet werden, indem die Struktur $y \in \mathcal{Y}$ gefunden wird, die F(x,y;w) maximiert.*

Das Lernen über strukturiereten Ausgaberäumen $\mathcal{Y}$, bezieht andere Verlustfunktionen mit ein als die Standard 0/1 - Klassifikationsverluste.

Beispiel 2 (natürlichsprachliche Syntaxanalyse) *So ist es sinnvoll einen Syntaxbaum, der sich vom korrekten Syntaxbaum in nur wenigen Knoten unterscheidet, anders zu behandeln, als einen Syntaxbaum der sich von diesem vollkommen unterscheidet.*

Deshalb wird auch davon ausgegangen, dass eine beschränkte Verlustfunktion $\Delta : \mathcal{Y} \times \mathcal{Y} \to \Re$ vorhanden ist. Dies bedeutet, wenn y der wahre Ausgabewert ist, misst $\Delta(y,\hat{y})$ den Verlust der mit der Vorhersage $\hat{y}$ verbunden ist.

Von der SVM zur SVMstruct Zunächst wird der separierbare Fall betrachtet, dass der Trainingsfehler gleich null ist. Angenommen, dass $\Delta(y,y') > 0$ ist, wenn $y \neq y'$, und dass $\Delta(y,y) = 0$ ist. Die Bedingung, dass der Trainingsfehler null ist, soll nun kompakt dargestellt werden, als die Menge der nicht linearen Bedingungen $\forall i : \max_{y \in \mathcal{Y} \setminus y_i} \{\langle w, \Psi(x_i,y) \rangle\} < \langle w, \Psi(x_i,y_i) \rangle$. Jede dieser nicht linearen Ungleichheiten, kann durch $|\mathcal{Y}| - 1$ lineare Ungleichheiten ersetzt werden. Insgesamt gibt es also $n|\mathcal{Y}| - n$ lineare Bedingungen der Form $\forall i, \forall y \in \mathcal{Y} \setminus y_i$: $\langle w, \delta\Psi_i(y) \rangle > 0$, mit $\delta\Psi_i(y) \equiv \Psi(x_i,y_i) - \Psi(x_i,y)$. Wenn die Menge dieser Ungleichheiten zulässig ist, dann gibt es typischerweise mehr als eine Lösung w^*. Um eine einzige Lösung zu erhalten, muss das w gewählt werden, für das sich die Punktzahl des korrekten Labels y_i konstant am meisten von dem nahesten zweitplatzierten unterscheidet, also $\hat{y}_i(w) = \arg\max_{y \neq y_i} \langle w, \Psi(x_i,y) \rangle$. Dies ist die Generalisierung des Maximum-Margin-Prinzips welches in der SVM angewandt wird. Das resultierende Hard-Margin-Optimierungsproblem, ist die SVM für linear trennbare Daten. $SVM^{linear} : \min_w \frac{1}{2}\|w\|^2$, mit den Nebenbedingungen $\forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq 1$. Um nun aber einen Fehler in der Trainingsmenge zu erlauben, wird eine Schlupfvariable eingeführt und zwar für jede nicht lineare Bededingung eine Schlupfvariable. Diese ist die obere Grenze des Trainingsfehlers. Das Einfügen einer Strafbedingung zu den Zielergebnissen die linear in den Schlupfvariablen ist, führt nun zur SVM für nicht lineare Daten. $SVM^{nonlinear} : \min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$, so dass $\forall i, \xi_i \geq 0$, mit den Nebenbedingungen $\forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq 1 - \xi_i$.

Bisher gibt es aber noch keine neuen Erkenntnisse, welche es nicht schon im Kapitel 1.4.1 über SVM gab. Diese Formulierung soll jetzt aber für beliebige Verlustfunktionen Δ generalisiert werden. Dafür muss zunächst den Maßstab der Schlupfvariablen geändert werden und zwar gemäß des Verlustes der in jeder linearen Bedingung angefallen ist. Das Verletzen einer Margin-Bedingung die einen Ausgabewert mit hohem Verlust $\Delta(y_i,y)$ zur Folge hat, soll strenger bestraft werden, als ein Verstoß aus dem ein kleiner Verlust folgt. Dies wird erreicht, indem die Schlupfvariablen mit dem inversen Verlust skaliert werden. Dies führt nun endlich zur SVMstruct, $SVM^{struct} : \min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$, so dass $\forall i, \xi_i \geq 0$, mit den Nebenbedingungen $\forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq 1 - \frac{\xi_i}{\Delta(y_i,y)}$.

SVMstruct Algorithmus Die Hauptaufgabe beim Lösen dieser verallgemeinerten SVM - Formulierung, liegt in der großen Anzahl der Margin-Bedingungen, denn die Gesamtanzahl der Bedingungen beträgt $n|\mathcal{Y}|$. Wobei $|\mathcal{Y}|$ in vielen Fällen extrem groß ist (z.B. beim Grammatik-Lernen). Daher sind Standardlösungen eher ungeeignet. Der SVMstruct-Algorithmus soll nun die Struktur des Maximum-Margin-Problems ausnutzen, also nur noch eine kleine Teilmenge von Bedingungen detailiert betrachten. Er soll eine kleine Menge von aktivierten Bedingungen finden, die eine ausreichend fehlerfreie Lösung garantieren. Dies ergibt eine gute und gültige Lösung, da es immer eine polynomiell große Teilmenge von Bedingungen gibt, so dass die zugehörige Lösung alle Bedingungen mit einer Genauigkeit von mindestens ε erfüllt. Alle anderen Bedingungen werden garantiert durch nicht mehr als ε verletzt, ohne dass sie dafür genauer betrachtet werden müssen. Der folgende Algorithmus kann auf alle SVM^{struct}-Formulierungen angewandt werden, die in [12] vorgestellt werden. Es muss nur die Kostenfunktion in Schritt 5 angepasst werden. Es gibt eine Arbeitsmenge S_i für jedes Trainingsbeispiel (x_i, y_i), welche in Zeile 2

Code 1.4.1 Undercuts-Algorithmus

```
1   Input: (x_1,y_1), ..., (x_n,y_n), C, ε
2   S_i ← ∅ for all i = 1, ..., n
3   repeat
4     for i = 1, ..., n do
5       set up cost function [z.B: SVM^s: H(y) ≡ (1 − ⟨δΨ_i(y), w⟩)Δ(y_i, y)]
6       compute ŷ = arg max_{y∈Y} H(y)
7       compute ξ_i = max{0, max_{y∈S_i} H(y)}
8       if H(ŷ) > ξ_i + ε then
9       S_i ← S_i ∪ {ŷ}
10      α_S ← optimize dual over S, S = ∪_i S_i
11      end if
12    end for
13  until no S_i has changed during iteration
```

definiert wird. Sie hält die aktuell ausgewählten Bedingungen fest. Beim abarbeiten der Trainingsbeispiele wird die Bedingung gesucht, die den größten Verlust zur Folge hat (siehe Zeile 6). Wenn dessen Margin-Überschreitung den in Zeile 7 errechneten Wert von ξ_i, mit mehr als ε überschreitet (diese Abfrage ist in Zeile 8 zu finden), dann soll die Bedingung $\hat{y}$ zur Arbeitsmenge hinzufügt werden (vgl. Zeile 9). Dies entspricht einer sukzessiven Verstärkung des grundlegenden Problems, durch eine Schnittebene, die die aktuelle Lösung von der zulässigen Menge entfernt. Die gewählte Schnittebene entspricht der Bedingung, die den kleinsten zulässigen Wert für ξ_i bestimmt. Nachdem eine Bedingung zur Arbeitsmenge hinzugefügt wurde, muss die Lösung, in Zeile 10 (in Bezug auf S), neu errechnet werden. Der Algorithmus stoppt, wenn, wie in Zeile 13 gefordert, keine Bedingung durch mehr als ε verletzt wird und sich somit kein S_i geändert hat. Es ist leicht zu zeigen, dass der Algorithmus eine Lösung hat die nahe am Optimum liegt. Für eine große Klasse von Problemen konvergiert der Algorithmus in polynomieller Zeit und das trotz eventuell exponentiellen oder unendlichen $|\mathcal{Y}|$. Das liegt daran, dass die Anzahl der Bedingungen in S nicht von $|\mathcal{Y}|$ abhängig ist. Wenn die Zeile 6 des Codes 1.4.1 also polynomiell ist, dann ist der ganze Algorithmus polynomiell.

Anwendungen und Experimente Dieser Ansatz ist effektiv und vielseitig. Er kann auf eine breite Klasse von Aufgaben angewandt werden, z.B. auf Multiclass Klassifikation, Klassifikation mit Taxonomien, Label Sequence Learning, Sequence Alignment und Natural Language Parsing. Um den Algorithmus an das neue Problem anzupassen muss lediglich eine Implementierung der Merkmalsabbildung $\Psi(x,y)$, der Verlustfunktion $\Delta(y_i,y)$ und der Maximierung des Verlusts (Zeile 6 des Codes 1.4.1) erfolgen.

Beispiel 3 (Label Sequence Learning) *Es soll eine Sequenz von Labeln $y = (y^1,...,y^n), y^k \in \Sigma$, gegeben einer Sequenz von Eingaben $x = (x^1,...,x^m)$, vorhergesagt werden. Diese Art der Klassifizierung wird z.B. bei der optischen Zeichenerkennung, der natürlichsprachlichen Syntaxanalyse, der Informationsgewinnung und der Bioinformatik angewandt. Hier wird das Label Sequence Learning am Beispiel NER betrachtet, welches bereits aus Kapitel 1.2 bekannt sein sollte. Es wird eine Sammlung von 300 Sätzen betrachtet. Die Zeichenmenge besteht aus Bezeichnungslosen und zusätzlich aus dem Anfang und der Weiterführung von Personennamen, Organisationen, Orten und diversen Namen. Insgesamt gibt es also 9 verschiedene Labels. $\Psi(x,y)$ ist in diesem Fall ein Histogramm der Zustandsübergänge und eine Menge von Merkmalen die die Ausgaben beschreiben. Tabelle 3 zeigt wie die unterschiedlichen Methoden bei diesem Datensatz abschneiden. Die Standard Hidden Markov Model Methode, welche bereits aus Kapitel 1.2.2 bekannt ist, wird von allen Lernmethoden deutlich übertroffen. Die Methoden Conditional Random Fields (welche bereits aus Kapitel 1.2.4 bekannt ist), Perceptron und SVM,*

Methode	HMM	CRF	Perceptron	SVM
Fehler	9.36	5.17	5.94	5.08

Tabelle 1.4: Methodenvergleich

schneiden fast gleich ab, wobei schon hier die SVM etwas besser ist als die anderen zwei Methoden. Die folgende Tabelle zeigt wie die SVM für nicht lineare Daten gegen die SVMstruct abschneidet. Die Ergebnisse der unter-

Methode	Train Fehler	Test Fehler	Const	Verlust
$SVM^{nonlinear}$	0.2 ± 0.1	5.1 ± 0.6	2824 ± 106	1.02 ± 0.01
SVM^{struct}	0.4 ± 0.4	5.1 ± 0.8	2626 ± 225	1.10 ± 0.08

Tabelle 1.5: Vergleich der SVM Methoden

schiedlichen SVM Methoden sind im großen und ganzen vergleichbar. Der Trainingsfehler ist für die $SVM^{nonlinear}$ etwas besser, der Testfehler hingegen ist nahezu identisch. Weniger Bedingungen werden bei der SVM^{struct} hinzugefügt und der durchschnittliche Verlust ist bei der $SVM^{nonlinear}$ etwas besser. Aber diese Unterschiede sind alle kaum nennenswert.

Ergebnis Als Resultat ist eine SVM für überwachtes Lernen mit strukturierten und voneinander abhängigen Ausgaben entstanden. Diese basiert auf einer Merkmalsabbildung über Eingabe/Ausgabe - Paare und deckt damit eine breite Klasse von Modellen ab. Das wären z.B. gewichtete kontextfreie Grammatiken, Hidden Markov Models und Sequence Alignment, um nur einige wichtige zu nennen. Dieser Ansatz ist flexibel im Umgang mit methodenspezifischen Verlustfunktionen und eine sehr positive Eigenschaft ist, dass nur einem allgemeingültigen Algorithmus an sämtliche Aufgaben herangehen kann. Was die Präzision des Ansatzes angeht, so ist diese für einen weiten Bereich mindestens vergleichbar mit den üblichen Ansätzen und oft sogar besser. Was durch den Algorithmus aber garantiert wird ist, dass dieser nutzbar ist um komplexe Modelle zu trainieren, die ansonsten nur schwer im üblichen Rahmen behandelbar wären.

1.4.2 Clustering

Ensemble Clustering

Das Forschungsgebiet der Cluster Ensembles erarbeitet Verfahren zur Vereinigung mehrerer Objekt-Zuordnungen. Ein Clusterer ordnet jedem Objekt einer Menge maximal eine Gruppe zu. Die Aufgabe dieses Bereichs der Forschung liegt in der Vereinigung der Ergebnisse verschiedener Clusterer zu einer einzelnen Zuordnung, die mit den Originalzuordnungen eine möglichst hohe Ähnlichkeit besitzt. Besondere Bedeutung besitzen Algorithmen, die jene Aufgabe ohne Kenntnis über die Zuordnungskriterien, durchführen, da derartige Verfahren wiederverwendbar und universell einsetzbar sind. Die Qualität der Verfahren kann unter verschiedenen Gesichtspunkten analysiert werden. Ein wesentliches Kriterium liegt trivialerweise in der Ähnlichkeit der neuen Zuordnung zu jenen aus der Eingabe. Wegen eventuell großen Datenmengen muss der Laufzeit in Abhängigkeit von der Eingabemenge ebenfalls eine große Bedeutung zugeordnet werden. Ein weiteres interessantes Kriterium, das nur eine untergeordneter Bedeutung besitzt, liegt in der potentiellen Verwendung dieses Algorithmus auf einem verteilten System. Diese zusätzliche Möglichkeit kann die Laufzeit des Verfahrens allerdings nur um einen konstanten Faktor senken, sollte aber dennoch nicht völlig vernachlässigt werden.

Als Basis für eine effiziente Bearbeitung dient eine Transformation der Eingabedaten in eine geeignete Form für die Verarbeitung. Hierzu werden die Eingabedaten in eine Matrix transformiert, wobei die Zeilen den einzelnen Objekten entsprechen, die zugeordnet werden sollen, und die Spalten den Clusterern, die eine Zuordnung erzeugen. Die Einträge der Matrix entsprechen folglich der Gruppe, die der Clusterer dem Objekt zuordnet. Wegen der Wiederverwendbarkeit des Verfahrens können beliebige Identifikatoren verwendet werden. Die Matrix kann nun durch Erweiterung einer Spalte in einen Hypergraphen umgewandelt werden. Hierzu findet eine Ersetzung eines Clusterers durch die Gruppen, die jener erzeugt, statt. Die Einträge der Matrix werden nun durch eine 1 ersetzt, falls das Objekt jener Gruppe zugeordnet ist, ansonsten durch eine 0. Eine Spalte repräsentiert folglich nun eine Hyperkante in einem Hypergraphen. Jegliche Objekte, die der gleichen Gruppe zugeordnet sind, werden durch Hyperkante verbunden.

Der Cluster-based Similarity Partitioning Algorithm (CSPA)[10] basiert auf einer ziemlich einfachen Idee. Die einzelnen Objekte werden miteinander verglichen und die Ähnlichkeit dieser Objekte ermittelt. Diese paarweise Betrachtung, der einzelnen Objekte kann je nach Anwendungsgebiet auch als Schwachpunkt dieses Verfahrens

angesehen werden. Die Ähnlichkeit wird durch eine Matrixmultiplikation bestimmt. Hierzu wird die Matrix des Hypergraphen mit seiner transponierten Matrix multipliziert. Anschließend wird die Ergebnismatrix mit dem Inversen der Anzahl der Clusterer multipliziert. Jeder Eintrag wird somit auf einen Werte zwischen null und eins normiert, da jedes Objekt nach Definition pro Clusterer nur einer Gruppe zugeordnet werden kann und sich folglich maximal Anzahl der Clusterer Einsen pro Zeile der Originalmatrix bzw. pro Spalte der transponierten Matrix befinden. Nach Ausführung dieser Schritte wurde das Ursprungsproblem auf das Problem eines ähnlichkeitsbasierten Zuordnungsalgorithmus reduziert. Die Matrix bestehend aus den paarweisen Ähnlichkeiten der Objekte kann nun mit Hilfe eines solchen Algorithmus zum Beispiel METIS in die gewünschte Anzahl von Cluster umgruppiert werden.

Der HyperGraph-Partitioning Algorithm (HGPA)[10] versucht die Schwachstelle des Cluster-based Similarity Partitioning Algorithm mit der paarweisen Betrachtung zu beheben. Dieser Algorithmus arbeitet wie schon der Name verrät auf einem Hypergraphen. Eine Spate in der Matrix repräesentiert eine Hyperkante innerhalb dieses Hypergraphen. Aus diesem Graphen sollen möglichst wenig Kanten entfernt werden um die anvisierte Anzahl von unverbundenen Komponenten zu erreichen, wobei jede Kante die gleiche Gewichtung besitzt. Analog zum CSPA wird das Problem auf einen anderen Problemtyp reduziert, für den beispielsweise mit Hilfe des hMETIS-Algorithmus gute Ergebnisse erzielt werden können. Eine weitere Anforderung liegt in der annähernd gleichen Größe der unverbundenen Komponenten, die eine höherer Qualität des Ergebnisses garantieren soll. Eine kleine Schwäche dieses Verfahrens mit der anschließenden Verwendung des hMETIS-Algorithmus ist eine fehlende Berücksichtigung der Schnitthäufigkeit einer zertrennten Kante.

Der Meta-Clustering Algorithm (MCLA)[11] basiert ebenfalls auf der Reduktion des eigentlichen Problems auf einen ähnlichkeitsbasierten Zuordnungsalgorithmus. Hierzu werden für alle Gruppen der Clusterer Knoten erzeugt. Die Kanten zwischen den einzelnen Gruppen werden hierbei mit ein Kantengewicht belegt, das die Ähnlichkeiten dieser Gruppen wiedergibt. Ein etabliertes Verfahren zur Ermittlung der Ähnlichkeit zwischen zwei Gruppen ist das Jaccard Measure. Dieser Maßstab wird durch den Quotienten aus den gemeinsamen Objekten und den Objekten, die mindestens in einer Menge vorkommen, ermittelt. Im Gegensatz zum CSPA, bei dem auf Basis der Ähnlichkeiten der Objekte neue Gruppen erzeugt werden, dient hier die Gruppenähnlichkeit als Eingabe für den Algorithmus. Nachdem die neuen Gruppen, die eine Vereinigung mehrerer ursprünglicher Gruppen sind, erzeugt wurden, wird jedes Objekt einer dieser Gruppe zugeordnet. Das Kriterium für die Zuordnung stellt der Quotient aus Anzahl der Unsprungsgruppen, in denen es vorkommt, und der Anzahl der Gruppen, die vereinigt wurden.

Beim Estimation-Maximization-Algorithm (EM-Algorithm) handelt es sich um einen k-means Algorithmus. Zu Beginn des Algorithmus werden Clusterzentren an beliebigen Position des Arbeitsraumes festgelegt. Die Anzahl der Clusterzentren entspricht der Gruppenanzahl, die erzeugt werden sollen. Jedes Objekt wird mit einer Wahrscheinlichkeit dem Zentrum mit der geringsten Distanz zugeordnet. Als Verfahren für die Ermittlung des Wahrscheinlichkeitwertes wird die Standardnormalverteilung empfohlen. Nachdem diese Zuordnung für alle Objekte vollzogen wurde, können anhand der enthaltenen Objekte und der Wahrscheinlichkeit ihrer Zugehörigkeit neue Clusterzentren ermittelt werden. Mit diesen neuen Clusterzentren werden alle Objekte erneut zugeordnet. Dieser Vorgang wird wiederholt bis ein definierter Schwellenwert unterschritten wurde oder eine definierte maximale Anzahl an Wiederholungen erreicht ist. Diese zusätzliche Bedingung ist notwendig, da dieser Algorithmus ansonsten nicht zwangsläufig terminiert.

Der Cluster-based Similarity Partitioning Algorithm wird besonders bei Großen Objektmengen sehr ineffizient, wenn nicht sogar unpraktikabel. Die Matrixmultiplikation ist zwar ein einfaches Verfahren besitzt jedoch eine Laufzeit von $O(n^2kr)$, wobei n der Anzahl der Objekte, k der Anzahl der Clusterer und r der Gruppen pro Clusterer entspricht. Zudem ist wegen der erzeugten n $\times$ n Matrix eine hohe Speicherbelastung gegeben. Im Gegensatz dazu ist der HyperGraph-Partitioning Algorithm sehr sparsam im Ressourcenverbrauch mit seiner Laufzeit von $O(nkr)$ und dürfte im Rahmen dieser Projektarbeit mit vielen Objekten eine größere Bedeutung besitzen. Der Meta-Clustering Algorithm ist mit seiner Laufzeit von $O(nk^2r^2)$ zwar langsamer als der HGPA, jedoch insbesondere bei wenigen Clusterern und wenig zugeordneten Gruppen interessant. Die Laufzeit des Estimation-Maximization-Algorithm $O(kNH)$ ist anhängig von der Anzahl der erzeugten Gruppen, Objekte und Anzahl der Wiederholungen bis zum erreichen des Schwellenwertes bzw. der maximalen Wiederholungshäufigkeit.

Die Qualität der Ergebnisse differiert in den vorgenommenen Studien stark von dem Eingabedaten. Zumal keinerlei Eingabedaten für ein vergleichbares Forschungsgebiet vorliegen, können hier nur schwer Ergebnisse

prognostiziert werden.

Semi-supervised Clustering

SVM Clustering

1.5 Anwendungsbereich

1.6 Corpus-Erstellung und Beispielmengen

1.7 Systementwurf

Kapitel 2

Informationsextraktion

Kapitel 3

Fragebeantwortung

Kapitel 4

Evaluation

Literaturverzeichnis

[1] D. Appelt and D. Israel. Introduction to information extraction technology. Tutorial for IJCAI-99, Stockholm, August 1999.

[2] Roger Boyle. Hidden-markov-model online tutorium. HMM.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual (web) search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[4] Hamish Cunningham and Kalina Bontcheva. Named entity recognition. Talk at the RANLP, 2003.

[5] Michelangelo Diligenti, Marco Gori, and Marco Maggini. A unified probabilistic framework for web page scoring systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):4–16, January 2004.

[6] Ronen Feldman. Information extraction, theory and practice. ICDM Tutorial, 2003.

[7] Thosten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of Knowledge Discovery in Databases*, 2002.

[8] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12. MIT-Press, 1999.

[9] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[10] Alexander Strehl and Joydeep Ghosh. Cluster ensembles – a knowledge reuse framework for combining partitionings. In Rina Dechter, Michael Kearns, and Rich Sutton, editors, *Proceedings of AAAI 2002, Edmonton, Canada*, pages 93–98, Menlo Park, USA, 2002. American Association for Artificial Intelligence.

[11] A. Topchy, A Jain, and W Punch. A mixture model for clustering ensembles. in: Proc. of siam conference on data mining. 2004.

[12] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[13] C. J. Van-Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd edition, 1979.