Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Kommunikationsforschnung und Phonetik
Prof. Dr. Winfried Lenders

Hauptseminar 'Information Extraction' (3919)

Kern-Methoden zur Extraktion von Informationen

Sebastian Marius Kirsch*
skirsch@moebius.inka.de

Wintersemester 2004/2005

Zusammenfassung

Kern-Methoden sind ein aktuelles Forschungsthema im Bereich der künstlichen Intelligenz und des maschinellen Lernens. Mit diesen Techniken kann man strukturierte Daten wie Bäume oder Graphen, die in der Computerlinguistik häufig vorkommen, mit vertretbarem Aufwand direkt verarbeiten. Gepaart mit speziellen Klassifikations-Algorithmen wie *Support Vector Machine* (SVM) und *Voted Perceptron* können Kern-Methoden reichhaltige syntaktische Strukturen benutzen. Im Gegensatz dazu sind herkömmlichen Methoden der Informations-Extraktion – wie endliche Automaten oder *Hidden Markov Models* – auf Sequenzen beschränkt.

Dieser Text führt in die Grundlagen von Kern-Methoden auf strukturierten Daten ein und stellt eine aktuelle Arbeit im Bereich Informations-Extraktion vor, die mit Kern-Methoden arbeitet.

*Dieser Text ist verfügbar unter http://sites.inka.de/moebius/docs/semie-ho.pdf

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Das Extrahieren von Informationen aus Texten in natürlicher Sprache ist ein recht junges Gebiet der Computerlinguistik. Die Anwendungsgebiete fallen hier in unterschiedliche Kategorien: Teils ist die Aufgabenstellung, Datenbankfelder aus einem natürlichsprachlichen Text zu füllen, teils sollen Texte danach ausgewählt werden, ob sie bestimmte Informationen enthalten. Bei den gesuchten Informationen handelt es sich beispielsweise um Namen von Personen, Firmen oder Orten, um Geldbeträge oder Jahreszahlen, sowie um die Beziehung, in der die genannten Elemente zueinander stehen.

Es zeigt sich, dass eine vollständige syntaktische Analyse und semantische Interpretation zur Lösung dieser Aufgabe oft nicht nötig ist; außerdem ist sie aus Gründen der Laufzeit und der universellen Einsetzbarkeit oft nicht möglich. Stattdessen haben sich auf diesem Gebiet insbesondere endliche Automaten und statistische Methoden bewährt.

Bekannte Systeme auf der Basis von endlichen Automaten sind das FASTUS-System (Appelt u. a., 1993) und das Proteus-System (Yangarber und Grishman, 1998). Beide Systeme benutzen von Hand geschrieben Regeln in Form von regulären Ausdrücken, um Textelemente zu erkennen. Dabei benutzen sie nicht-syntaktische Hinweise wie Groß-/Kleinschreibung oder Wortlisten, um Namen und Anreden zu erkennen. Neuere Systeme benutzen Textkorpora, um Regeln automatisch zu erzeugen; Beispiele hierfür sind das AutoSlog-System (Riloff und Lehnert, 1993; Riloff, 1993) oder der DIPRE-Algorithmus (Brin, 1998).

Eine weitere, rein statistisch fundierte Methode mit sehr guten Ergebnissen sind *Hidden Markov Models* (HMM) (Seymore u. a., 1999). Hidden-Markov-Modelle sind endliche Automaten, deren Zustände den zu extrahierenden Informationen entsprechen. Die Übergänge zu anderen Zuständen sind mit Wahrscheinlichkeiten versehen; außerdem besitzt jeder Zustand Ausgabe-Wahrscheinlichkeiten, mit denen ein bestimmtes Symbol (z. B. ein bestimmtes Wort, oder eine bestimmte Wortklasse) in diesem Zustand produziert wird. In der Trainings-Phase des HMM werden die Übergangs- und Ausgabewahrscheinlichkeiten aus einem annotierten Textkorpus bestimmt. Um Informationen aus einem nicht annotierten Text zu extrahieren, bestimmt man mit Hilfe des HMMs die wahrscheinlichste Sequenz von Zustandsübergängen, die den vorliegenden Text produziert haben könnte.

In den letzten Jahren setzt man verstärkt auf Methoden aus dem Bereich des maschinellen Lernens. Hier werden Klassifikationsalgorithmen benutzt, um zu entscheiden, ob ein Beispiel der gesuchten Information vorliegt oder nicht. Diesen Ansatz bezeichnet man auch als **diskriminativen Ansatz**, da er versucht, zwischen Beispielen mit den gesuchten Informationen und anderen Beispielen zu unterscheiden. Im Gegensatz dazu werden *Hidden Markov Models* als **generative Modelle** bezeichnet, da sie ein

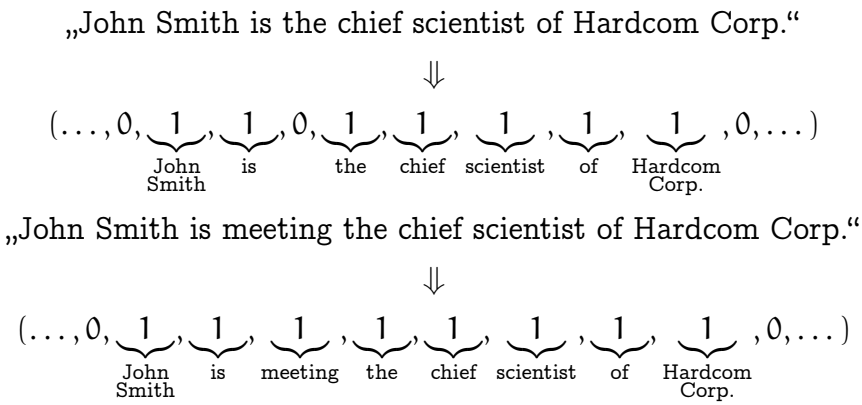


Abbildung 1: Das *bag-of-words*-Modell lässt oft nur unzureichende Rückschlüsse auf den Inhalt eines Textes zu, da grundlegende Änderungen in der syntaktischen Struktur nur geringe Änderungen im zugeordneten Vektor von Worthäufigkeiten nach sich ziehen können.

statistisches Modell für die Erzeugung des Textes benutzen.

Die übliche Repräsentation für natürlichsprachliche Texte im maschinellen Lernen ist das *bag-of-words*-Modell: In diesem Modell werden nur die Worthäufigkeiten der Texte gezählt und miteinander vergleichen. Dieser Ansatz hat sich bei der Kategorisierung von längeren Texten bewährt; bei kurzen Texten oder einzelnen Sätzen versagt er jedoch meistens, da er die syntaktische Struktur nicht berücksichtigt. Kern-Methoden, wie sie in dieser Arbeit vorgestellt werden, erlaube auch die Kategorisierung von einzelnen Sätzen unter Einbeziehung ihrer syntaktischen Struktur.

Dieser Text ist wie folgt gegliedert:

- Abschnitt 2 gibt eine kurze Einführung in die Aufgabenstellung des maschinellen Lernens und stellt zwei aktuelle Algorithmen vor.
- Abschnitt 3 führt den Begriff der Kernfunktion ein und motiviert ihn mit Hilfe von Beispielen.
- Abschnitt 4 stellt Kernfunktionen vor, die direkt auf strukturierten Daten wie Bäumen arbeiten.
- Abschnitt 5 stellt eine Arbeit zur Extraktion von Beziehungen vor; hier soll das System feststellen, ob eine bestimmte Beziehung zwischen Konstituenten eines Satzes vorliegt, z. B. ein Arbeitsverhältnis zwischen einer Person und einer Organisation.

- Abschnitt 6 enthält eine Zusammenfassung und gibt einen Ausblick auf weitere Forschungsthemen im Gebiet der Informations-Extraktion mit Kern-Methoden.

2 Maschinelles Lernen und Kern-Methoden

Die klassische Aufgabe im maschinellen Lernen ist das **Klassifikationsproblem**: Gegeben ist eine Menge von **Beispielen** $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{+1, -1\}$, die jeweils aus einer **Instanz** x aus einem **Instanzenraum** $\mathcal{X}$ und einem **Klassenbezeichner** $y \in \{+1, -1\}$ bestehen.

Die Aufgabe besteht nun darin, eine **Entscheidungsfunktion**

$$f : \mathcal{X} \to \{+1, -1\}, x \mapsto y$$

zu finden, die unbekannte Instanzen korrekt einer Klasse zuweist. In der Wahl der Funktion ist man je nach Algorithmus auf bestimmte Familien von Funktionen beschränkt; so erlauben die im nächsten Abschnitt vorgestellten *Support Vector Machines* Hyperebenen als Funktionen; während andere Algorithmen beispielsweise Entscheidungsbäume produzieren.

Man geht hierbei davon aus, dass die Beispiele einer Wahrscheinlichkeitsverteilung $P(x, y)$ entsprechen. Weiterhin ist eine **Verlustfunktion** $l : \{+1, -1\} \times \{+1, -1\} \to \mathbb{R}$ gegeben, die die Kosten einer Fehlklassifikation angibt. Ein Beispiel für eine Verlustfunktion ist der sog. *zero/one loss* $l(f(x), y) = \Theta(-yf(x))$, wobei Θ die Heaviside-Funktion darstellt ($\Theta(z) = 0$ für $z \leq 0$, $\Theta(z) = 1$ sonst.)

Es existieren verschiedene Kriterien zur Auswahl einer konkreten Funktion aus der gegebenen Funktionsfamilie. Ein Kriterium ist die Minimierung des Risikos:

Als **Risiko** bezeichnet man den erwarteten Verlust einer gewählten Funktion f bzgl. der Verteilung:

$$R(f) = \int l(f(x), y) dP(x, y)$$

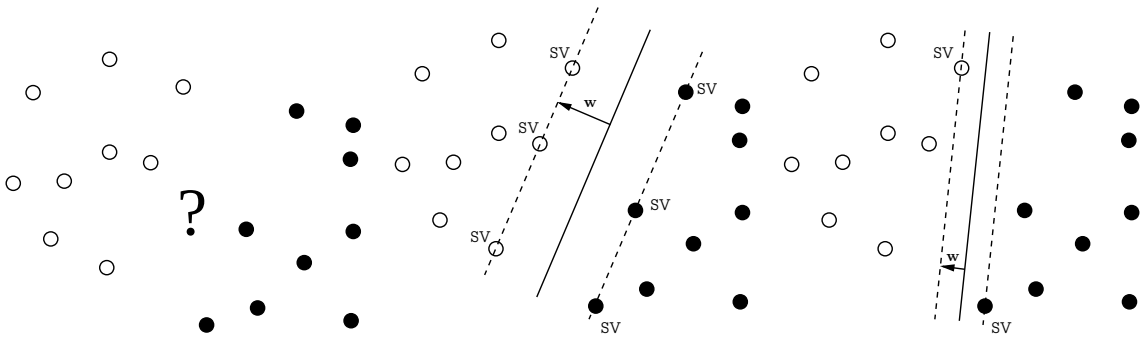
Die bestmögliche Funktion ist diejenige, die das Risiko minimiert:

$$f_{opt.} = \arg\max_f R(f)$$

Da normalerweise weder die Verteilung $P(x, y)$ bekannt ist, noch die Verlustfunktion genau spezifiziert ist, kann das Risiko nur empirisch bestimmt werden. Außerdem ist es bei den meisten Methoden nicht möglich, das Risiko aller möglichen Funktionen zu bestimmen, so dass oft andere Kriterien zur Auswahl benutzt werden.

2.1 Lernalgorithmen

In diesem Abschnitt werden die Prinzipien zweier populärer Lernverfahren vorgestellt.



(a) Aufgabe

(b) Lösung mit breitem Rand

(c) Lösung mit schmalem Rand

Abbildung 2: Aufgabe: Trenne weisse und schwarze Punkte mit Hilfe einer Gerade. Dargestellt sind zwei Lösungen: Eine Lösung mit maximal breitem Rand und eine mit schmalem Rand.

2.1.1 *Support Vector Machines*

Bei **Support Vector Machines** (Cortes und Vapnik, 1995) werden Instanzen betrachtet, die aus einen Vektorraum $\mathbb{R}^N$ stammen. Die Funktion f hat die Form $f(x) = \text{sign}(\mathbf{w} \cdot x + b)$: Eine Hyperebene, die den Raum der Instanzen in zwei Hälften teilt, so dass die Instanzen der beiden Klassen auf verschiedenen Seiten der Hyperebene liegen.

Ist es möglich, eine solche Hyperebene zu finden, so bezeichnet man die gegebene Menge von Instanzen als **linear separierbar**.

Theoretische Überlegungen deuten darauf hin, dass die Breite des Randes (engl. *margin*), sprich der minimale Abstand einer Instanz zur trennenden Hyperebene, ein Mass für die Güte der gefundenen Funktion darstellt: Je breiter der Rand, desto besser generalisiert die gefundene Funktion auf zukünftigen Lösungen. In Abbildung 2 sind zwei Lösungen dargestellt, eine mit schmalem sowie eine mit breitem Rand. Aus dieser Abbildung ist auch intuitiv ersichtlich, warum ein breiter Rand eine bessere Lösung darstellt.

Das Auffinden einer Lösung mit maximal breitem Rand führt auf ein quadratisches Optimierungsproblem:

$$\max_{(\alpha_1,\ldots,\alpha_n)} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{1}$$

$$\text{so dass } \alpha_i \geq 0 \text{ für } i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3}$$

Die Zahlen α_i stellen hierbei Lagrange-Multiplikatoren dar. Der Vektor $\mathbf{w}$ wird in

dieser Formulierung als Kombination aus den Lernbeispielen dargestellt, so dass sich eine Funktion der Form

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (x \cdot x_i) + b\right) \tag{4}$$

ergibt. Dieses Problem lässt sich mit Techniken der Quadratischen Programmierung lösen. (Für eine Herleitung dieser Form sei auf (Müller u. a., 2001) verwiesen; ebenso für eine Formulierung im Fall, dass das Problem nicht linear separierbar ist, sprich die beiden Klassen sich nicht durch eine Hyperebene trennen lassen.)

Diejenigen Beispiele, die direkt auf dem Rand liegen, werden als *support vectors* bezeichnet und sind in Abbildung 2 mit „SV“ beschriftet. Der Name *Support Vector Machine* für dieses Lernverfahren ergibt sich aus folgender Tatsache: Die Zahlen $\alpha_i, \ldots, \alpha_n$ in der obigen Formulierung haben nur für diejenigen Beispiele einen Wert ungleich Null, die direkt auf dem Rand liegen. Deshalb genügt es zur Berechnung der Entscheidungsfunktion, nur die *support vectors* zu betrachten – alle anderen Beispiele haben keinen Einfluss auf den Wert der Entscheidungsfunktion.

2.1.2 *Voted Perceptron*

Ein zweiter Algorithmus mit ähnlichen Charakteristiken wie der SVM-Algorithmus ist das ***Voted Perceptron***, vorgestellt von Freund und Schapire (1999). Er basiert auf dem Perceptron-Algorithmus (Rosenblatt, 1988), der Entscheidungsfunktionen der Form $f(x) = \text{sign}(\mathbf{w} \cdot x)$ benutzt.

Im Vergleich zum SVM-Algorithmus, für dessen Lösung quadratische Programmierung notwendig ist, lässt sich der *Voted Perceptron*-Algorithmus sehr einfach implementieren; der komplette Algorithmus ist in Abbildung 3 zu finden.[1]

Beim Vergleich zum ebenfalls abgedruckten Perceptron-Algorithmus fällt der Unterschied zwischen beiden Algorithmen direkt ins Auge: Während der Perceptron-Algorithmus nur einen Vektor $\mathbf{w}$ ausgibt, produziert *Voted Perceptron* eine Liste von Vektoren $\mathbf{w}_1, \ldots, \mathbf{w}_k$ mit zugehörigen Gewichten $c_1, \ldots, c_k$. Dabei gibt das Gewicht an, wie lange ein Vektor „überlebt“ hat, bevor er ein Beispiel falsch klassifizierte. Die Vorhersage wird als gewichtete Mehrheitsentscheidung mit allen Gewichtsvektoren getroffen – auf diese Weise werden Vektoren bevorzugt, die besonders lange „überlebt“ haben.

[1] Der abgedruckte Algorithmus entspricht der Formulierung in (Freund und Schapire, 1999); er unterscheidet sich vom klassischen Perceptron-Algorithmus insofern, dass er einen Schwellwert von Null für die Sprungfunktion benutzt. Die einzige mir verfügbare Implentierung von *Voted Perceptron* (im Softwarepaket Weka, `http://www.cs.waikato.ac.nz/~ml/weka/`) benutzt implizit einen Schwellwert ungleich Null, ohne dies näher zu erläutern.

Perceptron

Daten : $(x_1, y_1), \ldots, (x_n, y_n)$
T
Ergebnis : $\mathbf{w}$
$\mathbf{w} := 0$
$t := 1$
wiederhole
  für $i := 1, \ldots, n$ **tue**
    $f(x_i) := \text{sign}\,(\mathbf{w}_k \cdot x_i)$
    wenn $f(x_i) \neq y_i$ **dann**
      $\mathbf{w} := \mathbf{w} + y_i x_i$
  $t := t + 1$
bis $t = T$

Vorhersage:

$f(x) := \text{sign}\ (\mathbf{w} \cdot x)$

Voted Perceptron

Daten : $(x_1, y_1), \ldots, (x_n, y_n)$
T
Ergebnis : $(\mathbf{w}_1, c_1), \ldots, (\mathbf{w}_k, c_k)$
$k := 0$
$\mathbf{w}_0 := 0$
$c_0 := 1$
$t := 1$
wiederhole
  für $i := 1, \ldots, n$ **tue**
    $f(x_i) := \text{sign}\,(\mathbf{w}_k \cdot x_i)$
    wenn $f(x_i) = y_i$ **dann**
      $c_k := c_k + 1$
    sonst
      $\mathbf{w}_{k+1} := \mathbf{w}_k + y_i x_i$
      $c_{k+1} := 1$
      $k := k + 1$
  $t := t + 1$
bis $t = T$

Vorhersage:

$f(x) := \text{sign}\ \left(\sum_{i=1}^{k} c_i \,\text{sign}\ (\mathbf{w}_i \cdot x)\right)$

Abbildung 3 : Der Lernalgorithmus *Voted Perceptron* im Vergleich zum klassischen Perceptron-Algorithmus.

Da sich die Gewichtsvektoren im Perceptron-Algorithmus als eine Summe der Lernbeispiele ergeben, lässt sich der Gewichtsvektor auch in der Form

$$\mathbf{w}_k = \sum_{j=1}^{k} y_{l_j} x_{l_j}$$

schreiben, wobei die Indizes $l_j, j = 1, \ldots, k$ die Indizes der im Laufe des Algorithmus falsch klassifizierten Beispiele darstellen. Die Entscheidungsfunktion ergibt sich damit als

$$f(x) = \text{sign}\left(\sum_{i=1}^{k} c_i \,\text{sign}\,(\mathbf{w}_i \cdot x)\right) \tag{5}$$

$$= \text{sign}\left(\sum_{i=1}^{k} c_i \,\text{sign}\left(\sum_{j=1}^{i} y_{l_j}(x_{l_j} \cdot x)\right)\right) \tag{6}$$

Für die Zahl der Fehlklassfikationen eines *Voted Perceptron* ergeben sich eine ähnliche Abschätzung wie für die SVM, die wiederum von der Breite des Rands abhängt. Aus diesem Grund bezeichnet man beide Algorithmen auch als *Large Margin*-Klassifikatoren.

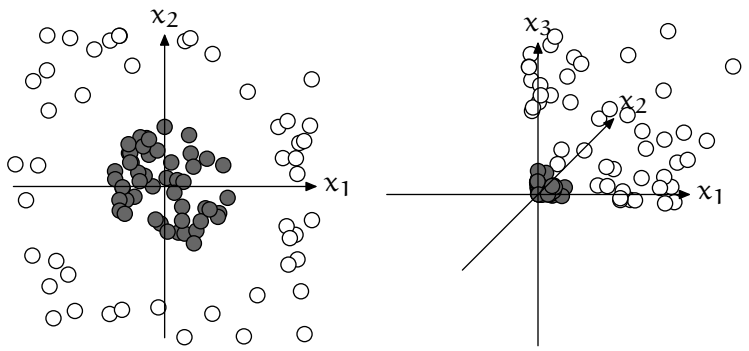
3 Lernen mit Kernfunktionen

Praxiserfahrungen im maschinellen Lernen haben gezeigt, dass es oft sinnvoll ist, Lernbeispiele vorzuverarbeiten und dabei Kombinationen aus einzelnen Merkmalen als neue Merkmale hinzuzufügen. Man bildet hierbei aus Merkmal A und Merkmal B ein neues Merkmal, das aussagt, ob Merkmal A und Merkmal B gemeinsam auftreten, oder ob eines der beiden Merkmale auftritt. Auf diese Weise lassen sich oft bessere Ergebnisse in der Klassifikation erreichen. Werden Merkmale als Zahlen kodiert, lassen sich neue Merkmale beispielsweise als Produkt von existierenden Merkmalen erzeugen.

Ein einfaches Beispiel für diese Tatsache ist in Abbildung 4 zu finden. In diesem Beispiele werden Instanzen durch Punkte in der Ebene dargestellt, eine Instanz hat also die Form $x = (x_1, x_2)$. Wie in der Abbildung ersichtlich, liegen alle Punkte der Klasse $+1$ in einem Kreis um den Ursprung.

Lernalgorithmen wie SVM oder *Voted Perceptron* versuchen in diesem Fall, die Instanzen mit Hilfe einer Gerade in zwei Klassen einzuteilen; im Beispiel kann dies offensichtlich nicht funktionieren: Es gibt keine Gerade, mit der man die Instanzen gemäss ihrer Klasse trennen kann.

Erzeugt man aus einem Beispiel $x = (x_1, x_2)$ ein neues Beispiel $x' = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$, so lassen sich die resultierenden Beispiele im dreidimensionalen Raum durch eine Ebene trennen. Eine korrekte Lösung lässt sich also erreichen, indem man alle Beispiele vor



(a) Ohne Kernfunktion (b) Mit Kernfunktion

Abbildung 4: Eine Kernfunktion kann zu linearer Separierbarkeit im Merkmalsraum führen.

Anwendung des Algorithmus in die angegebene Form umrechnet. Diese Umrechnung stellen wir auch als Abbildung dar:

$$\phi(x) : \mathbb{R}^2 \to \mathbb{R}^3, (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

und setzen $x' = \phi(x)$.

Betrachtet man das Skalarprodukt zwischen zwei Instanzen a und b, wie es in den Lernalgorithmen benötigt wird, so lässt sich leicht folgende Tatsache nachrechnen:

$$\begin{aligned}\phi(a) \cdot \phi(b) &= (a_1^2, \sqrt{2}a_1a_2, a_2^2) \cdot (b_1^2, \sqrt{2}b_1b_2, b_2^2) \\ &= a_1^2b_1^2 + 2a_1a_2b_1b_2 + a_2^2b_2^2 \\ &= (a_1b_1 + a_2b_2)^2 \\ &= (a \cdot b)^2\end{aligned}$$

Benutzt man nur Skalarprodukte, ist es also nicht nötig, die Instanzen $\phi(a)$ und $\phi(b)$ explizit zu berechnen; es genügt, das Skalarprodukt zwischen a und b zu quadrieren.

Damit stellt sich die Frage, welche anderen Funktionen sich als Skalarprodukt zwischen zwei transformierten Instanzen $\phi(a)$ und $\phi(b)$ interpretieren lassen – denn in diesem Fall kann man sich die Berechnung der Transformation mit anschliessendem Skalarprodukt ersparen und berechnet stattdessen einfach die passende Funktion. Diese Funktion kann – wie im Beispiel – wesentlich einfacher zu berechnen sein als die Abbildung mit anschliessendem Skalarprodukt.

Das Theorem über Kernfunktionen von Mercer bietet eine Antwort auf diese Frage: Eine symmetrische Funktion $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ ist eine **positiv-definite Kernfunktion** auf $\mathcal{X}$, wenn für alle $n \in \mathbb{Z}^+$, alle $x_1, \ldots, x_n \in \mathcal{X}$ und alle $c_1, \ldots, c_n \in \mathbb{R}$ gilt, dass

$$\sum_{i,j=1,\ldots,n} c_ic_jk(x_i, x_j) \geq 0$$

Mercers Theorem garantiert, dass für jede positiv-definite Kernfunktion eine Abbildung $\phi : \mathcal{X} \to \mathcal{H}$ in einen euklidischen Raum $\mathcal{H}$ existiert, so dass $k(x, x') = \phi(x) \cdot \phi(x')$, wobei $\cdot$ für das **Skalarprodukt** (oder **inneres Produkt**) im Raum $\mathcal{H}$ steht. Den Raum $\mathcal{H}$ bezeichnet man auch oft als **Merkmalsraum**.

Die Abbildung ϕ muss hier zu keiner Zeit explizit berechnet werden. Ihre genaue Form muss nicht einmal bekannt sein; Mercers Theorem garantiert nur, dass sie existiert, wenn die Funktion k die passenden Eigenschaften hat. Im Beispiel haben wir eine explizite Abbildung von der zweidimensionalen Ebene in den dreidimensionalen Raum betrachtet und gezeigt, dass sich das Skalarprodukt auch direkt, ohne Kenntnis der Abbildung berechnen lässt. Bei anderen Kernfunktionen kann es sein, dass die explizite Abbildung nicht berechenbar ist, da sie beispielsweise in einem Raum mit unendlich vielen Dimensionen (einen Hilbertraum) abbildet.

Die Lernalgorithmen in Formel 1 und die Entscheidungsfunktionen in Formel 4 bzw. 5, sind mit Absicht so formuliert, dass als einzige Operation auf Instanzen das Skalarprodukt benutzt wird. Durch Einsetzen einer Kernfunktion anstelle des Skalarprodukts lassen sich diese Algorithmen in einem Merkmalsraum anwenden, *ohne* die Abbildung in den Merkmalsraum jemals explizit zu berechnen.

Das Skalarprodukt in einem Vektorraum bezeichnet man im Umfeld von Kernmethoden auch als **lineare Kernfunktion** $k_{\text{lin.}}(x, x') = x \cdot x'$. Kernfunktionen lassen sich kombinieren, um neue Kernfunktionen zu erzeugen: Sind f und g zwei Kernfunktionen, so sind auch $f(x, x') + g(x, x')$ und $f(x, x')g(x, x')$ Kernfunktionen. Andere beliebte Kernfunktionen sind in Tabelle 1 dargestellt.

Die Auswahl einer passenden Kernfunktion, durch die das gegebene Problem linear separierbar wird, ist nicht offensichtlich. In der Tat benötigt man oft viele Tests, bis man eine passende Kernfunktion mit passenden Parametern gefunden hat. Ein anderer Ansatz wird im nächsten Abschnitt vorgestellt.

4 Kernfunktionen auf strukturierten Daten

Bisher haben wir nur die lineare Kernfunktion betrachtet, die auf Vektoren definiert ist, sowie Kernfunktionen, die durch Kombination von anderen Kernfunktionen entstehen. Mercers Theorem macht jedoch keine Aussage über die Natur des zugrundliegenden

Tabelle 1: Kernfunktionen

polynomiell	$(k(x, x') + l)^d$
sigmoid	$\tanh(k(x, x') + c)$
radial basis function	$\exp(-\gamma\|x - x'\|^2) = \exp(-\gamma(k(x, x) - k(x, x') + k(x, x')))$
normalisiert	$\frac{k(x,x')}{\sqrt{k(x,x)k(x',x')}}$

Raumes; eine Kernfunktion kann auf beliebigen Datentypen definiert werden – zum Beispiel auch auf Graphen oder Bäumen. Einzige Bedingung ist, dass die Kernfunktion symmetrisch und positiv-definit ist. Für das maschinelle Lernen ist es außerdem notwendig, dass die Kernfunktion auf eine gewisse Weise die Ähnlichkeit zwischen den Instanzen ausdrückt; die Kernfunktion dient als eine Art verallgemeinertes Abstandsmass zwischen den Instanzen.

Mit solchen Kernfunktionen können Datentypen direkt mit maschinellen Lernverfahren verarbeitet werden, die bis jetzt für diese Anwendung nicht zugänglich waren. Gerade für die Computerlinguistik ist dies interessant, da in diesem Bereich eher mit Bäumen, Graphen und Symbolketten gearbeitet wird als mit Vektoren.

Die Herausforderung besteht nun darin, passende Kernfunktionen zu finden.

Eine Reihe von Arbeiten der letzten Jahre haben sich mit diesem Thema befasst und Kernfunktionen auf Zeichenketten (Lodhi u. a., 2002), Mengen von Objekten (Kondor und Jebara, 2003) oder Graphen (Horváth u. a., 2004) vorgestellt; diese Kernfunktionen finden Anwendung beispielsweise in der Analyse von Genen oder von medizinischen Wirkstoffen. Ein Überblick über Kernfunktionen auf strukturierten Daten ist in (Gärtner, 2003) zu finden.

Für die Computerlinguistik besonders interessant ist eine Klasse von Kernfunktionen, die auf Bäumen arbeitet. Sie wurde von Collins und Duffy (2001) vorgestellt und gehört zu den **Faltungs-Kernfunktionen** (*convolution kernels*). Die allgemeine Form der Kernfunktion auf Bäumen wird nun kurz vorgestellt; die in Kapitel 5.1 benutzte Kernfunktion baut hierauf auf.

Ein **Baum** x wird im Folgenden durch seine **Knoten** $v_1, \ldots, v_d \in V[x]$ charakterisiert. Die Zahl der **Kinder** eines Knoten v ist $\delta^+(v)$, und $\text{child}(v, i)$ bezeichnet den i-ten Kindknoten für $i = 1, \ldots, \delta^+(v)$. Jedem Knoten ist außerdem ein **Bezeichner** $\text{label}(v)$ zugeordnet – dieser Bezeichner mag beispielsweise für die Kategorien (P, NP, VP, etc.) eines Syntaxbaums stehen.

Damit ergibt sich eine Kernfunktion zwischen zwei Bäumen als

$$k_{\text{tree}}(x, x') = \sum_{v \in V[x], v' \in V[x']} C(v, v') \tag{7}$$

wobei die Funktion C wie folgt definiert ist:

- $C(v, v') = 0$, falls $\text{label}(v) \neq \text{label}(v')$, also die beiden Knoten verschiedenen Kategorien angehören.
- $C(v, v') = 1$, falls $\text{label}(v) \neq \text{label}(v')$ und $\delta^+(v) = \delta^+(v') = 0$, also beide Knoten der selben Kategorie angehören und Terminale sind.
- Handelt es sich bei v und v' um Knoten der gleichen Kategorie, so ist C rekursiv

definiert:

$$C(v, v') = \prod_{i=1}^{\delta^+(v)} (1 + C(\text{child}(v, i), \text{child}(v', i)))$$

Man kann sich leicht davon überzeugen, dass es sich bei k_{tree} um eine Kernfunktion handelt, wenn man sie als Skalarprodukt in einem Merkmalsraum interpretiert: Die Bäume, die sich über einer endlichen Menge von Bezeichnern erzeugen lassen, ist abzählbar, sprich, es ist möglich, jedem Baum eine Zahl i zuzuordnen – auch wenn sich natürlich unendlich viele solcher Bäume erzeugen lassen. Für einen Baum x konstruiert man nun eine Abbildung $\phi : \mathcal{X} \to \mathcal{H}$ so, dass die i-te Komponente von $\phi(x)$ angibt, wie oft der Baum mit Nummer i im Baum x als Unterbaum vorkommt. Die Kernfunktion k_{tree} berechnet das Skalarprodukt zweier Instanzen unter der Abbildung ϕ: $k_{tree}(x, x') = \phi(x) \cdot \phi(x')$.

Eine ähnliche Kernfunktion wird in Abschnitt 5 benutzt, um Beispiele von Beziehungen zu unterscheiden.

5 Extraktion von Beziehungen mit Kernmethoden

Dieser Abschnitt stützt sich auf die Arbeiten von Dmitry Zelenko zur Extraktion von Beziehungen. Zelenko bearbeitete dieses Thema im Rahmen seiner Doktorarbeit „Machine Learning for Information Extraction“ (Zelenko, 2003); eine Zusammenfassung wurde in (Zelenko u. a., 2003) publiziert. Zelenko und seine Mitautoren arbeiten für SRA International, einer Firma für Information Retrieval und Sprachtechnologie.

Die Aufgabenstellung bei der Extraktion von Beziehungen besteht darin, zu entscheiden, ob die Elemente eines Satzes in einer bestimmten Beziehung zueinander stehen – ob beispielsweise eine im Satz erwähnte Person Mitglied einer ebenfalls erwähnten Organisation ist, oder ob eine Organisation ihren Sitz an einem erwähnten Ort hat. Dabei soll die Entscheidungsfunktion aus Beispielen gelernt werden.

Als Beispiele dienen eine grossen Anzahl von Sätzen, in denen Personen, Organisationen und Orte vorkommen; für jeden dieser Sätze hat ein menschlicher Bearbeiter entschieden, ob die Satzbestandteile in einer der gesuchten Beziehungen zueinander stehen.

Die Autoren benutzen eine Reihe von Vorverarbeitungsschritten. Sie setzen dabei Komponenten des von SRA entwickelte System REES (Aone und Ramos-Santacruz, 2000) ein, um flache Syntaxbäume der Sätze zu erzeugen. Das Erzeugen von **flachen Syntaxbäumen** (engl. *shallow parsing*) ist eine Technik, Konstituenten eines Satzes (wie Nominalphrasen und Verbalphrasen) zu bestimmen, wobei jedoch darauf verzichtet wird, einen kompletten Syntaxbaum zu erstellen. Dieses Verfahren ist wesentlich robuster als das Erstellen kompletter Syntaxbäume (engl. *deep parsing*), lässt jedoch

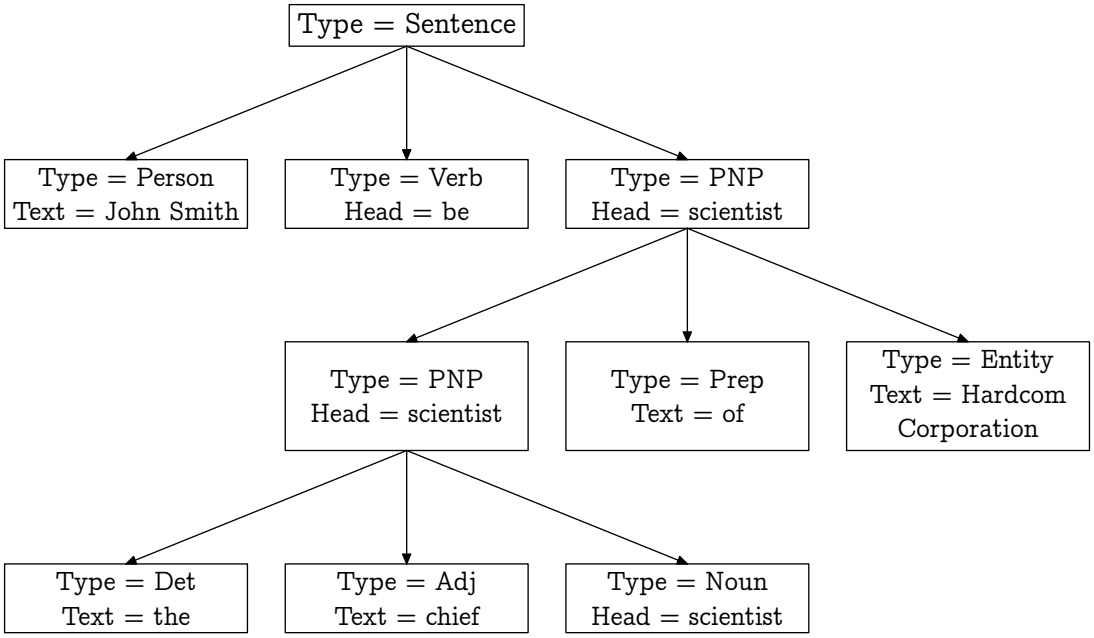


Abbildung 5: Der flache Syntaxbaum für den Satz „John Smith is the chief scientist of the Hardcom Corporation.“

weniger Rückschlüsse auf die Struktur des Satzes zu. Ein Beispiel für einen flachen Syntaxbaum ist in Abbildung 5 dargestellt.

Die Autoren befassen sich mit der Extraktion von zwei Typen von Beziehungen: Der Beziehung zwischen einer Organisation und einem Mitglied (*member–affiliation*), und zwischen einer Organisation und dem Ort, an dem sie angesiedelt ist (*organization–location*).

5.1 Kernfunktionen auf flachen Syntaxbäumen

Die Kernfunktion auf flachen Syntaxbäumen lehnt sich an die in Abschnitt 4 vorgestellte Kernfunktion für Bäume an. Die Eigenschaften von flachen Syntaxbäumen und die gegebene Aufgabe erfordern jedoch einige Anpassungen.

Die Faltungs-Kernfunktion 7 betrachtet nur die Kategorie der Knoten, nicht jedoch das Wort, das ihnen zugeordnet ist. Meistens weisen jedoch bestimmte Worte darauf hin, dass Bestandteile eines Satzes in Beziehung zueinander stehen: „is chairman of“ deutet beispielsweise auf die Beziehung *member–affiliation* hin. Aus diesem Grund wird jedem Knoten v in einem flachen Syntaxbaum neben seinem **Typ** $\mathrm{type}(v)$ auch das entsprechende Wort $\mathrm{text}(v)$ zugeordnet.

Außerdem wird jedem Knoten eine **Rolle** $\mathrm{role}(v)$ zugeordnet: Sie legt fest, welche Rolle der Knoten in einer Beziehung spielen kann. Rollen wie *member*, *organization* oder *location* lassen sich mit *named entity recognition* feststellen; allen anderen

Knoten wird die Rolle „None“ zugeordnet.

Die Idee hinter der Kernfunktion unterscheidet sich leicht von der Faltungs-Kernfunktion auf Bäumen: Statt identischen Unterbäumen sollen nun gemeinsame Teilbäume gezählt werden, die identische Worte enthalten und an den passenden Stellen bezüglich der Rolle übereinstimmen. Auf diese Weise bestimmt man Satzbestandteile, die auf das Auftreten einer Beziehung hindeuten, auch wenn die Rollen der Beziehung verschieden gefüllt werden.

Dabei müssen diese Teilbäume nicht an einem Stück vorkommen; sie können auch an verschiedenen Stellen des flachen Syntaxbaums auftreten – beispielsweise getrennt durch einen eingeschobenen Nebensatz. Sind die Teilstücke weit voneinander getrennt, so soll ihr Auftreten allerdings als geringere Ähnlichkeit gewertet werden.

Diese Überlegungen motivieren folgende Kernfunktion:

Auf den Knoten werden zunächst eine **Übereinstimmung** $\text{match} : V \times V \to \{0, 1\}$ und eine **Ähnlichkeit** $\text{sim} : V \times V \to \{0, 1\}$ definiert. $\text{match}(v, v')$ gibt an, ob zwei Knoten bezüglich Typ und Rolle übereinstimmen, während $\text{sim}(v, v')$ zwei Knoten bezüglich des ihnen zugeordneten Wortes vergleicht:

$$\text{match}(v, v') = \begin{cases} 1 & \text{wenn } \text{type}(v) = \text{type}(v') \text{ und } \text{role}(v) = \text{role}(v') \\ 0 & \text{sonst} \end{cases}$$

$$\text{sim}(v, v') = \begin{cases} 1 & \text{wenn } \text{text}(v) = \text{text}(v') \\ 0 & \text{sonst} \end{cases}$$

Die Funktion $k_\nu(v, v')$ ist ebenfalls rekursiv definiert; wie oben sind hier mehrere Fälle zu unterscheiden:

- $k_\nu(v, v') = 0$, wenn die Knoten v und v' nicht übereinstimmen, also $\text{match}(v, v') = 0$.
- Stimmen beide Knoten überein, so ist

  $$k_\nu(v, v') = \text{sim}(v, v') + C(v, v')$$

  Die Ähnlichkeit $C(v, v')$ ist wiederum rekursiv definiert:

  Man betrachtet nun alle möglichen Sequenzen gleicher Länge von Kindknoten von v und v'. Hierzu definiert man Teilmengen $S \subseteq \{1, \ldots, \delta^+(v)\}$ und $S' \subseteq \{1, \ldots, \delta^+(v')\}$ über die Indizes der Kindknoten, so dass $|S| = |S'|$. Der Abstand zwischen kleinstem und größtem Element einer solchen Sequenz wird mit $\text{dist}(S)$ bezeichnet und ist als $\text{dist}(S) = \max(S) - \min(S) + 1$ definiert.

Zur Definition von C summiert man über alle möglichen Sequenzen von Kindknoten:

$$C(v, v') = \sum_{\substack{S \subseteq \{1,\ldots,\delta^+(v)\} \\ S' \subseteq \{1,\ldots,\delta^+(v')\} \\ |S|=|S'|}} \lambda^{\mathrm{dist}(S)} \lambda^{\mathrm{dist}(S')} k_S(S, S')$$

Der Parameter λ dient dazu, Sequenzen abzuwerten, deren Mitglieder weiter auseinander liegen; er wird zwischen 0 und 1 gewählt.

k_S bezeichnet dabei die Ähnlichkeit zwischen zwei Sequenzen von Kindknoten zweier Knoten:

- $k_S(S, S') = 0$, falls die Typen der Kindknoten in den Sequenzen nicht übereinstimmen, wenn also $\mathrm{match}(\mathrm{child}(v, S_i), \mathrm{child}(v', S'_i)) = 0$ für mindestens ein $i = 1, \ldots, |S|$.
- Sonst summiert man k_v über die Kindknoten:

$$k_S(S, S') = \sum_{i=1}^{|S|} k_v(\mathrm{child}(v, S_i), \mathrm{child}(v', S'_i))$$

Die so definierte Funktion k_v ist eine Kernfunktion auf flachen Syntaxbäumen; sie lässt sich in den in Abschnitt 3 vorgestellten Algorithmen benutzen, um diese auf flache Syntaxbäume anzuwenden. Hierzu übergibt man den Wurzelknoten der zu vergleichenden Syntaxbäume an k_v.

Die vorgestellte Kernfunktion lässt sich effizient berechnen; die Autoren geben Algorithmus mit einer Laufzeit von $O(mn)$ für den Fall an, dass die Sequenzen keine Lücken aufweisen, und $O(mn^3)$ für Sequenzen mit Lücken. m und n bezeichnen hier die Zahl der Kindknoten zweier Knoten v und v', wobei angenommen wird, dass $m \geq n$.

5.2 Experimentelle Ergebnisse

Die Experimente in Zelenko u. a. (2003) wurden auf einem nicht weiter spezifizierten Korpus von 200 Nachrichtenbeiträgen aus verschiedenen Quellen durchgeführt. Nach der Generierung der flachen Syntaxbäume mit einem bestehenden System wurden Teilbäume ausgeschnitten, die potentiell eine Beziehung zwischen den Konstituenten enthalten. Aus dem flachen Syntaxbaum in Abbildung 5 können beispielsweise die drei Beispiele in Abbildung 6 für die Beziehung *member–affiliation* ausgeschnitten werden. Ebenso wurden Beispiele für die Beziehung *organization–location* generiert.

Machte das System gravierende Fehler bei der Erstellung des flachen Syntaxbaums, so wurde das Beispiel aus dem Korpus entfernt; die restlichen Beispiele wurden von einem menschlichen Beobachter dahingehend kategorisiert, ob sie eine der gesuchten Beziehungen ausdrücken.

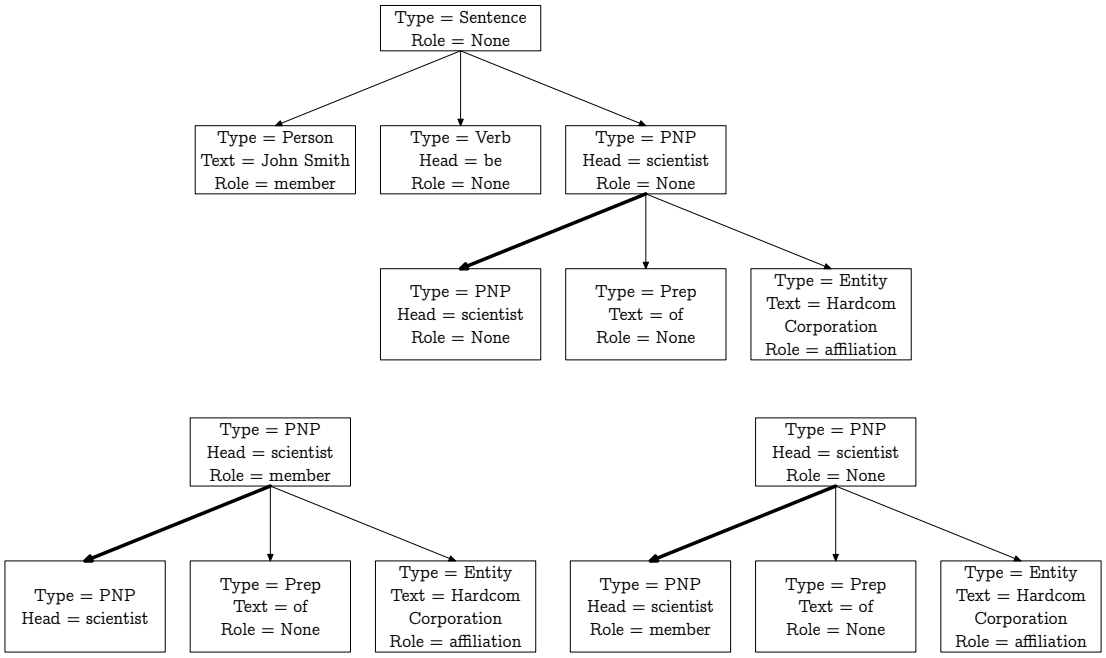


Abbildung 6: Drei Beispiele für die Beziehung *member–affiliation*, die aus dem Syntaxbaum für den Satz „John Smith is the chief scientist of the Hardcom Corporation.“ generiert wurden

Auf diese Weise wurden 2524 Kandidaten für die Beziehung *member–affiliation* gefunden. Von diesen enthalten 1262 in der Tat diese Beziehung; sie werden als **positive Beispiele** bezeichnet. Die restlichen 2261 Kandidaten, die diese Beziehung nicht ausdrücken, werden als **negative Beispiele** bezeichnet. Für die Beziehung *organization–location* wurden 1915 Kandidaten gefunden: 506 positive Beispiele und 1409 negative Beispiele.

Als Mass für die Performanz der Klassifikationsalgorithmen dienen die Werte *precision* (oder *true positive rate*) und *recall* (oder *false positive rate*). *precision* gibt den Anteil der korrekt vorhergesagten positiven Beispiele an den als positiv vorhergesagten Beispielen an. *recall* ist das Verhältnis zwischen korrekt als positiv vorhergesagten Beispiele und positiven Beispielen. Als kombiniertes Mass wird auch das F-Mass (*F-measure*) benutzt, das sich aus *precision* und *recall* als

$$\mathrm{Fm} = \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}}$$

ergibt.

Um die Verfahren auf Basis der Kernfunktion k_ν mit herkömmlichen Methoden vergleichen zu können, wurden eine Reihe von Merkmalen von Hand definiert und als Eingabe für merkmalsbasierte Verfahren benutzt. (Die benutzten Merkmale sind in Anhang C von (Zelenko u. a., 2003) zu finden.) Hier wurden die merkmalsbasierten Verfahren *Winnow* und *Naive Bayes* sowie die SVM mit linearer Kernfunktion zum Vergleich herangezogen.

Die Kernfunktion k_ν wurde in den bereits vorgestellten Verfahren *Voted Perceptron* und SVM eingesetzt; hier wurden jeweils zwei Varianten der Kernfunktion betrachtet: Einmal wurden Lücken in den Sequenzen der Kindknoten nicht erlaubt, bei weiteren Experimenten wurden sie zugelassen.

Die Ergebnisse der Experimente von Zelenko u. a. (2003) sind in Tabelle 2 abgedruckt. Es zeigt sich, dass die Methoden auf Basis der Kernfunktion ähnlich gute Ergebnisse erreichen wie die merkmalsbasierten Methoden; lässt man Lücken in den Sequenzen zu, erzielen sie sogar Ergebnisse, die von keiner merkmalsbasierten Methode erreicht werden. Man erreicht also gleich gute oder bessere Ergebnisse, ohne aufwändig Merkmale definieren und extrahieren zu müssen.

Die Lernkurven in Abbildung 7 geben das F-Mass in Abhängigkeit von der Zahl der Beispiele an. Hier zeigt sich, dass kernbasierte Methoden auch auf wenigen Beispielen ein höheres F-Mass erzielen als merkmalsbasierte Methoden.

6 Zusammenfassung

Neben statistischen Methoden und generativen Modellen sind diskriminative Methoden wie *Support Vector Machines* ein wichtiges Instrument zur Extraktion von Infor-

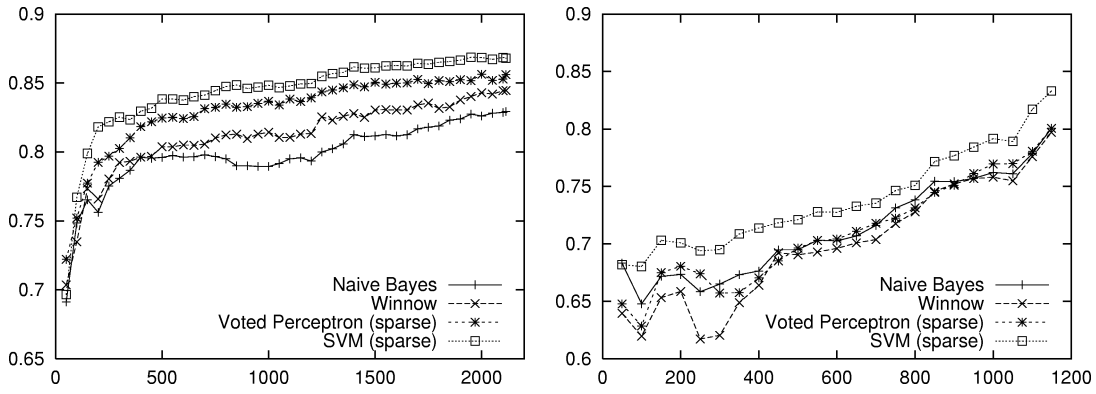
Tabelle 2: Verfahren auf Basis der Kernfunktion k_ν weisen auf dem benutzten Korpus eine höhere Performanz auf als merkmalsbasierte Verfahren (reproduziert aus (Zelenko u. a., 2003, Seite 1097).)

Performanz für die Beziehung *person–affiliation*

	recall [%]	*precision* [%]	*F-measure* [%]
Naive Bayes	75,59	91,88	82,93
Winnow	80,87	88,42	84,46
SVM (merkmalsbasiert)	76,21	91,67	83,22
Voted Perceptron (ohne Lücken)	79,58	89,74	84,34
SVM (ohne Lücken)	79,78	89,9	84,52
Voted Perceptron (mit Lücken)	81,62	90,05	85,61
SVM (mit Lücken)	82,73	91,32	**86,8**

Performanz für die Beziehung *organization–location*

	recall [%]	*precision* [%]	*F-measure* [%]
Naive Bayes	71,94	90,40	80,04
Winnow	75,14	85,02	79,71
SVM (merkmalsbasiert)	70,31	88,18	78,17
Voted Perceptron (ohne Lücken)	64,43	92,85	76,02
SVM (ohne Lücken)	71,43	92,03	80,39
Voted Perceptron (mit Lücken)	71	91,9	80,05
SVM (mit Lücken)	76,33	91,78	**83,3**



(a) *person–affiliation* (b) *organization–location*

Abbildung 7: Lernkurven für merkmalsbasierte Methoden im Vergleich zu Kernmethoden (reproduziert aus (Zelenko u. a., 2003, Seite 1098).) Auf der x-Achse ist die Zahl der Beispiele angegeben, während auf der y-Achse das erreichte F-Mass abgetragen ist

mationen. Durch den Einsatz von Kernfunktionen können in der Computerlinguistik übliche Repräsentationen direkt verarbeitet werden; dabei wird implizit eine wesentlich grössere Zahl von Merkmalen benutzt als mit anderen Methoden. Eine aufwändige Suche nach passenden Merkmalen ist nicht nötig.

Der vorgestellte Ansatz zur Extraktion von Beziehungen ist effizient berechenbar und bietet eine gute Erkennungsleistung im Vergleich zu Methoden, die von Hand bestimmte Merkmale benutzen. Bei Übertragung auf eine andere Domäne sind keine Anpassungen notwendig.

Aktuelle Arbeiten erlauben es, generative Modelle aus Beispielen zu lernen: Tsochantaridis u. a. (2004) stellen eine Methode vor, Hidden-Markov-Modelle und kontextfreie Grammatiken mit den *Support Vector Machine*-Ansatz zu lernen. Die so gewonnen Modelle können dann zur Extraktion von Informationen benutzt werden.

Literatur

[Aone und Ramos-Santacruz 2000] Aone, Chinatsu ; Ramos-Santacruz, Mila: REES: a large-scale relation and event extraction system. In: *Proceedings of the sixth conference on Applied natural language processing*, Morgan Kaufmann Publishers Inc., 2000, S. 76–83

[Appelt u. a. 1993] Appelt, Douglas E. ; Hobbs, Jerry R. ; Bear, John ; Israel,

David ; TYSON, Mabry: FASTUS: A Finite-State Processor For Information Extraction From Real-World Text. In: *Proceedings of IJCAI-93*, Aug 1993

[Brin 1998] BRIN, Sergey: Extracting Patterns and Relations from the World Wide Web. In: *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, URL citeseer.ist.psu.edu/brin98extracting.html, 1998

[Collins und Duffy 2001] COLLINS, Michael ; DUFFY, Nigel: Convolution kernels for natural language. In: *Proceedings of NIPS-2001*, 2001

[Cortes und Vapnik 1995] CORTES, C. ; VAPNIK, V. N.: Support vector networks. In: *Machine Learning* 20 (1995), S. 273–297

[Freund und Schapire 1999] FREUND, Yoav ; SCHAPIRE, Robert E.: Large margin classification using the perceptron algorithm. In: *Machine Learning* 37 (1999), Nr. 3, S. 277–296

[Gärtner 2003] GÄRTNER, Thomas: A survey of kernels for structured data. In: *SIGKDD Explor. Newsl.* 5 (2003), Nr. 1, S. 49–58. – URL http://doi.acm.org/10.1145/959242.959248

[Horváth u. a. 2004] HORVÁTH, Tamás ; GÄRTNER, Thomas ; WROBEL, Stefan: Cyclic pattern kernels for predictive graph mining. In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, S. 158–167. – ISBN 1-58113-888-9

[Kondor und Jebara 2003] KONDOR, Risi ; JEBARA, Tony: A Kernel between Sets of Vectors. In: FAWCETT, Tom (Hrsg.) ; MISHRA, Nina (Hrsg.): *Proceedings of the Twentieth International Conference on Machine Learning*, AAAI Press, 2003

[Lodhi u. a. 2002] LODHI, Huma ; SAUNDERS, Craig ; SHAWE-TAYLOR, John ; CHRISTIANINI, Nello ; WATKINS, Chris: Text classification using string kernels. In: *Journal of Machine Learning Research* (2002)

[Müller u. a. 2001] MÜLLER, Klaus-Robert ; MIKA, Sebastian ; RÄTSCH, Gunnar ; TSUDA, Koji ; SCHÖLKOPF, Bernhard: An introduction to kernel-based learning algorithms. In: *IEEE Transactions on Neural Networks* 12 (2001), Nr. 2, S. 181–202

[Riloff 1993] RILOFF, Ellen: Automatically Constructing a Dictionary for Information Extraction Tasks. In: *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI Press/MIT Press, 1993, S. 811–816. – URL http://www.cs.utah.edu/~riloff/psfiles/aaai93.pdf

[Riloff und Lehnert 1993] Riloff, Ellen ; Lehnert, Wendy: Automated dictionary construction for information extraction from text. In: *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, Mar 1993, S. 93–99

[Rosenblatt 1988] Rosenblatt, F.: *The perception: a probabilistic model for information storage and organization in the brain*. S. 89–114. In: *Neurocomputing: foundations of research*, MIT Press, 1988. – ISBN 0-262-01097-6

[Seymore u. a. 1999] Seymore, Kristie ; McCallum, Andrew ; Rosenfeld, Roni: *Learning Hidden Markov Model Structure for Information Extraction*. AAAI'99 Workshop on Machine Learning for Information Extraction. 1999. – URL http://www.cs.umass.edu/~mccallum/papers/iestruct-aaaiws99.ps

[Tsochantaridis u. a. 2004] Tsochantaridis, Ioannis ; Hofmann, Thomas ; Joachims, Thorsten ; Altun, Yasemin: Support Vector Machine Learning for Interdependent and Structured Output Spaces. In: *21st Int. Conf. Machine Learning*, URL citeseer.ist.psu.edu/tsochantaridis04support.html, 2004

[Yangarber und Grishman 1998] Yangarber, Roman ; Grishman, Ralph: NYU: Description of the Proteus/PET System as used for MUC-7 ST. In: *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, Apr 1998

[Zelenko 2003] Zelenko, Dmitry: *Machine Learning for Information Extraction*, University of Illinois at Urbana-Champaign, Dissertation, 2003

[Zelenko u. a. 2003] Zelenko, Dmitry ; Aone, Chinatsu ; Richardella, Anthony: Kernel methods for relation extraction. In: *Journal of Machine Learning Research* (2003), Nr. 3, S. 1083–1106