Statistische Methoden in der Sprachverarbeitung

Helmut Schmid

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
schmid@cis.uni-muenchen.de

Stand 15. Januar 2024

Änderungen

Datum: 11. 9. 2023 Einleitung für neue Übungsaufgaben angepasst

Datum: 18. 10. 2023 Folie 4 angepasst

Termine

- Vorlesung: Montag 12-14 Uhr (c.t.)
  - Klausur: prüft das theoretische Wissen
- Übungen: Montag 14-16 Uhr (c.t.)
  - vorwiegend Programmieraufgaben
  - Die Note ergibt sich aus der Bewertung der abgegebenen Programme.
  - Über eine Moodle-Seite können Sie Fragen zu den Aufgaben stellen.
- Repetitorium: Mittwoch 12-14 Uhr (c.t.), in C003, ab der 2. Semesterwoche
  - Wiederholung des Stoffes
  - Viel Zeit für Ihre Fragen
- Tutorium: (Online-Termin-Umfrage), ab der 2. Semesterwoche
  - gehalten von Herrn Nie
- voraussichtliche Klausurtermine
  - Vorlesung: letzte Woche der Vorlesungszeit

Übungen

- Sie lösen Programmieraufgaben in Gruppen von max. 3 Personen.
- Jede Gruppe schickt mir innerhalb der ersten Woche Ihren **Gruppennamen** und die Namen und **Gitlab-UserIDs** der Mitglieder per Email. Die Gitlab-UserID ist ein Zahl, die Sie in Ihrem Gitlab-Profile finden.
- Die neue Aufgabe wird am Montag in der Übungsstunde besprochen.
- Anschließend lade ich die neue Übung mit einer Codevorlage für jede Gruppe auf Gitlab hoch.
- Sie bearbeiten die Aufgabe bis zum Sonntag der nächsten Woche und laden Ihren Code auf Gitlab hoch.
- Ich lade Ihren Code herunter und evaluiere ihn automatisiert.
- Die Codevorlage umfasst eine Reihe von Funktionen/Methoden, die Sie implementieren müssen. Dazu ersetzen Sie den Befehl “pass” durch Ihre Implementierung.
- Bei der Evaluierung wird die Ausgabe Ihrer Funktion mit der Ausgabe einer Referenz-Implementierung verglichen.
- Für jede korrekt implementierte Funktion gibt es einen Punkt.
- Die Note ergibt sich aus der Zahl der erreichten Punkte.

Literatur

Christopher Manning und Hinrich Schütze:
Foundations of Statistical Natural Language Processing

Daniel Jurafsky und James H. Martin.
Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics

Weitere Informationen finden Sie auf der Kursseite, die über meine Homepage erreichbar ist.

Statistische Methoden und Computerlinguistik

Die Computerlinguistik entwickelt sich immer mehr zu einem Spezialgebiet des maschinellen Lernens

verwendete Methoden: statistische Modelle, neuronale Netzwerke

Vorgehensweise beim maschinellen Lernen (Beispiel: Wortart-Annotation)

- Daten sammeln und annotieren (bspw. mit Wortart annotiertes Korpus)
- statistisches Modell entwerfen (bspw. ein Hidden-Markow-Modell)
- Modell trainieren (Schätzung der Wahrscheinlichkeiten)
- Modell evaluieren (Genauigkeit auf neuen Daten ermitteln)

Überblick

1. Grundlagen
   - Textkorpora
   - Mathematische Grundlagen
2. Statistische Tests
   - Kollokationsextraktion
3. Generative Modelle und Anwendungen
   - Markowmodelle: Sprachidentifizierung
   - Parameterglättung
   - Naïve-Bayes-Modelle: Wortbedeutungs-Desambiguierung
   - Hidden-Markov-Modelle: Wortart-Tagging
   - PCFGs: Statistisches Parsen
   - Berkeley-Parser
4. Diskriminative Modelle
   - Perzeptron-Algorithmus
   - Maximum-Entropy-Modelle

Kursziele

Sie sollten im Kurs folgende Fähigkeiten erwerben:

- allgemein: kompliziertere mathematische **Formeln** lesen und verstehen
- die behandelten statistischen **Modelle** verstehen
- die **Anwendungen** der Modelle verstehen
- die **Implementierungen** der Modelle verstehen
- die Modelle selbst nachimplementieren können
- nicht behandelte statistische Modelle schnell verstehen

Anmerkungen zum Kurs

- Wie der Kurstitel vermuten lässt, ist dieser Kurs relativ **mathematisch**.
- Die mathematischen Kenntnisse aus der Schule sollten aber ausreichen.
- Für das Verständnis der statistischen Modelle reicht es nicht, die Vorlesung und die Übungen zu besuchen. Sie sollten den Stoff nach der Vorlesung noch einmal anhand der Folien **wiederholen**, sich notieren, was Sie nicht verstanden haben, und dann im Repetitorium/Tutorium **Fragen** dazu stellen.
- Rat: Bleiben Sie während des Semesters am Ball. Es ist schwer, einen größeren Rückstand am Semesterende aufzuholen.

Textkorpus

Korpus: Sammlung von Texten für linguistische Zwecke

Einfache Korpusstatistiken für die Erzählung “Tom Sawyer”:

- Gesamtzahl der Wörter: 71370 (Tokens)
- Zahl der unterschiedlichen Wörter: 8018 (Types)
- Zahl der Wörter, die einmal auftraten: 3993 (Hapax Legomena)

Wörter sortiert nach Häufigkeit

Wort	Häufigkeit
the	3332
and	2972
a	1775
to	1725
of	1440
was	1161
it	1027
in	906
that	877
he	877
I	783
his	772
you	686
Tom	679
with	642

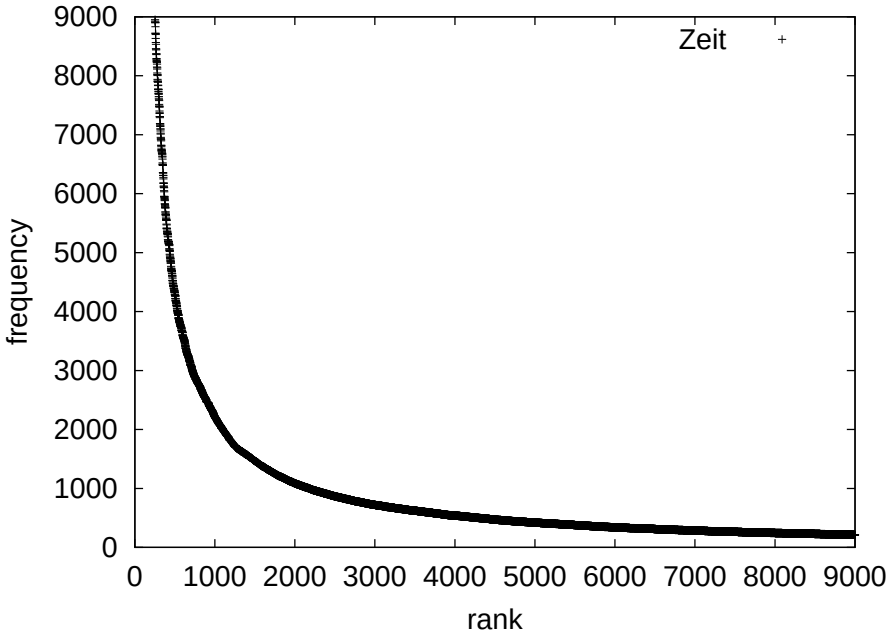
Funktionswörter: the, and, a, ...

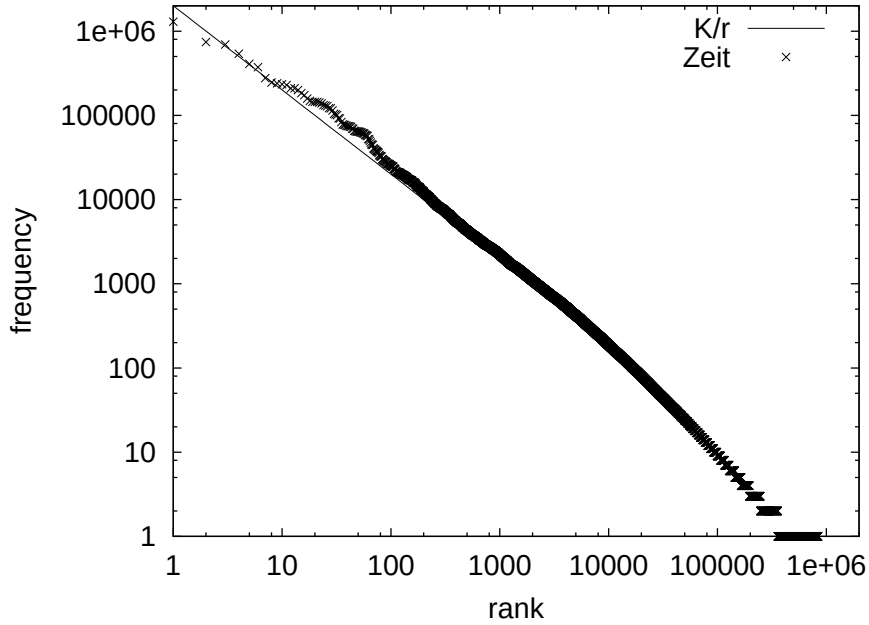
Inhaltswörter: Tom

$\Rightarrow$ Die meisten sehr häufigen Wörter sind Funktionswörter.

Grafische Darstellung der sortierten Liste

Deutsches Zeitungskorpus (Die Zeit)





lineare Skala

doppelt logarithmisch

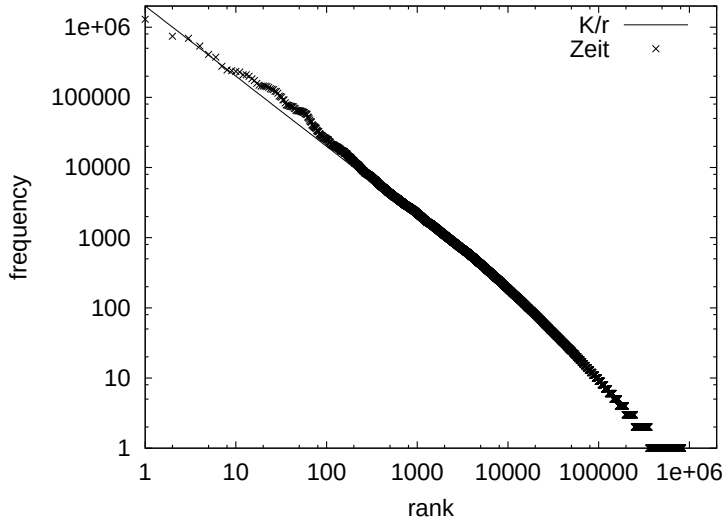
In den Schaubildern entspricht der x-Wert (rank) der Position eines Wortes in der sortierten Liste und der y-Wert seiner Häufigkeit.

$\Rightarrow$ Zipf'sche Verteilung

Zipf'sche Verteilung

Die Kurve im rechten Schaubild entspricht annähernd einer Geraden:

$$y = A - x$$



wobei A der y-Achsenabschnitt ist.

Da das Schaubild eine doppelt logarithmische Darstellung verwendet, gilt

$$x = \log r \text{ und } y = \log f(r)$$

$f(r)$ ist die gesuchte Funktion, die den Rang r des Wortes in der sortierten Liste auf seine angenäherte Häufigkeit abbildet.

Durch Einsetzen in die erste Formel erhalten wir

$$\log f(r) = A - \log r$$

Wenn beide Seiten zum Exponenten genommen werden, ergibt sich

$$e^{\log f(r)} = f(r) = e^{A - \log r} = \frac{e^A}{e^{\log r}} = \frac{K}{r} \sim \frac{1}{r} \qquad \text{mit } K = e^A$$

Die Worthäufigkeit ist also proportional zum Kehrwert des Wortranges.

Zipf'sches Gesetz

Wort	f	r	$f \cdot r$
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
two	104	100	10400
turned	51	200	10200
you'll	30	300	9000
family	8	1000	8000
brushed	4	2000	8000
sins	2	3000	6000
Could	2	4000	8000
Applausive	1	8000	8000

Zipf'sches Gesetz:

$$f \sim \frac{1}{r} \quad (f \cdot r \approx K)$$

- $\Rightarrow$ Wenige Wörter sind sehr häufig
- $\Rightarrow$ Die meisten Wörter sind sehr selten.
- $\Rightarrow$ Das Produkt $f \cdot r$ aus Rang und Häufigkeit variiert nicht sehr stark.

Häufigkeiten zweiter Ordnung

n	f_n	
1	3993	(Hapax Legomena)
2	1292	
3	664	
4	410	
5	243	
6	199	
7	172	
8	131	
9	82	
10	91	
11-50	540	
51-100	99	
>100	102	

Wieviele Wörter tauchten genau n Mal auf?

Mathematische Grundlagen

Zufallsexperiment

In der Statistik geht es um die Wahrscheinlichkeit von Ereignissen:

Beispiel: Wie wahrscheinlich ist es, sechs Richtige im Lotto zu haben?

Zufallsexperiment: Experiment (Versuch) mit mehreren möglichen Ausgängen (z.B. Wurf von zwei Würfeln)

Ergebnis: Resultat eines Experimentes
(z.B. 3 Augen auf Würfel 1 und 4 Augen auf Würfel 2)

Ergebnisraum Ω: Menge aller möglichen Ergebnisse
(hier {(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(2,1),..,(6,6)})

Ereignis $A \subseteq \Omega$: Teilmenge des Ergebnisraumes
(z.B. "7 Augen mit 2 Würfeln" = {(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)})

Elementarereignis: anderes Wort für Ergebnis

Stichprobe: Folge der Ergebnisse bei einem wiederholten Experiment

Wahrscheinlichkeitsverteilung

Wahrscheinlichkeitsverteilung: Funktion, die jedem Ergebnis o einen Wert zwischen 0 und 1 zuweist, so dass gilt:

$$\sum_{o \in \Omega} p(o) = 1$$

Die Wahrscheinlichkeit $P(A)$ eines Ereignisses A ist die Summe der Wahrscheinlichkeiten der zugehörigen Ergebnisse.

$$P(A) = \sum_{o \in A} p(o)$$

Beispiel: Die Wahrscheinlichkeit des Ereignisses A, dass die Zahl der Augen beim Wurf eines Würfels gerade ist:

$$P(A) = p(2) + p(4) + p(6) = \frac{3}{6} = \frac{1}{2}$$

Anmerkung: $P(.)$ ist keine Wahrscheinlichkeitsverteilung, da die Summe der Wahrscheinlichkeiten aller möglichen Ereignisse größer als 1 ist.

Bedingte Wahrscheinlichkeit

Bedingte Wahrscheinlichkeit: Wahrscheinlichkeit eines Ereignisses A, wenn das Ereignis B bekannt ist:

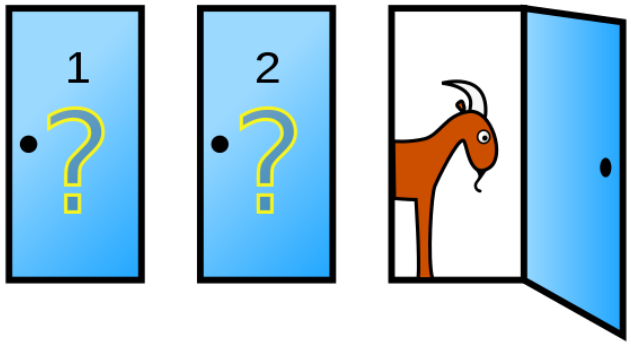
$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$\cap$ ist der Schnittmengenoperator.

Beispiel: Wahrscheinlichkeit $P(A|B)$, dass die Augenzahl eines Würfels gerade ist, wenn die Augenzahl größer als 3 ist:

$$P(A|B) = \frac{p(4) + p(6)}{p(4) + p(5) + p(6)} = \frac{2/6}{3/6} = \frac{2}{3}$$

Ziegenproblem



Nehmen Sie an, Sie sind in einer Spielshow und haben die Wahl zwischen drei Toren. Hinter einem der Tore ist ein Auto (das Sie gewinnen wollen), hinter den anderen sind Ziegen (als Trostpreise). Sie wählen ein Tor, sagen wir Tor 1, und der Showmaster (der weiß, was hinter den Toren ist) öffnet ein anderes Tor, sagen wir Tor 3, hinter dem eine Ziege steht. Er fragt Sie dann: "Möchten Sie das Tor wechseln und Tor 2 nehmen?"

Frage: Ist es vorteilhaft, die Wahl des Tores zu ändern?

–Whitaker/vos Savant 1990

Ziegenproblem (Lösung)

Erfolgsquoten der Strategien “Tor wechseln” und “Tor beibehalten”.

(Annahme: Die Kandidatin hat zunächst Tor1 gewählt.)

Tor1	Tor2	Tor3	Wechseln	Bleiben
Ziege	Ziege	Auto	1	0
Ziege	Auto	Ziege	1	0
Auto	Ziege	Ziege	0	1
			2/3	1/3

Wenn die Kandidatin zunächst ein anderes Tor gewählt hat, geht die Rechnung analog.

Schlussfolgerung: Es ist also besser, das Tor zu wechseln.

Zufallsvariablen

Zufallsvariable: Funktion, welche jedem Ergebnis eine reelle Zahl zuweist.

Beispiel: Abbildung der Noten *sehr gut, gut, befriedigend, ausreichend, mangelhaft, ungenügend* auf die Zahlen 1, 2, 3, 4, 5, 6.

Wahrscheinlichkeit eines Wertes x der Zufallsvariablen X:

$$p(X = x) = p(x) = P(A_x)$$

$p(x)$: ist eine abkürzende Schreibweise, die oft verwendet wird, wenn klar ist, zu welcher Zufallsvariablen der Wert x gehört.

A_x: Menge der Ergebnisse, für welche die Zufallsvariable X den Wert x liefert.

Bernoulli-Experiment

Eine Zufallsvariable, die nur die Werte 0 und 1 liefert, nennt man **Bernoulli-Experiment**.

Beispiel: Eine Zufallsvariable X mit

$$X(o) = \begin{cases} 0 & \text{falls } o \in \{\text{mangelhaft, ungenügend}\} \\ 1 & \text{falls } o \in \{\text{sehr gut, gut, befriedigend, ausreichend}\} \end{cases}$$

$$p(X{=}1) = p(\text{sehr gut}) + p(\text{gut}) + p(\text{befriedigend}) + p(\text{ausreichend})$$

Erwartungswert

Der **Erwartungswert** ist der Mittelwert einer Zufallsvariablen:

$$E(X) = \sum_{x \in \Omega_X} p(x)x$$

Ω_X ist hier der Wertebereich der Zufallsvariablen X.

Beispiel: erwartete Augenzahl bei einem fairen Würfel: 1/6*(1+2+3+4+5+6)=3,5

Erwartungswert einer Funktion f:

$$E(f(X)) = \sum_{x \in \Omega_X} p(x)f(x)$$

Beispiel: erwartete quadrierte Augenzahl: 1/6*(1+4+9+16+25+36) = 91/6

Varianz

Die **Varianz** ist ein Maß dafür, wie stark die einzelnen Werte vom Mittelwert abweichen:

$$Var(X) = E((X - E(X))^2) = E(X^2) - E(X)^2$$

Beispiel: Würfel: $1/6 * (1 - 3,5)^2 + 1/6 * (2 - 3,5)^2 + ...$ $\rightarrow$ Ü

Die **Standardabweichung** ist die Wurzel aus der Varianz.

Gemeinsame Verteilungen

Die **gemeinsame Verteilung** zweier Zufallsvariablen X und Y:

$$p(x, y) = p(X{=}x, Y{=}y) = P(A_x \cap A_y)$$

A_x **(A_y):** Menge der Ergebnisse, die von der Zufallsvariablen X (Y) auf den Wert x (y) abgebildet werden.

Beispiel: Augenzahl eines Würfels

AZ	X (AZ>3)	Y (AZ gerade)
1	0	0
2	0	1
3	0	0
4	1	1
5	1	0
6	1	1

X	Y	$p(x, y)$
0	0	2/6
0	1	1/6
1	0	1/6
1	1	2/6

Randverteilungen

Aus der gemeinsamen Verteilung der Zufallsvariablen X und Y, kann man die Verteilungen von X und Y berechnen.

Man nennt diese die **Randverteilungen** (marginal distribution):

$$p_X(x) = \sum_{y \in \Omega_Y} p(x, y) \qquad p_Y(y) = \sum_{x \in \Omega_X} p(x, y)$$

Oft schreibt man statt $p_X(x)$ einfach $p(x)$.

Beispiel:

AZ	X	Y
1	0	0
2	0	1
3	0	0
4	1	1
5	1	0
6	1	1

X	Y	$p(x,y)$
0	0	2/6
0	1	1/6
1	0	1/6
1	1	2/6

	$p_X(x)$	$p_Y(y)$
0	3/6	3/6
1	3/6	3/6

Bedingte Wahrscheinlichkeit: $p(y|x) = \frac{p(x,y)}{p_X(x)}$

Statistische Unabhängigkeit

Unabhängigkeit: Die Zufallsvariablen X und Y sind statistisch unabhängig, falls für alle x und y gilt:

$$p(x, y) = p_X(x)\, p_Y(y)$$

Beispiel: Wurf zweier Würfel (analog für andere Würfelergebnisse)

$$p(W_1 = 1, W_2 = 4) = 1/36 \quad p(W_1 = 1) \cdot p(W_2 = 4) = 1/6 \cdot 1/6 = 1/36$$

wobei $W_1(o)$ die Augenzahl des ersten Würfels beim Wurfergebnis o ist.

Gegenbeispiel: Augenzahl > 3 (=X) und Augenzahl gerade (=Y)

$$p(X = 0, Y = 1) = 1/6 \quad \neq \quad 1/4 = p(X = 0) \cdot p(Y = 1)$$

Kettenregel

Eine gemeinsame Wahrscheinlichkeit kann in ein Produkt bedingter Wahrscheinlichkeiten zerlegt werden.

$$\begin{aligned} p(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) &= p(x_1, x_2, ..., x_n) \\ &= p(x_1)p(x_2|x_1)...p(x_n|x_1, ..., x_{n-1}) \\ &= \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) \end{aligned}$$

Beispiel: Für Folgen von 3 Wörtern gilt:

$p(W_1{=}Es, W_2{=}gibt, W_3{=}ein) =$
$p(W_1{=}Es)\ p(W_2{=}gibt|W_1{=}Es)\ p(W_3{=}ein|W_1{=}Es, W_2{=}gibt)$

kürzer aber weniger eindeutig: $p(Es, gibt, ein) = p(Es)\ p(gibt|Es)\ p(ein|Es, gibt)$

$p(W_i = w), i \in \{1, 2, 3\}$: Summe der Wahrscheinlichkeiten aller Wort-Tripel, bei denen an Position i das Wort w steht.

Anmerkung: Der Ausdruck $W_1 = Es$ ist etwas salopp, da eine Zufallsvariable nur Zahlen als Werte liefert. Nehmen Sie daher an, dass jedes Wort auf eine eindeutige Zahl abgebildet wird, und dass Es die Zahl des Wortes "Es" repräsentiert.

Theorem von Bayes

Mit dem Theorem von Bayes kann eine bedingte Wahrscheinlichkeit "umgedreht" werden

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Man nennt $p(x)$ auch *Apriori-* und $p(x|y)$ *Aposteriori-Wahrscheinlichkeit*.

Beispiel: Wortpaare

$$p(W_2{=}York|W_1{=}New) = p(W_1{=}New|W_2{=}York)p(W_2{=}York)/p(W_1{=}New)$$

Ohne die Zufallsvariablen wird die Formel mehrdeutig:

$$p(York|New) = p(New|York)p(York)/p(New)$$

Bei Verwendung von Variablen wird die Formel wieder eindeutig, wenn die Zuordnung der Variablen zu den Zufallsvariablen klar ist:

$$p(w_2|w_1) = p(w_1|w_2)p(w_2)/p(w_1)$$

Anwendung des Bayes'schen Theorems

Angenommen von 10000 Menschen leidet einer an der seltenen Krankheit X. Es gibt einen experimentellen Test, der die Krankheit in 90% der Fälle erkennt und bei Gesunden in 5% der Fälle Fehlalarm gibt. Sie machen diesen Test und bekommen ein positives Ergebnis. Wie wahrscheinlich ist es, dass Sie erkrankt sind?

$$
\begin{aligned}
p(\text{krank}|\text{positiv}) &= \frac{p(\text{positiv}|\text{krank})p(\text{krank})}{p(\text{positiv})} \\
&= \frac{p(\text{positiv}|\text{krank})p(\text{krank})}{p(\text{positiv},\text{krank}) + p(\text{positiv},\text{gesund})} \\
&= \frac{p(\text{positiv}|\text{krank})p(\text{krank})}{p(\text{positiv}|\text{krank})p(\text{krank}) + p(\text{positiv}|\text{gesund})p(\text{gesund})} \\
&= \frac{0.9 * 0.0001}{0.9 * 0.0001 + 0.05 * 0.9999} = 0.001797
\end{aligned}
$$

$\Rightarrow$ Das positive Testergebnis hat die Wahrscheinlichkeit, dass Sie krank sind, von 0.00001 auf 0.0018 erhöht.

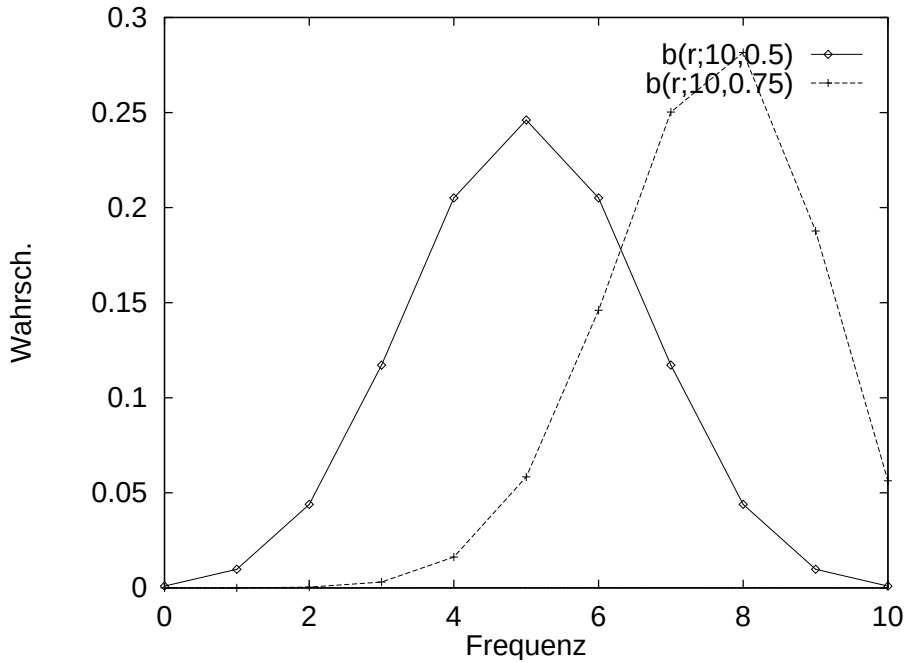
Binomialverteilung

Eine **Binomialverteilung** ergibt sich, wenn ein **Bernoulli-Experiment** (Ergebnisse 0 und 1, Bsp. Münzwurf: X(Kopf)=1, X(Zahl)=0) n-mal wiederholt wird. Die Binomialverteilung ist die Wahrscheinlichkeit, dabei genau r viele "1"-Ereignisse zu bekommen, wenn die Wahrscheinlichkeit des "1"-Ereignisses p ist:

$$b(r;n,p) = \binom{n}{r} p^r (1-p)^{n-r}$$

$$\binom{n}{r} = \frac{n!}{(n-r)!\, r!}$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$



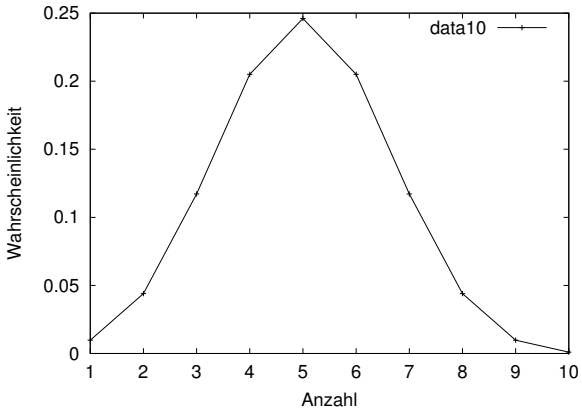
Beispiel: Die mittlere Kurve gibt an, wie wahrscheinlich es bei 10 Münzwürfen ist, 0-mal, 1-mal, 2-mal etc. Kopf zu erhalten.

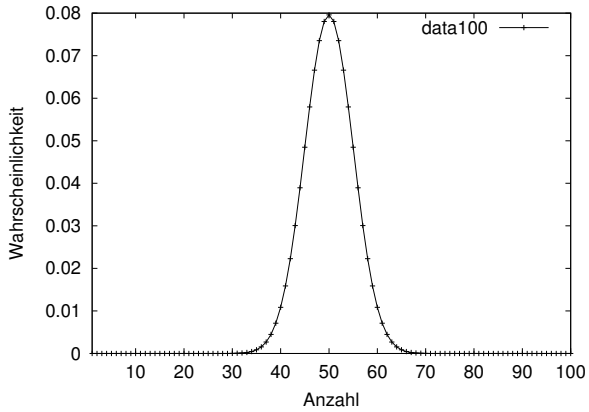
Binomialverteilung

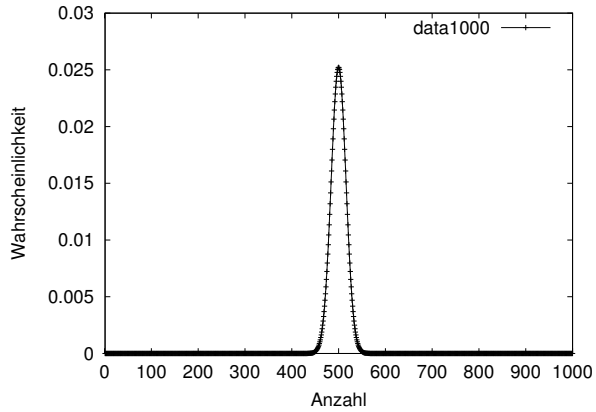
Wie ergibt sich die Formel: $\frac{n!}{(n-r)!\ r!}\ p^r(1-p)^{n-r}$

- Jede mögliche Stichprobe der Größe n mit genau r vielen "1"-Ergebnissen hat die Wahrscheinlichkeit $p^r(1-p)^{n-r}$, wenn die Einzelergebnisse unabhängig sind und das "1"-Ergebnis die Wahrscheinlichkeit p und das "0"-Ergebnis die Wahrscheinlichkeit 1-p besitzt.
- Es gibt nicht nur eine sondern viele Folgen von Nullen und Einsen der Länge n, die genau r viele Einsen enthalten. Ihre Wahrscheinlichkeiten müssen addiert werden. Wieviele Folgen sind es genau?
- Jede mögliche Folge ist durch die Positionen ihrer "1"-Ereignisse charakterisiert. Wieviele Möglichkeiten gibt es, r viele "1"-Positionen aus n Positionen auszuwählen? (verwandtes Problem: Zahl der Lotto-Kombinationen)
- Für die erste Position haben wir n Möglichkeiten, für die zweite (n-1), weil eine Position weggefallen ist, usw. Insgesamt haben wir $\frac{n!}{(n-r)!}$ Möglichkeiten.
- Davon sind viele äquivalent: Ob Sie die Positionen 3,9,7 oder 7,3,9 wählen, ist egal.
- Bei jeder Positionsfolge der Länge r gibt es $r!$ viele solche Permutationen.
- Nach Division durch die Zahl der Permutationen, bekommen wir die obige Formel.

Binomialverteilung







Der Erwartungswert der Binomialverteilung $b(r; n, p)$ ist np.

$\Rightarrow$ In einer Stichprobe der Größe n erwarten wir, dass die Zahl f der 1-Ereignisse etwa np beträgt.

$\Rightarrow$ Die Wahrscheinlichkeit p ist also annähernd f/n. Je größer die Stichprobe ist, desto zuverlässiger approximiert f/n die Wk. p.

$\Rightarrow$ Wir können daher die Häufigkeit f in einer Stichprobe mit ausreichender Größe n benutzen, um die Wahrscheinlichkeit p zu schätzen.

Wahrscheinlichkeitsschätzung

$$\tilde{p}(x) = \frac{f(x)}{n}$$

Die **Relative Häufigkeit** $f(x)/n$ ist die Zahl der Vorkommen $f(x)$ eines Ereignisses x geteilt durch die Stichprobengröße n.

Für zunehmende Stichprobengröße n konvergiert die relative Häufigkeit zu der tatsächlichen Wahrscheinlichkeit eines Ereignisses.

Genauer: Die Wahrscheinlichkeit, dass die relative Häufigkeit um mehr als ϵ von der tatsächlichen Wahrscheinlichkeit abweicht, konvergiert für zunehmende Stichprobengröße gegen 0.

Information

Welcher Artikel hat bessere Chancen, es auf die Titelseite zu schaffen?

- Bayern München besiegt 1860 München
- 1860 München besiegt Bayern München

$\Rightarrow$ Je unwahrscheinlicher ein Ereignis ist, desto informativer ist es.

Informationsgehalt: $I(x) = -log_2\ p(x)$

Beispiel: Der Informationsgehalt des Ergebnisses eines Münzwurfes beträgt

$$-\log_2 0.5 = 1\text{Bit}$$

Bit ist die Maßeinheit der Information.

Information

Welcher Artikel hat bessere Chancen, es auf die Titelseite zu schaffen?

- Bayern München besiegt 1860 München
- 1860 München besiegt Bayern München

$\Rightarrow$ Je unwahrscheinlicher ein Ereignis ist, desto informativer ist es.

Informationsgehalt: $I(x) = -log_2\ p(x)$

Beispiel: Der Informationsgehalt des Ergebnisses eines Münzwurfes beträgt

$$-\log_2 0.5 = 1\text{Bit}$$

Bit ist die Maßeinheit der Information.

Entropie

Die **Entropie** misst, wieviel Information ein Zufallsereignis im Mittel liefert.

Entropie einer Zufallsvariablen X mit der Wahrscheinlichkeitsverteilung $p(x)$:

$$H(X) = H(p) = -\sum_{x \in \Omega_X} p(x) \log_2 p(x) = E(-log_2 p(x))$$

Beispiel: Die Entropie beim Wurf eines Würfels beträgt
$-6 \cdot 1/6 \cdot \log_2 1/6 = \log_2 6 = 2,58 Bit$

Bezug zur Kodierungstheorie: Die Entropie ist eine Untergrenze für die Zahl der Bits, die im Mittel benötigt werden, um die Ergebnisse einer Folge von (unabhängigen, identisch verteilten) Zufallsexperimenten zu kodieren.

Kodierung

Wir wollen die 4 Vokale a, e, i, o kodieren, wobei p(x) die jeweilige Wk. ist.

x	p(x)	$-\log_2 p(x)$	Code(x)	Länge(Code(x))
a	0.5	1	1	1
e	0.25	2	01	2
i	0.125	3	001	3
o	0.125	3	000	3

Kein Code darf ein Präfix eines anderen Codes sein!

(Sonst ist keine eindeutige Dekodierung möglich.)

$$H(p) = -\sum_{x \in \Omega_X} p(x) log_2 p(x) = 0.5*1+0.25*2+0.125*3+0.125*3 = 1.75$$

1.75 ist auch der Erwartungswert der Codelänge im obigen Beispiel.

$\Rightarrow$ Der Code ist also optimal.

Gemeinsame Entropie

Die **gemeinsame Entropie** zweier Zufallsvariablen ist wie folgt definiert

$$H(X,Y) = -\sum_{x \in \Omega_X} \sum_{y \in \Omega_Y} p(x,y)\ log_2\ p(x,y)$$

Beispiel: Die Entropie $H(W_1, W_2)$ beim Wurf von 2 Würfeln beträgt
$-36 \cdot 1/36 \cdot \log_2 1/36 = \log_2 36 = 5,17$ Bit

W_1 = Augenzahl des 1. Würfels

W_2 = Augenzahl des 2. Würfels

Gemeinsame Entropie

Wir definieren zwei Zufallsvariablen auf den Ergebnisses eines Würfelwurfs:
X = 1 falls Augenzahl $>$ 3 sonst 0
Y = 1 falls Augenzahl gerade sonst 0

Bei einem fairen Würfel erhalten wir die gemeinsame Verteilung:

X	Y	$p(x, y)$
0	0	2/6
0	1	1/6
1	0	1/6
1	1	2/6

$$H(X, Y) = -(2/6 \log 2/6 + 1/6 \log 1/6 + 1/6 \log 1/6 + 2/6 \log 2/6)$$

Bedingte Entropie

Die bedingten Wahrscheinlichkeiten $p(y|x)$ bilden zwei Wahrscheinlichkeitsverteilungen $p(y|0)$ und $p(y|1)$.

X	Y	$p(x,y)$	$p(y\|x)$
0	0	2/6	2/3
0	1	1/6	1/3
1	0	1/6	1/3
1	1	2/6	2/3

Für jede dieser Wahrscheinlichkeitsverteilungen können wir die Entropie berechnen

$$H_0(Y) = -\sum_{y \in \Omega_Y} p(y|0) \log p(y|0) = -(2/3 \log 2/3 + 1/3 \log 1/3)$$

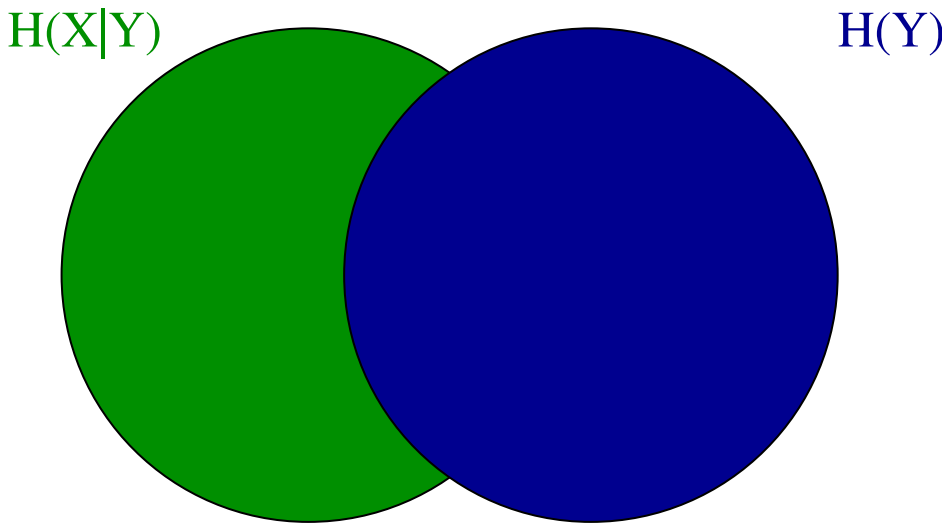
$$H_1(Y) = -\sum_{y \in \Omega_Y} p(y|1) \log p(y|1) = -(1/3 \log 1/3 + 2/3 \log 2/3)$$

Bedingte Entropie

Wir definieren die **bedingte Entropie** $H(Y|X)$, indem wir die Entropien $H_x(Y)$ mit $p(x) = \sum_{y \in \Omega_Y} p(x,y)$ gewichten und aufsummieren.

$$
\begin{aligned}
H(Y|X) &= \sum_{x \in \Omega_X} p(x) H_x(Y) \\
&= \sum_{x \in \Omega_X} p(x) \left[- \sum_{y \in \Omega_Y} p(y|x) \; log_2 \; p(y|x) \right] \\
&= - \sum_{x \in \Omega_X} \sum_{y \in \Omega_Y} p(x) \; p(y|x) \; log_2 \; p(y|x) \\
&= - \sum_{x \in \Omega_X} \sum_{y \in \Omega_Y} p(x,y) \; log_2 \; p(y|x)
\end{aligned}
$$

Bedingte Entropie



Es gilt: $H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$

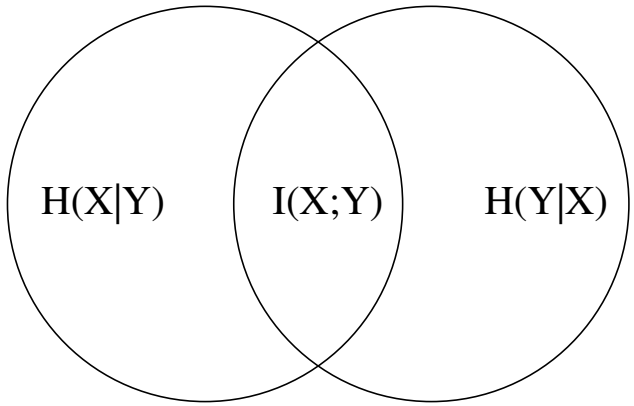
$\rightarrow$ Ü

$H(X|Y)$ ist die Rest-Information, die X noch liefert, wenn Y bekannt ist.

Mutual Information

Wegen $H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$

gilt auch $H(X) - H(X|Y) = H(Y) - H(Y|X) =: I(X;Y)$



Die Mutual Information $I(X;Y)$ ist die "Schnittmenge" der Informationgehalte der beiden Zufallsvariablen X unf Y.

Mutual Information

Die **Mutual Information** $I(X;Y)$ ist ein nicht-negatives, symmetrisches Maß (d.h. $I(X;Y) = I(Y;X)$) der gemeinsamen Information zweier Zufallsvariablen.

$$I(X;Y) = \sum_{x \in \Omega_X} \sum_{y \in \Omega_Y} p(x,y)\ log_2\ \frac{p(x,y)}{p(x)p(y)}$$

$\rightarrow$ Ü

Die **punktweise Mutual Information** $I(x;y)$ ist ein Maß für die Korrelation zwischen zwei Ereignissen

$$I(x;y) = log_2\ \frac{p(x,y)}{p(x)p(y)}$$

Beispiel: Punktweise Mutual Information zwischen den Wörtern des Wort-Bigrammes *(Bayern, München)*:

I(Bayern;München) = $\log_2$ p(Bayern,München) / [p(Bayern) p(München)]

Relative Entropie

Die **Relative Entropie** (Kullback-Leibler-Abstand) zwischen zwei Verteilungsfunktionen $p(x)$ und $q(x)$ ist wie folgt definiert:

$$D(p||q) = \sum_{x \in \Omega_X} p(x) \, log_2 \, \frac{p(x)}{q(x)}$$

Die relative Entropie ist nie negativ, nicht symmetrisch und 0 falls $p = q$.
Sie wird oft als Abstandsmaß für Wahrscheinlichkeitsverteilungen verwendet.

Bezug zur Mutual Information: MI misst, wie weit die gemeinsame Verteilung $p(x, y)$ zweier Zufallsvariablen entfernt ist von einer statistisch unabhängigen Verteilung $p(x)p(y)$:

$$I(X; Y) = D(p(x, y) \; || \; p(x)p(y))$$

→ Ü

Relative Entropie

Die relative Entropie gibt an, wieviele Bits im Mittel verschenkt werden, wenn Ereignisse mit einer Verteilung p mit einem Code kodiert werden, der optimal für die Verteilung q ist.

Beispiel:

x	p(x)	Code(x)	q(x)
a	0.45	1	0.5
e	0.30	01	0.25
i	0.15	001	0.125
o	0.10	000	0.125

$$D(p||q) = 0.45 \log_2 \frac{0.45}{0.5} + 0.30 \log_2 \frac{0.30}{0.25} + 0.15 \log_2 \frac{0.15}{0.125} + 0.10 \log_2 \frac{0.10}{0.125} \approx 0.01777 \text{ Bit}$$

Das ist nicht dasselbe wie

$$D(q||p) = 0.5 \log_2 \frac{0.5}{0.45} + 0.25 \log_2 \frac{0.25}{0.30} + 0.125 \log_2 \frac{0.125}{0.15} + 0.125 \log_2 \frac{0.125}{0.10} \approx 0.01760 \text{ Bit}$$

$D(p||q)$ ist unendlich, falls es ein x gibt mit $p(x) > 0$ und $q(x) = 0$, d.h. falls es ein Symbol gibt, das nicht kodiert werden kann!

Crossentropie

Die **Crossentropie** zwischen zwei Verteilungen p und q

$$\begin{aligned} H(p,q) &= -\sum_x p(x)\ log_2\ q(x) \\ &= E_p(\log_2 \frac{1}{q(x)}) \\ &= H(p) + D(p||q) \end{aligned}$$

Die **Crossentropie** eines Korpus $x_1^n = x_1 x_2 ... x_n$ ist so definiert:

$$H(x_1^n, q) = -\frac{1}{n}\ log_2\ q(x_1^n)$$

Die **Perplexität** ist eng mit der Crossentropie verwandt:

$$perp(x_1^n, q) = 2^{H(x_1^n, q)}$$

Was wir oft benutzen werden

bedingte Wahrscheinlichkeit: $p(x|y) = \frac{p(x,y)}{p(y)}$

Kettenregel $p(x, y, z) = p(x)p(y|x)p(z|xy)$

Bayes'sches Theorem: $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$

Schätzung von Wahrscheinlichkeiten:

$$\tilde{p}(x) = \frac{f(x)}{N} \qquad N = \sum_{x'} f(x')$$

$$\tilde{p}(x|y) = \frac{f(x,y)}{f(y)} \qquad f(y) = \sum_{x'} f(x', y)$$

Statistische Tests

Ist die Münze gezinkt?

Zahl

Kopf

Beim Wurf einer Münze sollten die Seiten “Kopf” und “Zahl” beide die Wahrscheinlichkeit 0.5 besitzen.

Andernfalls ist die Münze **gezinkt**.

Um zu testen, ob eine Münze gezinkt ist, können wir durch wiederholtes Werfen eine **Stichprobe** erzeugen.

Angenommen wir bekommen bei 30 Würfen 20-mal “Kopf”.

Ist die Münze gezinkt und zeigt zu oft Kopf?

Das untersuchen wir mit einem **statistischen Test**.

Statistische Tests

Ähnlich wie bei einem Widerspruchsbeweis machen wir eine Annahme (Nullhypothese), die wir widerlegen wollen. Dann zeigen wir, dass die beobachteten Stichprobenresultate der Annahme widersprechen.

Nullhypothese: Die Wahrscheinlichkeit von "Kopf" ist 0.5.

Unter der Annahme der Nullhypothese **erwarten** wir ungefähr $n \cdot p = 30 * 0.5 = 15$ Mal "Kopf" zu sehen (Stichprobengröße n, Kopf-Wahrscheinlichkeit p)

Wenn die beobachtete Häufigkeit viel größer als die erwartete Häufigkeit ist, nehmen wir an, dass **die Nullhypothese falsch ist**, weil das beobachtete Resultat zu unwahrscheinlich ist, um mit der Nullhypothese erklärt werden zu können.

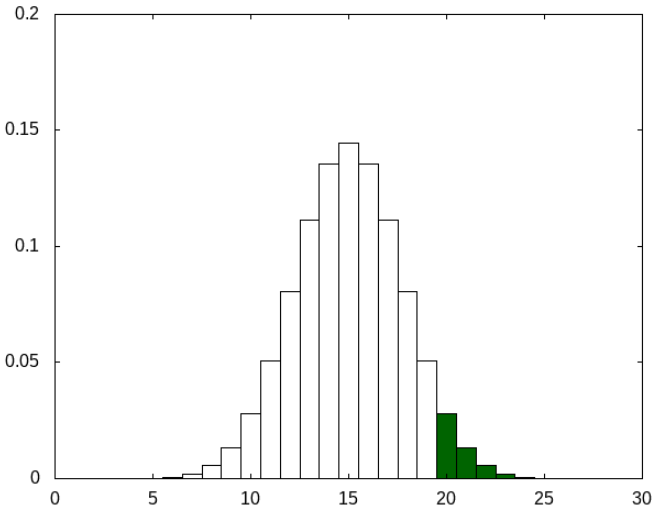
In unserem Fall würden wir daraus schließen, dass die Münze gezinkt ist.

Statistische Tests

Da auch sehr unwahrscheinliche Resultate keine Wahrscheinlichkeit von 0 haben, machen wir möglicherweise einen **Fehler**, wenn wir die Nullhypothese verwerfen.

Wir sind bereit zu akzeptieren, dass in maximal **5%** der Fälle, in denen wir die Nullhypothese verwerfen, die Nullhypothese tatsächlich doch korrekt war. Die Gesamtwahrscheinlichkeit aller Resultate, bei denen wir die Nullhypothese verwerfen, darf daher maximal 0.05 sein.

Dann sprechen wir von einem **signifikanten Ergebnis**.



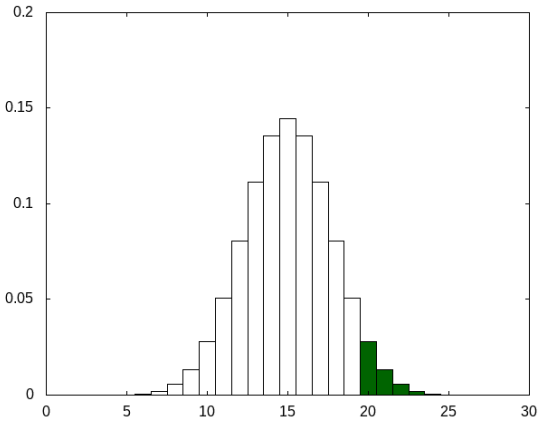
Die Grafik zeigt die Wahrscheinlichkeiten der möglichen Stichproben-Resultate unter Annahme der Nullhypothese.

Der grüne Bereich umfasst alle Resultate, die statistisch signifikant sind.

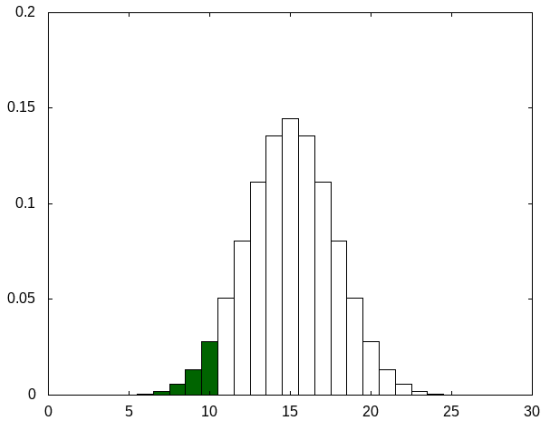
Die Summe ihrer Wahrscheinlichkeiten ist maximal 0.05.

Statistische Tests

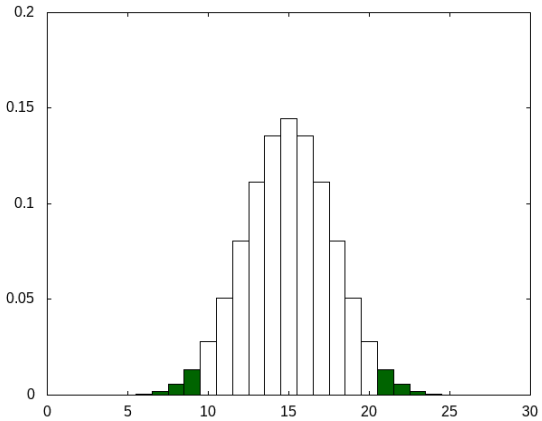
Es gibt drei Varianten von statistischen Tests:



rechtsseitiger Test



linksseitiger Test



beidseitiger Test

Beim beidseitigen Test interessieren wir uns für Abweichungen in beide Richtungen. Die Gesamtwahrscheinlichkeit muss hier auf jeder Seite kleiner als $0.05/2$ sein.

Binomialtest

Der Münzwurf ist ein Bernoulli-Experiment, bei dem bspw. das "1"-Ereignis für "Kopf" und das "0"-Ereignis für "Zahl" steht.

Die Binomialverteilung $b(r; n, p)$ gibt an, wie wahrscheinlich r viele 1-Ereignisse in einer Stichprobe der Größe n sind, wenn p die Wk. des 1-Ereignisses ist.

Die Wahrscheinlichkeit, 20 oder mehr 1-Ereignisse in n=30 Wiederholungen eines Bernoulliversuches mit der Wahrscheinlichkeit p=0.5 zu bekommen, ist somit

$$b(\geq 20; 30, 0.5) = \sum_{i=20}^{30} b(i, 30, 0.5) \approx 0.049$$

$\Rightarrow$ Das Ergebnis ist knapp **statistisch signifikant**.

$\Rightarrow$ Wahrscheinlich ist die Nullhypothese falsch und die Münze gezinkt.

$\Rightarrow$ Die Nullhypothese kann verworfen werden, wobei die Fehlerwahrscheinlichkeit mit fast 5% jedoch relativ hoch ist.

Statistische Tests

Anmerkung

Wenn das Testergebnis **nicht signifikant** lautet, bedeutet das nicht, dass die Nullhypothese wahrscheinlich stimmt!

Wir konnten nur nicht zeigen, dass die Nullhypothese **nicht** stimmt.

Mit einer größeren Stichprobe wäre eventuell ein signifikantes Ergebnis erzielt worden.

Kollokationsextraktion

Kollokationen

Kollokationen sind feste Wortkombinationen, die beim Erwerb einer Sprache gelernt werden müssen.

Häufig verwendete Kriterien (nicht immer alle erfüllt):

- **nicht kompositionell:** Die Bedeutung einer Kollokation ist nicht aus den Bedeutungen ihrer Teile ableitbar.

  to kick the bucket

- **nicht austauschbar:** Teile der Kollokation können nicht durch semantisch äquivalente Ausdrücke ersetzt werden.

  to kick the bin

- **nicht modifizierbar:**

  to kick two buckets

- **nicht wörtlich übersetzbar:**

  to kick the bucket – *den Eimer treten

Indiz für Kollokationen: Eine Kollokation ist häufiger als aufgrund der Einzelwort-Häufigkeiten zu erwarten wäre.

Häufige Wortpaare

Häufigkeit	Wort	Wort
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the

Was verbindet die Wortpaare?

Es handelt sich um Funktionswörter, die meist in einer **syntaktischen** Beziehung stehen.

Häufige Wortpaare

Häufige Adjektiv-Nomen-Paare in kanadischen Parlamentsdebatten

Häufigkeit	Adjektiv	Nomen
87655	hon.	member
20997	federal	government
12260	hon.	members
8447	private	sector
8071	last	year
7871	supplementary	question
7683	Canadian	people
7532	same	time
7511	unanimous	consent
6980	small	business
6814	great	deal
6724	federal	Government

Was verbindet die Wortpaare?

Zwischen den Wörtern besteht eine **semantische** Abhängigkeit.

Häufige Wortpaare

Lemmatisierte Verb-Nomen-Paare mit einem Maximalabstand von 5

Verb	Nomen	Häufigkeit
spielen	Rolle	28
sagen	Mann	20
stellen	Frage	18
sehen	Seite	16
geben	Grund	16
schütteln	Kopf	16
kommen	Jahr	14
geben	Zeit	14
geben	Möglichkeit	12
sagen	Frau	12
erzählen	Geschichte	12
kosten	Geld	11
kommen	Frage	11
erscheinen	Buch	11
machen	Spaß	10
tragen	Name	10

Hier finden sich einige echte Kollokationen

Häufige Wortpaare

Gründe für die Häufigkeit von Wortpaaren:

- Die Einzelwörter sind häufig.
- Es gibt syntaktische oder semantische Abhängigkeiten.
- Sie bilden eine Kollokation.

Bei der Kollokationsextraktion suchen wir Wortpaare, die häufiger sind, als auf Grund der Einzelwort-Häufigkeiten zu erwarten wäre.

Dazu wenden wir einen **statistischen Test** an.

Binomialtest

Beispiel: Ist das Wortpaar new companies signifikant häufiger als erwartet?

Daten: In einem Korpus mit n=14,307,668 Wörtern, taucht new f_{new}=15,828 Mal auf, companies $f_{companies}$=4,675 Mal und new companies 8 Mal.

Nullhypothese: Die gemeinsame Wahrscheinlichkeit p von new und companies ist nicht höher als das Produkt ihrer Randverteilungen, also maximal

$$p = \tilde{p}_{new}\,\tilde{p}_{companies} = \frac{f_{new}}{n}\,\frac{f_{companies}}{n} = \frac{15828}{14307668}\,\frac{4675}{14307668} = 3.615 \cdot 10^{-7}$$

Die Wahrscheinlichkeiten $\tilde{p}_{new}$ und $\tilde{p}_{companies}$ wurden hier mit relativen Häufigkeiten aus der Stichprobe geschätzt:

$$\tilde{p}_{new} = \frac{f_{new}}{n} \qquad \tilde{p}_{companies} = \frac{f_{companies}}{n}$$

Binomialtest

Die Wahrscheinlichkeit, 8 oder mehr 1-Ereignisse (hier Wortpaare *new companies*) in n=14,307,668 Wiederholungen eines Bernoulliversuches mit der Wahrscheinlichkeit p zu bekommen, ist somit

$$b(\geq 8, n, p) = 1 - \sum_{i=0}^{7} b(i, n, p) \approx 0.15$$

$\Rightarrow$ Das Ergebnis ist **nicht** statistisch signifikant.

Kollokationsextraktion

In der Praxis wenden wir statistische Tests bei der Kollokationsextraktion nicht an, um die **Signifikanz** zu berechnen, sondern nur um die Wortpaar-Kandidaten zu **ranken**.

Wir können dann bspw. die am höchsten gerankten Wortpaare manuell untersuchen, um echte Kollokationen zu extrahieren.

Die gleichzeitige Anwendung eines statistischen Testes auf viele Wortpaare ist auch gar nicht zulässig:

Wenn man bspw. bei 1000 Wortpaaren untersucht, ob sie signifikant häufiger auftreten, würde man ja erwarten, $1000 \cdot 0.05 = 50$ "signifikante" Wortpaare zu finden, auch wenn die Nullhypothese für alle 1000 Wortpaare stimmt.

χ^2 Test

Der χ^2-Test ist ein weiterer statistischer Test.
Wir erstellen dafür zunächst die *Kontingenztabelle*:

	$w_2 = companies$	$w_2 \neq companies$	
$w_1 = new$	$O_{11} = 8$	$O_{12} = 15820$	$O_{1-} = 15828$
$w_1 \neq new$	$O_{21} = 4667$	$O_{22} = 14287173$	$O_{2-} = 14291840$
	$O_{-1} = 4675$	$O_{-2} = 14302993$	$O_{--} = 14307668$

Dann berechnen wir die χ^2-Teststatistik:

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \quad \text{wobei } E_{ij} = p_{i-}p_{-j}O_{--} = \frac{O_{i-}O_{-j}}{O_{--}}$$

O_{ij} ist der beobachtete Wert aus der Kontingenztabelle.
E_{ij} sind die erwarteten Werte unter Annahme der Nullhypothese.

$p_{1-} = \frac{O_{1-}}{O_{--}}$ ist die Wahrscheinlichkeit, dass *new* das 1. Wort ist.

$p_{2-} = \frac{O_{2-}}{O_{--}}$ ist die Wahrscheinlichkeit, dass *new* **nicht** das 1. Wort ist.

$p_{-1} = \frac{O_{-1}}{O_{--}}$ ist die Wahrscheinlichkeit, dass *companies* das 2. Wort ist.

$p_{-2} = \frac{O_{-2}}{O_{--}}$ ist die Wahrscheinlichkeit, dass *companies* **nicht** das 2. Wort ist.

p-Wert

Beim Binomialtest gibt der berechnete Wert direkt die Wahrscheinlichkeit (= **p-Wert**) dafür an, einen Fehler zu machen, wenn wir die Nullhypothese zurückweisen.

Dagegen muss der χ^2-Wert erst mit Hilfe einer Tabelle in einen p-Wert **umgerechnet** werden.

Der χ^2-Wert misst die Abweichung von den erwarteten Werten in der Kontingenz-Tabelle. Je größer er ist, desto kleiner ist der p-Wert.

Im Beispiel erhalten wir einen χ^2-Wert von 1.55, der einem p-Wert von 0.21 entspricht. Das Ergebnis ist also auch bei diesem Test **nicht signifikant**.

Wie die Ergebnisse zeigen, können sich die p-Werte verschiedener statistischer Tests deutlich unterscheiden.

Weitere Wortassoziationsmaße

- t-Score:

$$t = \frac{O_{11} - E_{11}}{\sqrt{O_{11}}}$$

- Log-Likelihood Ratio

$$L = 2 \sum_{ij} O_{ij} \log_2 \frac{O_{ij}}{E_{ij}}$$

- punktweise Mutual Information

$$MI = \log_2 \frac{O_{11}}{E_{11}}$$

(Siehe auch die Dissertation von Stefan Evert.)

Herleitung der Formel für den t-Score

Wenn der t-Test für den Vergleich **einer Stichprobe** $x_1, ..., x_n$ mit (auf Basis einer Nullhypothese) erwarteten Werten verwendet werden soll, nimmt man die Formel:

$$t = \sqrt{n}\ \frac{\bar{x} - \mu_0}{s}$$

Es gilt: $\bar{x} = O_{11}/n, \quad \mu_0 = E_{11}/n.$

Für die Standardabweichung s der Bernoulliverteilung gilt die Formel:

$$s = \sqrt{p(1-p)} \approx \sqrt{p} = \sqrt{O_{11}/n}$$

da $(1 - p) \approx 1$.

Somit gilt:

$$t \approx \sqrt{n}\ \frac{O_{11} - E_{11}}{n\sqrt{O_{11}/n}} = \frac{O_{11} - E_{11}}{\sqrt{O_{11}}}$$

Kurzzusammenfassung: Binomialtest

gegeben: die Häufigkeiten f(x,y), f(x) und f(y) der Wörter x und y im Trainingskorpus der Größe n

Nullhypothese: Das Wortpaar (x,y) ist nicht wahrscheinlicher, als bei statistischer Unabhängigkeit zu erwarten ist, also

$$p(x,y) \leq p(x)p(y) = \frac{f(x)}{n}\frac{f(y)}{n}$$

Das Wortpaar ist **signifikant** häufiger, falls

$$b(\geq f(x,y), p(x)p(y), n) < 0.05$$

Markowmodelle: Sprachidentifizierung

Sprachmodelle und Sprachidentifizierung

Sprachmodelle werden oft in der Sprachverarbeitung verwendet (Spracherkennung, maschinelle Übersetzung, Sprachidentifizierung).

Was ist die Sprache des folgenden Textes?

Decidiu guardar suas fichas para nova oportunidade.

Spanisch, Italienisch, Portugiesisch, Rumänisch? ⇒ Portugiesisch

Wir werden für die **Sprachidentifizierung** ein statistisches Modell verwenden, welches vor allem Häufigkeiten von Buchstaben-NGrammen als Information nutzt:

Decid/ecidi/cidiu/idiu /diu g/diu g/iu gu/u gua/guar...

Wir werden dieses Modell nun mathematisch herleiten.

Sprachmodelle und Sprachidentifizierung

Sprachmodelle werden oft in der Sprachverarbeitung verwendet (Spracherkennung, maschinelle Übersetzung, Sprachidentifizierung).

Was ist die Sprache des folgenden Textes?

Decidiu guardar suas fichas para nova oportunidade.

Spanisch, Italienisch, Portugiesisch, Rumänisch? $\Rightarrow$ Portugiesisch

Wir werden für die **Sprachidentifizierung** ein statistisches Modell verwenden, welches vor allem Häufigkeiten von Buchstaben-NGrammen als Information nutzt:

Decid/ecidi/cidiu/idiu /diu g/diu g/iu gu/u gua/guar...

Wir werden dieses Modell nun mathematisch herleiten.

Sprachmodelle und Sprachidentifizierung

Ein **Sprachidentifizierer** (Language Guesser) berechnet die wahrscheinlichste Sprache $\hat{L}$ eines gegebenen Textes T:

$$\hat{L} = \arg\max_{L} p(L|T)$$

Es ist unmöglich, die Wahrscheinlichkeit $p(L|T)$ einer Sprache L für einen beliebigen Text direkt zu schätzen, weil es unendlich viele Texte und damit Wahrscheinlichkeiten gibt.

Statistische Modellierung

Trick: Anwendung des Bayes'schen Theorems:

$$\arg\max_L p(L|T) = \arg\max_L \frac{p(T|L)p(L)}{p(T)}$$

Die Textwahrscheinlichkeit $p(T)$ ist eine Konstante, die keinen Einfluss auf das Ergebnis der argmax-Operation hat und daher weggelassen werden kann:

$$\arg\max_L p(L|T) = \arg\max_L p(T|L)p(L)$$

Falls keine Information über die Apriori-Wahrscheinlichkeiten $p(L)$ der Sprachen verfügbar ist, können wir sie als gleichverteilt annehmen und ebenfalls ignorieren:

$$\arg\max_L p(L|T) = \arg\max_L p(T|L)$$

Wenn eine repräsentative Stichprobe von Texten existiert, kann $p(L)$ geschätzt werden, indem die Zahl der Texte in der Sprache L durch die Gesamtzahl der Texte geteilt wird.

Markowmodelle

Wir nehmen an, dass der Text T aus der Zeichenfolge $a_1, a_2, ..., a_n =: a_1^n$ besteht, und zerlegen $p(T|L)$ in ein Produkt von bedingten Wahrscheinlichkeiten:

$$\begin{aligned} p(T|L) &= p(a_1^n|L) = p_L(a_1^n) = p_L(a_1, ..., a_n) \\ &= p_L(a_1)p_L(a_2|a_1)p_L(a_3|a_1, a_2)...p_L(a_n|a_1, ..., a_{n-1}) \\ &= \prod_{i=1}^{n} p_L(a_i|a_1, ..., a_{i-1}) \end{aligned}$$

Wir verwenden hier für $p(a_1^n|L)$ die alternative Notation $p_L(a_1^n)$. $p_L(\cdot)$ ist ein *Sprachmodell*.

Problem: Die Kettenregel ist eigentlich nur anwendbar, wenn alle Texte dieselbe Länge n besitzen. Bei variablem n ist $\prod_{i=1}^{n} p_L(a_i|a_1, ..., a_{i-1})$ nur die Gesamtwahrscheinlichkeit aller Strings mit **Präfix** a_1^n.

Daher hätte *"Er liest ein Buc"* eine größere Wk. als *"Er liest ein Buch"*.

Wir lösen das Problem durch Hinzufügen einer eindeutigen Endemarkierung $a_{n+1} = \langle /s \rangle$:

$$p_L(a_1^n) = \prod_{i=1}^{n+1} p_L(a_i|a_1, ..., a_{i-1})$$

Endesymbol

Die Notwendigkeit eines Endesymboles zeigt auch das folgenden Beispiel:

Wir betrachten die Menge aller Zeichenfolgen, die nur aus x bestehen, also x, xx, xxx, xxxx, usw.

Da x das einzig mögliche Zeichen ist, muss seine Wahrscheinlichkeit 1 sein.

Gemäß der Formel $p(a_1, ..., a_n) = \prod_{i=1}^{n} p(a_i|a_1, ..., a_{i-1})$ gilt somit:

$p(x) = 1$
$p(xx) = 1*1=1$
$p(xxx) = 1*1*1=1$
$p(xxxx) = 1*1*1*1=1$

Die Summe der Wahrscheinlichkeiten aller möglichen Zeichenfolgen ist daher unendlich. $p(a_1, ..., a_n)$ ist somit keine Wahrscheinlichkeitsverteilung.

Wenn wir ein Endesymbol $\langle/s\rangle$ hinzufügen und ihm eine Wahrscheinlichkeit $p(\langle/s\rangle)$ geben, reduziert sich die Wahrscheinlichkeit $p(x)$ zu $1 - p(\langle/s\rangle)$.

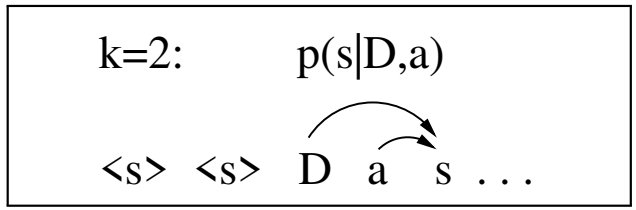
Die Summe der Wahrscheinlichkeiten aller möglichen Zeichenfolgen (inkl. ϵ) beträgt nun:

$$p(\langle/s\rangle) + p(x)p(\langle/s\rangle) + p(x)p(x)p(\langle/s\rangle) + ... = p(\langle/s\rangle) \sum_{n=0}^{\infty} p(x)^n = \frac{p(\langle/s\rangle)}{1-p(x)} = 1$$

Markowmodelle

Wir machen die **vereinfachende** Annahme, dass a_i nur von den k vorhergehenden Zeichen abhängt

$$p_L(T) = \prod_{i=1}^{n+1} p_L(a_i|a_{i-k}...a_{i-1})$$



Ein solches Modell heißt **Markowmodell** der Ordnung k.

Für $k = 2$ bekommen wir:
p(Er ölt) =
p(E|⟨s⟩,⟨s⟩) p(r|⟨s⟩,E) p(_ |E,r) p(ö|r,_) p(l|_,ö) p(t|ö,l) p(⟨/s⟩|l,t)

⇒ Es werden k Startsymbole $\langle s \rangle$ und ein Endesymbol $\langle /s \rangle$ hinzugefügt:
$a_{-1} = a_0 = \langle s \rangle$ und $a_{n+1} = \langle /s \rangle$

Die Startsymbole sind notwendig, damit $p(a_1|a_{-1}, a_0)$ definiert ist.
Ohne das Endesymbol wäre die Summe der Wahrsch. aller Zeichenfolgen größer als 1.

Markowmodelle

Für zunehmende Ordnung k modellieren Markowmodelle die Sprache immer genauer und die Crossentropie von Testdaten nimmt ab:

Modell	Crossentropie
Uniforme[1] Verteilung	4.76 Bit ($= \log_2 27$)
Ordnung 0	4.03 Bit
Ordnung 1	2.8 Bit
Mensch	1.3 Bit (Shannon)

Aber, bei Modellen höherer Ordnung nimmt die Zahl der Wk.-Parameter schnell zu, und es wird immer schwieriger, deren Werte zu schätzen.

In der Praxis ist k ein Hyperparameter, der auf Development-Daten optimiert werden kann.

[1] Eine **uniforme** Verteilung über 27 Buchstaben weist jedem Buchstaben die Wahrscheinlichkeit 1/27 zu.

Markowmodelle über Wortfolgen

Markowmodelle können nicht nur über Zeichenfolgen definiert werden, sondern auch über Wortfolgen. Markowmodelle über Wortfolgen werden sogar häufiger benutzt.

Für $k = 1$ bekommen wir:

$p(Peter, liebt, Maria) =$
$p(Peter|\langle s \rangle)\ p(liebt|Peter)\ p(Maria|liebt)\ p(\langle /s \rangle|Maria)$

$\Rightarrow$ Experiment

Parameterschätzung

Die Parameter werden mit relativen Häufigkeiten aus Trainingsdaten geschätzt:

$$p(c|Spra) = \frac{f(Sprac)}{\sum_x f(Sprax)}$$

Ein solches Modell heißt Markowmodell 4. Ordnung, oder auch 5-Gramm-Modell, weil wir die Häufigkeiten von 5-Grammen brauchen, um die Parameter zu schätzen.

Falls das 5-Gramm *Sprac* nicht in den Trainingsdaten auftaucht, wird seine Wahrscheinlichkeit mit 0 geschätzt!

⇒ Die Wk. der gesamten Buchstabenfolge wird 0,
egal wie gut die anderen Textteile modelliert werden.

⇒ Sparse-Data Problem

⇒ Notwendigkeit der Parameterglättung (später behandelt)

Sprachidentifizierung

Vorgehensweise

- Training
  - Man legt die Menge der zu erkennenden Sprachen fest.
  - Man sammelt für jede dieser Sprachen ein **Textkorpus**.
  - Man trainiert für jede Sprache ein Markowmodell auf dem entsprechenden Korpus
    d.h. man **schätzt** seine Parameter aus dem Korpus.
- Anwendung
  - Man berechnet die **Wahrscheinlichkeit des Textes** für jedes Sprachmodell (als Produkt bedingter Wahrscheinlichkeiten).
    (Falls bekannt, multipliziert man noch die Apriori-Wk. p(L) der Sprache.)
  - Man gibt die Sprache mit der größten **Wahrscheinlichkeit** aus.

Sprachidentifizierung

Beispiele:

Das ist ein deutscher Satz.

Modell	Bits/Symbol
Dänisch	4.048654
Niederländisch	3.027868
Englisch	3.604354
Finnisch	3.976386
Französisch	3.426143
Deutsch	**1.754679**
Italienisch	4.256318
Portugiesisch	4.362893
Spanisch	4.314057
Schwedisch	3.879200

This is an English sentence.

Modell	Bits/Symbol
Dänisch	3.588310
Niederländisch	3.558052
Englisch	**1.578621**
Finnisch	4.080841
Französisch	3.091803
Deutsch	2.806066
Italienisch	3.980872
Portugiesisch	3.839128
Spanisch	3.833479
Schwedisch	3.492607

Statt der Wahrscheinlichkeit $p(a_1^n)$ wird hier die Crossentropie gezeigt:

$$H(a_1^n, p) = -\frac{1}{n} log_2 p(a_1^n) \text{ Bits/Symbol}$$

Kurzzusammenfassung: Markowmodelle

Ein Markowmodell definiert die Wahrscheinlichkeit einer Wortfolge (Buchstabenfolge) $x_1, ..., x_n$ als Produkt bedingter Wahrscheinlichkeiten:

$$p(x_1, ..., x_n) = \prod_{i=1}^{n+1} p(x_i | x_{i-k}, ..., x_{i-1})$$

k ist die Ordnung des Markowmodelles.

Am Anfang der Folge werden k Startsymbole hinzugefügt und am Ende ein Endesymbol x_{n+1}.

Die Parameter werden aus (k+1)-Gramm-Häufigkeiten geschätzt:

$$p(x_{k+1} | x_1, ..., x_k) = \frac{f(x_1, ..., x_{k+1})}{\sum_x f(x_1, ..., x)}$$

Zur Vermeidung von Nullwahrscheinlichkeiten ist eine Parameterglättung nötig.

Beispiel mit $k = 1$: $p(ABBA) = p(A|\langle s \rangle)p(B|A)p(B|B)p(A|B)p(\langle /s \rangle|A)$

Parameterglättung

Parameterschätzung

Die Parameter werden gewöhnlich mit relativen Häufigkeiten aus Trainingsdaten geschätzt.

Beispiel: Wort n-Gramm-Wahrscheinlichkeiten

$$p_{MLE}(w_n|w_1 \ldots w_{n-1}) = \frac{f(w_1 \ldots w_n)}{\sum_w f(w_1 \ldots w_{n-1} w)}$$

Dies ist die **Maximum-Likelihood-Schätzung** (MLE) bzgl. der Trainingsdaten. Keine andere Wahl der Parameter weist den Trainingsdaten eine höhere Wahrscheinlichkeit zu.

Aber: Neue Texte können n-Gramme enthalten, die nicht in den Trainingsdaten auftauchten. Ihre relative Häufigkeit ist 0.

Anm.: Im Englischen wird der Begriff *Likelihood* verwendet, wenn man von den Wahrscheinlichkeiten spricht, die unterschiedliche Modelle denselben Daten zuweisen.

Parameterglättung

Nullwahrscheinlichkeiten müssen vermieden werden, außer das entsprechende n-Gramm ist unmöglich.

Prinzip der Parameterglättung: Man nimmt den beobachteten n-Grammen etwas Wahrscheinlichkeit weg und verteilt sie an die nicht beobachteten n-Gramme.

Notation

Die Notation der Formeln folgt in der Regel folgenden Konventionen:

w steht für ein Wort

w_i steht für das i-te Wort einer Wortfolge

w_i^k steht für die Teil-Wortfolge $w_i, w_{i+1}, ..., w_k$

t steht für ein Tag

w' wird verwendet, wenn eine neue Variable benötigt wird, die von w verschieden ist. w und w' können dasselbe Wort bezeichnen, auch wenn die Variablen verschieden sind! Das Apostroph $'$ dient hier allein dazu, die Variable w' als von w verschieden zu markieren. Man könnte statt w' auch bspw. z schreiben, aber dann wäre nicht erkennbar, dass die Variable für ein Wort steht. Auch w'' und $\hat{w}$ werden in derselben Weise verwendet.

Addiere-1 Glättung (Laplace-Glättung)

Bei der **Addiere-1** Glättung wird zu allen Häufigkeiten (auch Nullhäufigkeiten) der Wert 1 addiert:

$$p_{Add1}(w) = \frac{f(w) + 1}{N + B}$$

N = Zahl der Wort-Tokens B = Anzahl der Wort-Types

Die Addiere-1 Glättung ist **optimal**, falls die uniforme Verteilung am wahrscheinlichsten ist, was in der Sprachverarbeitung selten der Fall ist, weil hier Zipf'sche Verteilungen dominieren.

Die Addiere-1 Glättung **überschätzt** daher die Wahrscheinlichkeit ungesehener Wörter.

Addiere-λ Glättung

reduziert das Ausmaß der Glättung

$$p_{Add\lambda}(w) = \frac{f(w) + \lambda}{N + B\lambda} \quad \text{mit } 0 < \lambda < 1$$

Die Addiere-λ Glättung ist äquivalent zu einer Interpolation der relativen Häufigkeit f(w)/N mit einer uniformen Verteilung 1/B.

$$p_{Add\lambda}(w) = \mu\frac{f(w)}{N} + (1-\mu)\frac{1}{B} \quad \text{wobei } \mu = \frac{N}{N + B\lambda}$$

$\Rightarrow$ Addiere-λ Glättung reduziert die Wahrscheinlichkeit von gesehenen Wörtern um einen Betrag, der linear mit der Wahrscheinlichkeit steigt.

$\Rightarrow$ "Linear Discounting"

Übung: Zeigen Sie die Äquivalenz zur ersten Formel, indem Sie die Definition von μ in die zweite Formel einsetzen und umformen.

Glättungsexperiment

Wir wollen nun untersuchen, wie gut die geglätteten Wahrscheinlichkeiten mit den tatsächlichen Wahrscheinlichkeiten der Wörter übereinstimmen.

Dazu verwenden wir ein Trainingskorpus und ein Testkorpus derselben Größe und vergleichen die Worthäufigkeiten im Testkorpus mit den erwarteten Worthäufigkeiten $f(w) = p(w) \cdot n$, wobei n die Korpusgröße und $p(w)$ die aus den Trainingsdaten geschätzte Wahrscheinlichkeit ist.

Um zuverlässige Aussagen treffen zu können, betrachten wir nicht die Häufigkeiten einzelner Wörter, sondern fassen alle Wörter derselben Häufigkeit im Trainingskorpus zusammen und berechnen deren mittlere Häufigkeit im neuen Korpus.

Wir werden statt einzelner Wörter jedoch Wortpaare betrachten.

Glättungsexperiment

$f_{emp}(i)$ ist die mittlere Häufigkeit im Korpus C_2 von all denjenigen Bigrammen, die im Korpus C_1 i-mal aufgetreten sind, wobei $|C_1| = |C_2|$.

$$f_{emp}(i) = \frac{\sum_{x:f_1(x)=i} f_2(x)}{|\{x|f_1(x) = i\}|}$$

$$f_{add1}(i) = |C_2|\ p_{add1}(x) \quad \text{falls } f_1(x) = i$$

i	$f_{emp}(i)$	$f_{add1}(i)$
0	0.000027	0.000137
1	0.448	0.000274
2	1.25	0.000411
3	2.24	0.000548
4	3.23	0.000685
5	4.21	0.000822
6	5.23	0.000959
7	6.21	0.00109
8	7.21	0.00123
9	8.26	0.00137

$f_1(x)$ ist die Häufigkeit des Bigrammes x in C_1

$f_2(x)$ ist die Häufigkeit des Bigrammes x in C_2

Beobachtungen:

- Die f_{add1}-Werte stimmen nicht mit der Wirklichkeit überein.
- Die Differenz zwischen i und $f_{emp}(i)$ ist annähernd konstant für $i > 1$.
  $\Rightarrow$ "Absolute Discounting"

Absolute Discounting

Linear Discounting (Add-λ Glättung) führt zu schlechten Ergebnissen, wenn die Daten eine Zipf'sche Verteilung aufweisen.

Absolute Discounting

- subtrahiert einen festen Betrag δ (Discount) von den Häufigkeiten der beobachteten Ereignisse
- verteilt die Discounts über die unbeobachteten Ereignisse.

$$p(w) = \begin{cases} \frac{f(w)-\delta}{N} & \text{falls } f(w) > 0 \\ \frac{(B-N_0)\delta}{N_0 N} & \text{sonst} \end{cases}$$

B: Zahl der Types
N: Zahl der Tokens
N_i: Zahl der Types mit Häufigkeit i

$B - N_0$	$=$	Anzahl der Discount-Operationen
$(B - N_0)\delta$	$=$	Summe der Discounts
$\frac{(B - N_0)\delta}{N}$	$=$	gesamte umverteilte Wahrscheinlichkeit
$\frac{(B - N_0)\delta}{N_0 N}$	$=$	Wk. eines einzelnen unbekannten Wortes

Absolute Discounting

Kneser, Essen, Ney haben die folgende Formel für den Discountbetrag δ hergeleitet:

$$\delta = \frac{N_1}{N_1 + 2N_2}$$

Beispiel für Absolute Discounting

Ein Korpus mit 100 000 Tokens, in dem 500 Wörter einmal und 250 Wörter zweimal aufgetreten sind, enthält das Wort **Brief** 10 Mal und das Wort **Reifenflickzeug** 0 Mal. Es gibt insgesamt 2000 verschiedene Wörter (z.B. aus einem Wörterbuch), von denen 1000 nicht im Korpus aufgetreten sind.

$\delta = 500/(500 + 2 \cdot 250) = 0.5$
$p(\text{Brief}) = (10 - 0.5)/100000 = 0.000095$
$p(\text{Reifenflickzeug}) = (2000 - 1000) \cdot 0.5/(1000 \cdot 100000) = 0.000005$

Backoff-Glättung

Bei bedingten Wahrscheinlichkeiten wie p(Uni|an,der) könnte man versuchen, das Problem der Nullwahrscheinlichkeiten so zu lösen:

$$p(Uni|an,der) = \begin{cases} \frac{f(an,der,Uni)}{\sum_x f(an,der,x)} & \text{falls } f(an,der,Uni) > 0 \\ \frac{f(der,Uni)}{\sum_x f(der,x)} & \text{falls } f(an,der,Uni) = 0, f(der,Uni) > 0 \\ \frac{f(Uni)}{\sum_x f(x)} & \text{sonst} \end{cases}$$

Aber: Die Wahrscheinlichkeiten aller Wörter, die nach "an der" aufgetreten sind, (1. Fall) addieren bereits zu 1. Dazu kommen nun noch die Wahrscheinlichkeiten der dort nicht aufgetretenen Wörter (Fälle 2 und 3). Also wird die Summe größer als 1.

Das darf nicht sein! Bei der Parameterglättung müssen wir immer den beobachteten Wörtern etwas von ihrer Wahrscheinlichkeit wegnehmen.

Katz'sche Backoff-Glättung

Wir kombinieren daher die Idee der Backoff-Glättung mit Absolute Discounting:

$$p(w_i|w_{i-k}^{i-1}) = \begin{cases} \frac{f(w_{i-k}^{i})-\delta_k}{\sum_x f(w_{i-k}^{i-1}x)} & \text{falls } f(w_{i-k}^{i}) > 0 \\ \alpha(w_{i-k}^{i-1})\ p(w_i|w_{i-k\underline{+1}}^{i-1}) & \text{sonst} \end{cases}$$

Der Backoff-Faktor $\alpha(w_{i-k}^{i-1})$ wird so gewählt, dass sich eine Wk.-Verteilung ergibt.

Die Backoff-Verteilung $p(w_i|w_{i-k+1}^{i-1})$ besitzt einen kleineren Kontext (ohne w_{i-k}) und wird rekursiv auf dieselbe Weise geglättet.

Die Unigramm-Wahrscheinlichkeiten $p(w_i)$ werden entweder mit relativen Häufigkeiten $f(w_i)/\sum_x f(x)$ geschätzt oder rekursiv mit einer uniformen Verteilung $p_{\text{uniform}}(w_i) = 1/B$ geglättet.

Für jede Kontext-Größe k wird ein eigener Discount δ_k berechnet.

Backoff-Faktor für Katz'sche Backoff-Glättung

Der Backoff-Faktor $\alpha(C)$ stellt sicher, dass die Wahrscheinlichkeiten zu 1 summieren: (Es gilt $C = w_1^{n-1}$ und $C' = w_2^{n-1}$ und $f(C) = \sum_w f(C, w)$)

$$\sum_w p(w|C) = \underbrace{\sum_{w:f(C,w)>0} \frac{f(C,w)-\delta}{f(C)}}_{bekannt} + \underbrace{\sum_{w:f(C,w)=0} \alpha(C)p(w|C')}_{unbekannt} = 1$$

$p(w|C')$ ist hier die Backoff-Verteilung. Durch Umformen erhalten wir:

$$\sum_{w:f(C,w)=0} \alpha(C)p(w|C') = 1 - \sum_{w:f(C,w)>0} \frac{f(C,w)-\delta}{f(C)}$$

Ausklammern von $\alpha(C)$ und Division durch die restliche Summe liefert:

$$\alpha(C) = \frac{1 - \sum_{w:f(C,w)>0} \frac{f(C,w)-\delta}{f(C)}}{\sum_{w:f(C,w)=0} p(w|C')}$$

Dies ist äquivalent zu:

$$\alpha(C) = \frac{1 - \sum_{w:f(C,w)>0} \frac{f(C,w)-\delta}{f(C)}}{1 - \sum_{w:f(C,w)>0} p(w|C')}$$

Backoff-Glättung mit Interpolation

Hier wird die mit α gewichtete Backoff-Wahrscheinlichkeit hinzuaddiert.

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{\sum_x f(w_{i-k}^{i-1}x)} + \alpha(w_{i-k}^{i-1})\ p(w_i|w_{i-k+1}^{i-1})$$

Die max-Operation verhindert, dass der vordere Term negativ wird.

$\Rightarrow$ Interpolierte Modelle sind meistens etwas besser und die Berechnung des Backoff-Faktors ist einfacher.

Backoff-Faktor für interpolierte Backoff-Glättung

Der Backoff-Faktor $\alpha(C)$ stellt sicher, dass die Wahrsch. zu 1 summieren:

$$\sum_w p(w|C) = \sum_w \frac{max(0, f(C,w) - \delta)}{f(C)} + \alpha(C)p(w|C') = 1$$

mit $C = w_1^{n-1}$ und $C' = w_2^{n-1}$ und $f(C) = \sum_x f(C,x)$

Durch Umformen erhalten wir:

$$\sum_w \alpha(C)p(w|C') = 1 - \sum_w \frac{max(0, f(C,w) - \delta)}{f(C)}$$

Ausklammern von $\alpha(C)$ liefert:

$$\alpha(C) \underbrace{\sum_w p(w|C')}_{=1} = 1 - \sum_w \frac{max(0, f(C,w) - \delta)}{f(C)}$$

Dies ist äquivalent zu:

$$\alpha(C) = 1 - \sum_{w:f(C,w)>0} \frac{f(C,w) - \delta}{f(C)}$$

Interpolierte Backoff-Glättung

Beispiel:

$$\begin{aligned} p(\text{Buch} \mid \text{das, rote}) = {} & p^*(\text{Buch} \mid \text{das, rote}) + \alpha(\text{das, rote}) \; (\\ & p^*(\text{Buch} \mid \text{rote}) + \alpha(\text{rote}) \; (\\ & p^*(\text{Buch}))) \end{aligned}$$

$p^*(\text{Buch} \mid \text{das, rote}) = (f(\text{das, rote, Buch}) - \delta_2) \; / \; \sum_x f(\text{das, rote, x})$

$p^*(\text{Buch} \mid \text{rote}) = (f(\text{rote, Buch}) - \delta_1) \; / \; \sum_x f(\text{rote, x})$

$p^*(\text{Buch}) = f(\text{Buch}) \; / \; \sum_x f(\text{x})$

wobei

$f(\text{rote, Buch}) = \sum_x f(\text{x, rote, Buch})$

$f(\text{Buch}) = \sum_x f(\text{x, Buch})$

Bessere Backoff-Verteilungen

Problem:

Im *San Francisco Chronicle* ist das Wortpaar **San Francisco** sehr häufig und die beiden Einzelwörter **San** und **Francisco** sind ähnlich häufig.

Wenn nun das Wort **sunny** weder vor **San** noch vor **Francisco** aufgetaucht ist, würde bei Backoff-Glättung gelten:

$p(\text{San}|\text{sunny}) = \alpha(\text{sunny})\ p(\text{San})$
$p(\text{Francisco}|\text{sunny}) = \alpha(\text{sunny})\ p(\text{Francisco})$

Beide Wahrscheinlichkeiten wären also annähernd gleich.

Aber: p(Francisco|sunny) sollte viel kleiner sein, weil *Francisco* fast nur nach *San* erscheint, während *San* in vielen Kontexten auftaucht.

Kneser-Ney Backoff-Verteilung

Je mehr Kontexte es gibt, in denen ein Wort aufgetaucht ist,

- desto eher erscheint es in einem neuen Kontext
- desto größer sollte seine Backoff-Wahrscheinlichkeit sein.

$\Rightarrow$ Wähle die Backoff-Wahrscheinlichkeit proportional zur Zahl der unterschiedlichen Kontexte, in denen das Wort aufgetreten ist.

Kneser-Ney Backoff-Verteilung

Bei der bisherigen Berechnung der n-1-Gramm-Häufigkeiten zur Schätzung der Backoff-Verteilung summieren wir die Häufigkeiten über alle möglichen Vorgängerwörter w':

$$f(C, w) = \sum_{w'} f(w', C, w)$$

C ist eine (eventuell leere) Folge von Wörtern.

Bei Kneser-Ney zählen wir, wieviele **unterschiedliche** Wörter vor dem Wort-n-Gramm aufgetreten sind:

$$f^*(C, w) = \sum_{w'} \mathbf{1}_{f(w', C, w) > 0}$$

$\mathbf{1}_{test}$ ist 1, falls *test* wahr ist und sonst 0.

Die Kneser-Ney-Methode zählt n-Gramm-Types (statt -Tokens).

Aus den so ermittelten Häufigkeiten, werden dann die Parameter der Backoff-Wahrscheinlichkeits-Verteilungen geschätzt.

$$p_{backoff}(w|C) = \frac{f^*(C, w)}{\sum_{w'} f^*(C, w')}$$

Beispiel

Gegebene Häufigkeiten

f(a,a) = 1 f(b,a) = 0
f(a,b) = 2 f(b,b) = 1

Discount-Berechnung

$N_1 = 2$ $\delta = 2 / (2 + 2*1) = 0.5$
$N_2 = 1$

Berechnung der Backoff-Wahrscheinlichkeitsverteilung

Standard-Backoff-Verfahren

f(a) = f(a,a) + f(b,a) = 1
f(b) = f(a,b) + f(b,b) = 3
p(a) = f(a) / (f(a) + f(b)) = 1/4
p(b) = f(b) / (f(a) + f(b)) = 3/4

Kneser-Ney-Verfahren

$f^*(a) = |\{(a, a)\}| = 1$
$f^*(b) = |\{(a, b), (b, b)\}| = 2$
$p(a) = f^*(a) / (f^*(a) + f^*(b)) = 1/3$
$p(b) = f^*(b) / (f^*(a) + f^*(b)) = 2/3$

Berechnung der relativen Häufigkeiten mit Discount (Standard)

r(a|a) = max(0,f(a,a)-δ)/(f(a,a)+f(a,b)) = max(0, (1-0.5))/(1+2) = 1/6
r(b|a) = max(0,f(a,b)-δ)/(f(a,a)+f(a,b)) = max(0, (2-0.5))/(1+2) = 3/6

r(a|b) = max(0,f(b,a)-δ)/(f(b,a)+f(b,b)) = max(0, (0-0.5))/(0+1) = 0
r(b|b) = max(0,f(b,b)-δ)/(f(b,a)+f(b,b)) = max(0, (1-0.5))/(0+1) = 0.5

Beispiel

Berechnung der Backoff-Faktoren (interpolierte Glättung)

α(a) = 1 - r(a|a) - r(b|a) = 1 - 1/6 - 3/6 = 1/3
α(b) = 1 - r(a|b) - r(b|b) = 1 - 0 - 0.5 = 1/2

Berechnung der geglätteten Wahrscheinlichkeiten

p(a|a) = r(a|a) + α(a) p(a) = 1/6 + 1/3 * 1/3 = 5/18
p(b|a) = r(b|a) + α(a) p(b) = 3/6 + 1/3 * 2/3 = 13/18

p(a|b) = r(a|b) + α(b) p(a) = 0 + 1/2 * 1/3 = 1/6
p(b|b) = r(b|b) + α(b) p(b) = 1/2 + 1/2 * 2/3 = 5/6

Evaluierung

Wenn man ein neues Verfahren entwickelt hat, vergleicht man es mit existierenden Verfahren, um beurteilen zu können, ob es besser ist.

Sprachmodelle werden meist auf Basis der **Crossentropie** (oder der **Perplexität**) auf Testdaten miteinander verglichen.

$$H(x_1^n, p) = -\frac{1}{n} \, log_2 \, p(x_1^n)$$

Das Modell mit der geringsten Crossentropie/Perplexität ist das beste.

Die Testdaten x_1^n müssen **neue** Daten sein (nicht im Training verwendet).

Kurzzusammenfassung: Katz'sche Backoff-Glättung

gegeben: n-Gramm-Häufigkeiten $f(x_1^n)$, $N_i(n)$ = Zahl der n-Gramme mit Häufigkeit i

Berechnung relativer Häufigkeiten mit **Absolute Discounting**

$$r(x_n|x_1^{n-1}) = \frac{f(x_1^n) - \delta_n}{\sum_x f(x_1^{n-1}x)} \quad \text{mit } \delta_n = \frac{N_1(n)}{N_1(n) + 2N_2(n)}$$

Rekursive **Backoff-Glättung** bedingter Wahrscheinlichkeiten:

$$p(x_n|x_1^{n-1}) = \begin{cases} r(x_n|x_1^{n-1}) & \text{falls } f(x_1^n) > 0 \\ \alpha(x_1^{n-1})p(x_n|x_2^{n-1}) & \text{sonst} \end{cases}$$

Backoff-Faktor:

$$\alpha(x_1^{n-1}) = \frac{1 - \sum_{x \in O} r(x|x_1^{n-1})}{1 - \sum_{x \in O} p(x|x_2^{n-1})}$$

wobei $O = \{x|f(x_1^{n-1}, x) > 0\}$ die Menge der beobachten Symbole ist

Kurzzusammenfassung: Interpolierte Backoff-Glättung

gegeben: n-Gramm-Häufigkeiten $f(x_1^n)$, $N_i(n)$ = Zahl der n-Gramme mit Häufigkeit i

Berechnung relativer Häufigkeiten mit **Absolute Discounting**

$$r(x_n|x_1^{n-1}) = \frac{max(0, f(x_1^n) - \delta_n)}{\sum_x f(x_1^{n-1}x)} \quad \text{mit } \delta_n = \frac{N_1(n)}{N_1(n) + 2N_2(n)}$$

Rekursive **Backoff-Glättung** bedingter Wahrscheinlichkeiten:

$$p(x_n|x_1^{n-1}) = r(x_n|x_1^{n-1}) + \alpha(x_1^{n-1})p(x_n|x_2^{n-1})$$

Backoff-Faktor:

$$\alpha(x_1^{n-1}) = 1 - \sum_{x \in O} r(x|x_1^{n-1})$$

wobei $O = \{x|f(x_1^{n-1}, x) > 0\}$ die Menge der beobachten Symbole ist

Kurzzusammenfassung: Kneser-Ney-Glättung

"Normale" Berechnung der (n-1)-Gramm-Häufigkeiten für die Backoff-Verteilung aus n-Gramm-Häufigkeiten:

$$f(x_2^n) = \sum_x f(x\, x_2^n)$$

Hier wird gezählt, wieviele **Tokens** vor x_2^n aufgetaucht sind.

Berechnung der (n-1)-Gramm-Häufigkeiten beim **Kneser-Ney**-Verfahren:

$$f(x_2^n) = |\{x | f(x\, x_2^n) > 0\}|$$

Hier wird gezählt, wieviele unterschiedliche **Types** vor x_2^n aufgetaucht sind.

Naïve-Bayes-Modelle: Wortbedeutungs-Desambiguierung

Wortbedeutungen

Wörter sind oft mehrdeutig (ambig):

Bank (Institut vs. Sitzbank)
Hahn (Wasserhahn vs. Tier)
Maus (Tier vs. Computergerät)
blau (Farbe vs. Betrunkenheit)
feuern (schießen vs. entlassen)

Die korrekte Bedeutung eines Wortes erschließt sich aus dem Kontext.

Wörterbücher listen die verschiedenen Bedeutungen eines Wortes auf, unterscheiden sich aber oft in der Granularität der Unterscheidungen.

Bank: weitere Unterscheidung zwischen Gebäude, Institut, und den handelnden Personen möglich

Wortbedeutungen

Manchmal sind die verschiedenen Bedeutungen nicht scharf gegeneinander abgegrenzt.

Er hat ein Konto bei der **Bank** ($\rightarrow$ Geldinstitut)

Nach der **Bank** müssen Sie rechts abbiegen ($\rightarrow$ Gebäude)

Die **Bank** hat den Kredit bewilligt ($\rightarrow$ Geldinstitut, Mitarbeiter?)

Wortbedeutungs-Desambiguierung und Übersetzung

Die Bedeutungen eines Wortes werden oft unterschiedlich übersetzt:

Hahn → tap, cock
Karte → ticket, card, map
Bank → bank, bench

Ufer, Bank ← bank

Anmerkung: Die Desambiguierung zwischen Wortarten (Ich sichere das zu. vs. eine sichere Sache) ist eine andere Aufgabe.

Wortbedeutungs-Desambiguierung mit statistischen Modellen

Wir werden nun ein statistisches Modell für die Wortbedeutungs-Desambiguierung betrachten, welches als wesentliche Information die in der Nähe des mehrdeutigen Wortes aufgetretenen Inhaltswörter benutzt.

Der Hahn am Waschbecken im Badezimmer tropft.

Der Hahn auf dem benachbarten Bauernhof kräht jeden Morgen.

Wir werden dieses statistische Modell nun mathematisch herleiten.

Wortbedeutungs-Desambiguierung mit Naïve Bayes

Wir wollen die wahrscheinlichste Bedeutung $\hat{s}$ des Wortes w im Kontext C bestimmen:

$$\begin{aligned}\hat{s} &= \arg\max_s p(s|C) \\ &= \arg\max_s \frac{p(s,C)}{p(C)} \\ &= \arg\max_s p(s,C)\end{aligned}$$

p(C) ist eine Konstante, die keinen Einfluss auf das Ergebnis der argmax-Operation hat.

Wie wird $p(s, C)$ definiert?

Wortbedeutungs-Desambiguierung mit Naïve Bayes

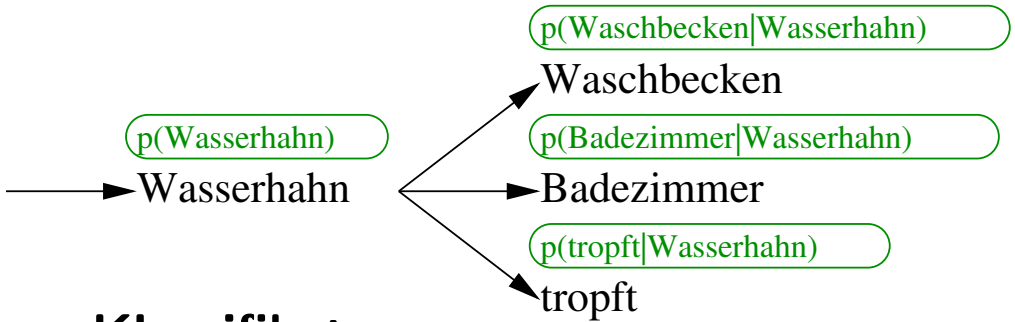
Der Kontext C wird durch eine Menge von Kontextwörtern $= w_1 \ldots w_n$ repräsentiert (z.B. alle Inhaltswörter, die höchstens 50 Positionen entfernt sind).

$$p(s, C) = p(s, w_1 \ldots w_n) = p(s) \prod_{i=1}^{n+1} p(w_i | s, w_1 \ldots w_{i-1})$$

Auch hier fügen wir ein Endesymbol hinzu.

Zur Vereinfachung des Modelles wird angenommen, dass die Kontextwörter statistisch unabhängig sind, wenn die Bedeutung s gegeben ist:

$$p(s, C) = p(s) \prod_{i=1}^{n+1} p(w_i | s)$$



Damit erhalten wir den **Naïve Bayes-Klassifikator**:

$$\hat{s} = \arg\max_s p(s) \prod_{i=1}^{n+1} p(w_i | s)$$

Beispiel

$$p(s, C) = p(s) \prod_{i=1}^{n+1} p(w_i|s)$$

Text: Der Hahn am Waschbecken im Badezimmer tropft.

Klasse: Wasserhahn

Wahrscheinlichkeit:

p(Wasserhahn, Waschbecken, Badezimmer, tropft) = p(Wasserhahn) * p(Waschbecken|Wasserhahn) * p(Badezimmer|Wasserhahn) * p(tropft|Wasserhahn) * p($\langle/s\rangle$|Wasserhahn)

Parameterschätzung

Um einen Wortbedeutungsdesambiguierer für ein bestimmtes Wort zu erstellen, braucht man ein Korpus, in dem jedes Vorkommen dieses Wortes manuell mit seiner korrekten Bedeutung annotiert wurde.

Man zählt die Bedeutungen ($\rightarrow f(s)$) und berechnet ihre Apriori-Wahrscheinlichkeit $p(s)$:

$$p(s) = \frac{f(s)}{\sum_{s'} f(s')}$$

Dann zählt man alle Wörter, die im Kontext der Bedeutung s des ambigen Wortes auftauchen, ($\rightarrow f(w, s)$) und berechnet:

$$p(w|s) = \frac{f(w, s)}{\sum_{w'} f(w', s)}$$

$p(w|s)$ muss noch **geglättet** werden, bspw. durch Backoff-Glättung mit der Backoff-Verteilung $p(w) = f(w)/N$ mit $f(w) = \sum_{s'} f(w, s')$ und $N = \sum_{w'} f(w')$.

Desambiguierung

Die Anwendung des Naïve Bayes-Klassifikators ist einfach.

Für ein gegebenes mehrdeutiges Wort w im Kontext der Wörter $w_1 \ldots w_n$

- berechnet man das Produkt $p(s) \prod_{i=1}^{n} p(w_i|s)$ für jede Bedeutung s
- und wählt die Bedeutung mit dem höchsten Wert.

Kontextwörter, die nirgends in den Trainingsdaten aufgetaucht sind, können bei der Klassifikation weggelassen werden, da sie keine Information über die Lesart liefern.

Alternativ könnten die Backoff-Wahrscheinlichkeiten $p(w)$ der Wörter mit einer uniformen Verteilung 1/B geglättet werden, was aber die Kenntnis der Vokabulargröße B voraussetzt.

Beispiel

Seine einzige Verwandte, die **Schwester**, liegt im Krankenhaus

lemmatisierte Kontextwörter:
einzig, Verwandte, Krankenhaus, liegen

Wahrscheinlichkeiten:

Wort w	p(w|SchwesterV)	p(w|SchwesterK)
einzig	0.001	0.00015
Verwandte	0.02	0.0001
liegen	0.0005	0.002
Krankenhaus	0.0001	0.03

p(SchwesterV) = p(SchwesterK) = 0.5

$p(SchwesterV, einzig, Verwandte, Krankenhaus, liegen) = 5e\text{-}13$
$p(SchwesterK, einzig, Verwandte, Krankenhaus, liegen) = 4.5e\text{-}13$

$\Rightarrow$ Wir wählen hier die Lesart *SchwesterV*.

Pseudowort-Evaluierung

Die manuelle Annotation von Daten ist teuer.

Zur Evaluierung einer Desambiguierungsmethode kann man jedoch ein großes perfekt annotiertes Trainingskorpus erstellen, indem man zwei Wörter zu einem neuen mehrdeutigen "Pseudo"-Wort zusammenfasst:

Pseudowort-Evaluierung:

- Nimm ein großes Textkorpus.
- Wähle zwei Wörter w_1 und w_2 und ersetze alle Vorkommen von w_1 und w_2 durch das Pseudowort w_1-w_2.
- Behalte das ursprüngliche Wort als ,,Bedeutungs"-Annotation.
- Teile das Korpus in zwei Teile.
- Trainiere den Desambiguierer auf dem ersten Teil, zwischen den Bedeutungen w_1 und w_2 des Pseudowortes w_1-w_2 zu unterscheiden.
- Evaluiere den Desambiguierer auf dem zweiten Teil durch Vergleich der Klassifikatorausgabe mit dem Originalwort.

Beispiel zur Pseudowort-Evaluierung

Originalkorpus:

Jedes Kind besucht eine Schule die der Staat finanziert

Die Wörter *Staat* und *Kind* werden zu einem Pseudowort *Staat-Kind* zusammengefasst, und das Originalwort als Annotation gespeichert:

Jedes Staat-Kind besucht eine Schule die der Staat-Kind finanziert
Kind Staat

Darauf wird ein Klassifikator trainiert und dann auf Testdaten evaluiert:
Ein Staat-Kind kommt zu spät zur Schule
Kind

Baseline und Obergrenze

Wenn kein Vergleichssystem zur Verfügung steht, bietet sich die folgende Baseline-Methode für Vergleichszwecke an:

Baseline: Wähle immer die Lesart, die in den Trainingsdaten am häufigsten war.

Wenn diese Baseline nicht übertroffen wird, (was oft gar nicht so einfach ist), ist das Verfahren nutzlos.

Wenn man wissen möchte, wie weit die Klassifizierungsgenauigkeit noch gesteigert werden kann, kann man die Genauigkeit, die ein Mensch bei der Aufgabe erreicht, als **Obergrenze** zum Vergleich heranziehen.

Kurzzusammenfassung: Naïve-Bayes-Modell

gegeben: Menge von Texten mit korrekter Klasse

Bei der Wortbedeutungsdesambiguierung entspricht der Text dem Kontext des ambigen Wortes und die Klasse der Lesart.

Das **Naïve-Bayes-Modell** definiert die gemeinsame Wahrscheinlichkeit eines Klasse c und eines Textes x_1^n wie folgt:

$$p(c, x_1^n) = p(c) \prod_{i=1}^{n+1} p(x_i|c)$$

Wir fügen ein Endetoken x_{n+1} hinzu, damit wir bei variabler Textlänge eine Wahrscheinlichkeitsverteilung erhalten.

Schätzung der Wahrscheinlichkeiten:

$$p(c) = \frac{f(c)}{\sum_{c'} f(c')} \qquad p(x|c) = \frac{f(x,c)}{\sum_{x'} f(x',c)}$$

Beispiel: p(Spam|Spende, von, Bill, Gates) =
p(Spam) p(Spende|Spam) p(von|Spam) p(Bill|Spam) p(Gates|Spam) p($\langle/s\rangle$|Spam)

Hidden-Markov-Modelle: Wortart-Tagging

Wortart-Ambiguitäten

Viele Wörter sind bzgl. der Wortart ambig (mehrdeutig).

shop: Singularnomen, finites oder infinites Verb

shops: Pluralnomen oder finites Verb

saw: finites Vergangenheitsverb, Singularnomen, finites Präsensverb, Infinitiv

Wir leiten nun ein statistisches Modell für Wortart-Desambiguierung her, welches folgende Arten von Information aus den Trainingsdaten nutzt:

- Wie häufig trat das Wort shop mit der Wortart NN auf?
- Wie häufig folgte die Wortart NN auf die Wortart DT?

Fragen

Bei der Entwicklung eines statistischen Modelles sind meistens 4 Fragen zu beantworten:

1. Zwischen welchen Alternativen muss desambiguiert werden?
   hier: Wortartfolgen
2. Mit welchem Modell werden die Wahrscheinlichkeiten der Alternativen definiert?
3. Wie werden die Modell-Parameter geschätzt?
4. Wie wird die beste Analyse effizient berechnet?

Wahrscheinlichkeitsmodell

Gesucht ist die wahrscheinlichste Wortartfolge $\hat{t}_1^n$ für eine gegebene Wortfolge $w_1^n = w_1, ..., w_n$

$$\hat{t}_1^n = \arg\max_{t_1^n} p(t_1^n | w_1^n) = \arg\max_{t_1^n} \frac{p(t_1^n, w_1^n)}{p(w_1^n)} = \arg\max_{t_1^n} p(t_1^n, w_1^n)$$

Die Konstante $p(w_1^n)$ hat keinen Einfluss auf das Maximierungsergebnis.

Zerlegung in ein Produkt bedingter Wahrscheinlichkeiten:

$$p(t_1^n, w_1^n) = \left(\prod_{i=1}^{n+1} p(t_i | t_1^{i-1}) \right) \prod_{i=1}^{n+1} p(w_i | t_1^{n+1}, w_1^{i-1})$$

Wir fügen hier ein Endetag $t_{n+1} = \langle /s \rangle$ und ein Endetoken $w_{n+1} = \epsilon$ hinzu, mit $p(\epsilon | \langle /s \rangle) = 1$.

Hidden-Markow-Modelle

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i|t_1^{i-1})\; p(w_i|w_1^{i-1}, t_1^{n+1})$$

Wir machen nun die folgenden vereinfachenden Annahmen:

- Das Wortart-Tag t_i hängt nur von den k vorherigen Tags ab.
- Das Wort w_i hängt nur von seiner Wortart t_i ab.
- Die Wahrscheinlichkeiten sind unabhängig von der Position.

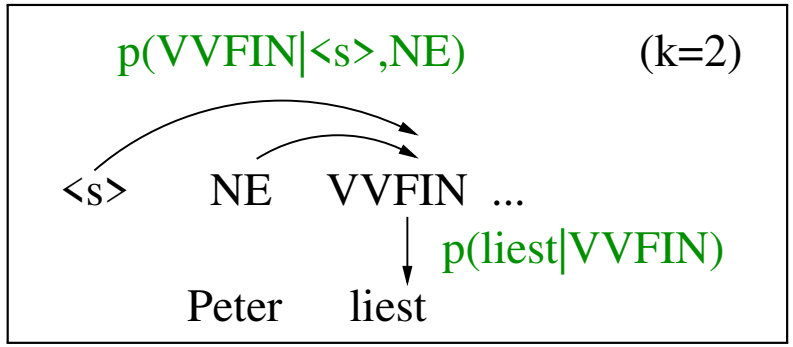
$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i|t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i|t_i)}_{\text{lexikalische Wk.}}$$

Dieses Modell heißt **Hidden-Markow-Modell**, weil die Zustände (Tags bzw. Tagpaare beim Trigramm-Tagger) nicht direkt beobachtbar sind.

Wir fügen am Anfang k Starttags hinzu, damit $p(t_1|t_{1-k}^0)$ definiert ist.

Beispiel

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1}) p(w_i|t_i)$$



Bei einem HMM 1. Ordnung (k=1) gilt:

$$\begin{aligned} p(\text{NE VVFIN, Peter liest}) = {} & p(\text{NE}|\langle s\rangle)\ p(\text{Peter}|\text{NE}) \cdot \\ & p(\text{VVFIN}|\text{NE})\ p(\text{liest}|\text{VVFIN}) \cdot \\ & p(\langle /s\rangle|\text{VVFIN})\ p(\epsilon|\langle /s\rangle) \end{aligned}$$

Beim HMM 2. Ordnung (k=2) gilt:

$$\begin{aligned} p(\text{NE VVFIN, Peter liest}) = {} & p(\text{NE}|\langle s\rangle, \langle s\rangle)\ p(\text{Peter}|\text{NE}) \cdot \\ & p(\text{VVFIN}|\langle s\rangle,\text{NE})\ p(\text{liest}|\text{VVFIN}) \cdot \\ & p(\langle /s\rangle|\text{NE,VVFIN})\ p(\epsilon|\langle /s\rangle) \end{aligned}$$

Zwei Typen von Parametern

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i|t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i|t_i)}_{\text{lexikalische Wk.}}$$

Die Kontextwahrscheinlichkeiten erfassen die syntaktischen Abhängigkeiten der Wortart-Tags vom linken Satzkontext.

Die lexikalischen Wahrscheinlichkeiten erfassen die syntaktischen Eigenschaften des aktuellen Wortes.

Spielt der rechte Kontext also keine Rolle?

Rechter Satzkontext

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i | t_{i-k}^{i-1}) p(w_i | t_i)$$

Auch der rechte Satzkontext ist für die Wortart-Annotation wichtig!

Beispiel: Mario, der Klempner, siegte. vs. Mario, der siegte, ist Klempner.

Im 1. Satz ist "der" ein Artikel (ART), im 2. Satz dagegen ein Relativpronomen (PRELS). Die linken Satzkontexte sind identisch.

Der HMM-Tagger berechnet die Tagfolge mit der höchsten Wahrscheinlichkeit. Wenn dem Wort "der" das falsche Tag zugewiesen wird, sind die Wahrscheinlichkeiten der nachfolgenden Tags – p(NN | PRELS) bzw. p(VVFIN | ART) – klein, und die gesamte Tagfolge wird unwahrscheinlich.

Weil alle Tags zusammen desambiguiert werden, beeinflusst also auch der rechte Kontext die Annotation eines Wortes.

Rechter Satzkontext: Beispiel 2

Ein HMM ist gegeben durch die Tabelle:

	A	B	$\langle/s\rangle$	a	b	x	ϵ
A	0.5	0	0.5	0.5	0	0.5	0
B	0	0.5	0.5	0	0.5	0.5	0
$\langle s\rangle$	0.5	0.5	0	0	0	0	1

mit $p(\langle/s\rangle|A) = 0.5$ und $p(x|A) = 0.5$).

- Die einzig mögliche Tagfolge der Tokenfolge xxxxxxa lautet AAAAAAA.
- Die einzig mögliche Tagfolge der Tokenfolge xxxxxxb lautet BBBBBBB.
- Bei allen anderen Tagfolgen ist die gemeinsame Wahrscheinlichkeit mit der Tokenfolge gleich 0.
- Dasselbe gilt für alle anderen Tokenfolgen der Form x^*a und x^*b.
- Hier besteht also eine (indirekte) Abhängigkeit von einem beliebig weit entfernten Token!

Parameterschätzung

Zur Schätzung der Parameter braucht man ein **Trainingskorpus**, in dem jedes Wort mit seiner Wortart annotiert ist.

Man zählt die k+1-Gramm-Häufigkeiten $f(t_1, ..., t_{k+1})$ und die Tag-Wort-Häufigkeiten $f(t, w)$ und schätzt dann die Wahrscheinlichkeiten wie folgt:

$$p(w|t) = \frac{f(t,w)}{\sum_{w'} f(t,w')} \qquad p(t|t_1, ..., t_k) = \frac{f(t_1, ..., t_k, t)}{\sum_{t'} f(t_1, ..., t_k, t')}$$

Beispiel:

$$p(\text{Haus}|\text{NN}) = \frac{f(\text{NN}, \text{Haus})}{\sum_{w'} f(\text{NN}, w')} \qquad p(\text{NN}|\text{ART}) = \frac{f(\text{ART}, \text{NN})}{\sum_{t'} f(\text{ART}, t')}$$

Die Kontextwahrscheinlichkeiten $p(t|t_1, ..., t_k)$ müssen aber noch geglättet werden.

Behandlung unbekannter Wörter

Kein manuell annotiertes Korpus ist so groß, dass es alle möglichen Wörter enthält. Ein Wortart-Tagger muss daher mit **unbekannten Wörtern** umgehen können.

Großschreibung (Teil vs. teil) und **Wortendungen** (-keit, -lich) liefern wertvolle Information über die möglichen Wortarten eines Wortes.

Um diese Informationen nutzen zu können, müssen wir das Modell modifizieren. Wir wenden dazu das Bayes'sche Theorem auf die lexikalischen Wahrscheinlichkeiten an und erhalten:

$$p(w|t) = \frac{p(t|w)p(w)}{p(t)}$$

Die Konstante $p(w)$ können wir bei der Suche nach der besten Tagfolge ignorieren (sofern die Tokenisierung eindeutig ist).

Behandlung unbekannter Wörter

Die Apriori-Wahrscheinlichkeit $p(t) = f(t)/N$ (N = Korpusgröße) der Tags kann leicht aus dem Trainingskorpus geschätzt werden.

Die bedingte Wahrscheinlichkeit $p(t|w)$ kann anhand des Wortsuffixes (z.B. der Länge L) geschätzt werden:

$$p(t|w) = p(t|a_1 a_2 ... a_n) \approx p_{suff}(t|a_{n-L+1} ... a_n)$$

$$p(\text{NN}|\text{schwärzlich}) \approx p_{suff}(\text{NN}|\text{zlich}) \quad \text{falls } L = 5$$

Die Wahrscheinlichkeit eines Tags t gegeben ein Suffix $s_1, ..., s_L$

$$p_{suff}(t|s_1 ... s_L) = \frac{f_{suff}(s_1 ... s_L, t)}{\sum_{t'} f_{suff}(s_1 ... s_L, t')} \quad \text{mit } f_{suff}(s_1 ... s_L, t) = \sum_{w=\alpha s_1 ... s_L} f(w, t)$$

(α ist hier eine beliebige Buchstabenfolge.)
wird aus Trainingsdaten geschätzt und mit einem Backoff-Verfahren geglättet.

Für groß- und kleingeschriebene Wörter schätzt man am besten separate Parameter. Dafür kann man einfach einen Buchstaben (z.B. "G" für Groß- und "K" für Kleinschreibung) an das Wortsuffix anhängen: $p_{suff}(\text{NN}|\text{zlichK})$

Behandlung unbekannter Tags

neue Formel zum Taggen mit unbekannten Wörtern:

$$\hat{t}_1^n = \arg\max_{t_1^n} \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1}) \frac{p(t_i|w_i)}{p(t_i)}$$

Diese Formel ist keine Wahrscheinlichkeitsverteilung $p(t_1^n, w_1^n)$, weil $p(w)$ fehlt.

Neben unbekannten Wörtern stellen auch ungesehene Tags bekannter Wörter ein Problem dar (bspw. wenn das englische Wort "indicate" nur als Infinitiv aber nicht als finites Verb aufgetaucht ist.)

Die suffixbasierten Tag-Wahrscheinlichkeiten geben meist auch den ungesehenen Tags eine kleine Wahrscheinlichkeit.

Daher ist es sinnvoll, wortbasierte und suffixbasierte Wahrscheinlichkeiten in einem Backoff-Modell zu kombinieren:

$$p(t|w) = r(t|w) + \alpha(w)\ p_{suff}(t|suffix(w))$$

mit $r(t|w) = max(0, \frac{f(w,t)-\delta}{\sum_{t'} f(w,t')})$

Beispiel zur Berechnung der lexikalischen Wahrscheinlichkeiten

für Suffixlänge L=5:

$$
\begin{aligned}
p(NN|schwärzlich) = {} & r(NN|schwärzlich) + \alpha(schwärzlich)\ (\\
& r_{suff}(NN|zlichK) + \alpha_{suff}(zlichK)\ (\\
& r_{suff}(NN|lichK) + \alpha_{suff}(lichK)\ (\\
& r_{suff}(NN|ichK) + \alpha_{suff}(ichK)\ (\\
& r_{suff}(NN|chK) + \alpha_{suff}(chK)\ (\\
& r_{suff}(NN|hK) + \alpha_{suff}(hK)\ p(NN|K)))))
\end{aligned}
$$

Wörter mit weniger als L Buchstaben werden am Anfang mit Leerzeichen aufgefüllt, bevor das Suffix extrahiert wird.

Suche

Problem: Viele Sätze haben zuviele mögliche Tagfolgen, um alle aufzählen und ihre Wahrscheinlichkeit berechnen zu können.

Wie könnte eine effizientere Methode für die Berechnung der besten Tagfolge aussehen?

Beobachtung: Wenn zwei mögliche Wortartfolgen $T = t_1, ..., t_m$ und $S = s_1, ..., s_m$ (für die ersten m Wörter eines Satzes) in den letzten k Tags übereinstimmen, dann kann bei einem HMM k-ter Ordnung die weniger wahrscheinliche der beiden Tagfolgen nicht ein Präfix der besten Gesamttagfolge sein, und wir können sie ignorieren.

Widerspruchsbeweis

Angenommen S ist weniger wahrscheinlich als T, aber ein Präfix der besten Tagfolge. Dann hat die beste Tagfolge die Form SV, wobei V eine weitere Tagfolge ist. In diesem Fall hätte aber die Tagfolge TV eine höhere Wahrscheinlichkeit als SV, weil T wahrscheinlicher als S ist und die Wahrscheinlichkeiten, die jeweils für V hinzumultipliziert werden, bei SV und TV identisch sind. Also kann S kein Präfix der besten Tagfolge sein. □

Beispiel: k=2 T=X,A,B S=Y,A,B V=C,D
TV=X,A,B,C,D SV=Y,A,B,C,D

C hängt hier nur von A und B ab. D hängt nur von B und C ab.
Also sind die Tags C,D nach X,A,B genauso wahrscheinlich wie nach Y,A,B.
Also muss die Tagfolge X,A,B,C,D wahrscheinlicher als Y,A,B,C,D sein.

Suche

Diese Eigenschaft erlaubt eine effiziente Verarbeitung, hat aber zur Folge, dass manche Ambiguitäten nicht korrekt aufgelöst werden können:

The	horse	raced	past	the	barn	(fell)	.
DT	NN	VBD	IN	DT	NN		.
DT	NN	VBN	IN	DT	NN	VBD	.

Ein HMM 2. Ordnung würde das Wort "raced" gleich taggen, egal ob "fell" nachfolgt oder nicht. Erst ein HMM 4. Ordnung kann weit genug zurückschauen, um korrekt zu desambiguieren.

Anmerkung: Man könnte das Problem lösen, indem man das Tagset verfeinert und bspw. alle Tags, die nach einem finiten Verb folgen, mit einem Apostroph markiert.

The	horse	raced	past	the	barn	(fell)	.
DT	NN	VBD	IN'	DT'	NN'		.
DT	NN	VBN	IN	DT	NN	VBD	.

Viterbi-Algorithmus (Bigramm-Tagger)

1. Initialisierung: $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für $1 \leq k \leq n+1$)

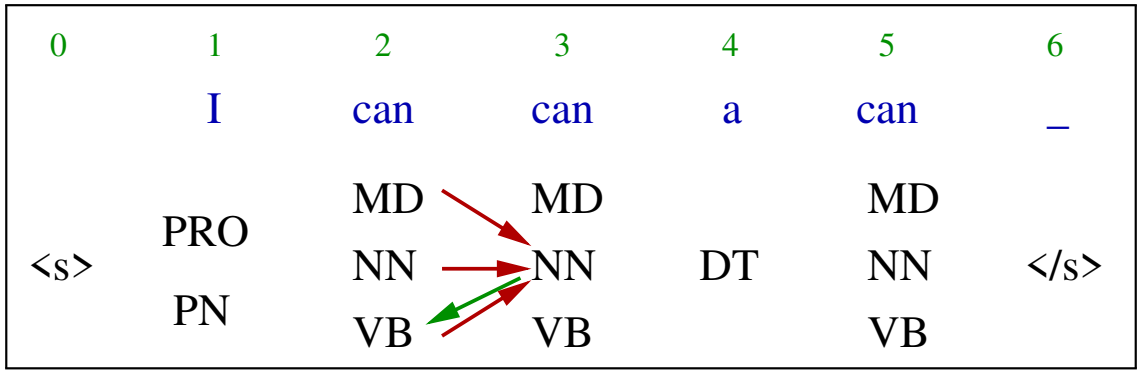
$$\begin{aligned} \delta_t(k) &= \max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \\ \psi_t(k) &= \arg\max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \end{aligned}$$

3. Ausgabe: (für $0 < k \leq n$)

$$\begin{aligned} t_{n+1} &= \langle /s \rangle \\ t_k &= \psi_{t_{k+1}}(k+1) \quad \text{für } 1 \leq k \leq n \end{aligned}$$

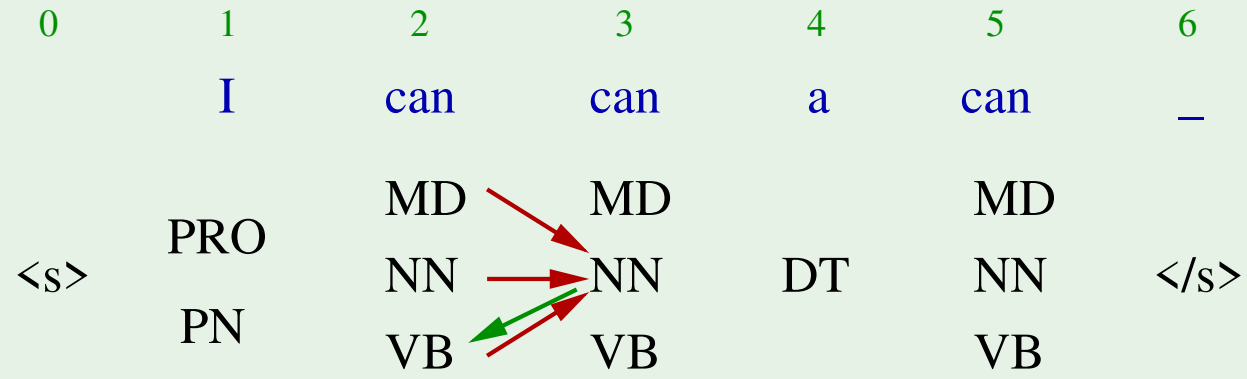
t, t' sind Tags, k sind Wortpositionen, $\delta_t(k)$ sind die Viterbiwahrscheinlichkeiten und $\psi_t(k)$ sind die besten Vorgängertags

Viterbi-Algorithmus



- Wir berechnen für jedes mögliche Tag t an jeder Position i von links nach rechts eine Viterbi-Wahrscheinlichkeit $\delta_t(i)$.
- Die Viterbi-Wk. $\delta_t(i)$ ist die Wahrscheinlichkeit der besten Tagfolge, die an Position 0 mit $\langle s \rangle$ beginnt und an Position i mit Tag t endet.
- Wir berechnen die Viterbi-Wk. auf Basis der Viterbi-Wken der Tags an der vorherigen Position $i-1$.
- Dazu multiplizieren wir die Viterbi-Wk. $\delta_{t'}(i-1)$ jedes Vorgängertags t' mit der Kontext-Wk. $p(t|t')$ und der lexikalischen Wk. $p(w_i|t)$.
- Wir merken uns dasjenige Vorgänger-Tag, welches den besten Wert geliefert hat, in $\psi_t(i)$ und den besten Wert selbst in $\delta_t(i)$.
- Zum Schluss extrahieren wir von rechts nach links die beste Tagfolge:
  $t_n = \psi_{\langle /s \rangle}(n+1), \quad t_{n-1} = \psi_{t_n}(n), \quad t_{n-2} = \psi_{t_{n-1}}(n-1), \quad \ldots$

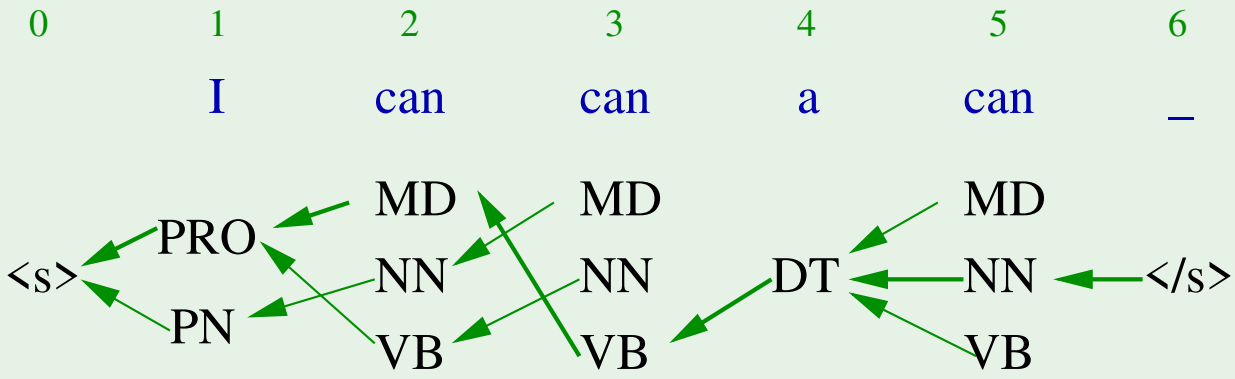
Beispiel: Viterbi-Algorithmus



$$\delta_{NN}(3) = \max \begin{pmatrix} \delta_{MD}(2)\ p(NN|MD)\ p(can|NN), \\ \delta_{NN}(2)\ p(NN|NN)\ p(can|NN), \\ \delta_{VB}(2)\ p(NN|VB)\ p(can|NN) \end{pmatrix}$$

Der grüne Pfeil repräsentiert $\psi_{NN}(3)$, falls VB das beste Vorgängertag war.

Beispiel: Viterbi-Algorithmus



Die Pfeile repräsentieren die Werte der ψ-Variablen.

Extraktion der besten Tagfolge $t_1,\ldots,t_5$ in umgekehrter Reihenfolge:

$$
\begin{aligned}
t_5 &= \psi_{\langle /s \rangle}(6) = \text{NN} \\
t_4 &= \psi_{NN}(5) = \text{DT} \\
t_3 &= \psi_{DT}(4) = \text{VB} \\
t_2 &= \psi_{VB}(3) = \text{MD} \\
t_1 &= \psi_{MD}(2) = \text{PRO}
\end{aligned}
$$

Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modelles nicht einem einzelnen Tag, sondern einem Tagpaar.
- Übergänge gibt es nur zwischen Zuständen (t, t') und (t'', t''') mit $t' = t''$

1. Initialisierung: $\delta_{t',t}(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \text{ und } t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für $0 < k \leq n+2$)

$$\begin{aligned} \delta_{t',t}(k) &= \max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t) \\ \psi_{t',t}(k) &= \arg\max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t) \end{aligned}$$

Wir fügen hier 2 Endetags hinzu und iterieren bis n+2. Das hat den Vorteil, dass wir das letzte Tag direkt in $\psi_{\langle/s\rangle,\langle/s\rangle}(n+2)$ finden. Andernfalls müssten wir erst in der Spalte n+1 durch Maximierung über t nach dem Eintrag $\delta_{t,\langle/s\rangle}(n+1)$ mit der größten Viterbi-Wahrscheinlichkeit suchen, um von dort ausgehend die beste Tagfolge zu extrahieren.

Trigramm-Tagger-Beispiel (für den Satz: *I can can a can*)

$$\delta_{\langle s\rangle,\langle s\rangle}(0) = 1$$

$$\delta_{\langle s\rangle,PRO}(1) = \delta_{\langle s\rangle,\langle s\rangle}(0)\ \mathrm{p}(\mathrm{PRO}|\langle s\rangle,\langle s\rangle)\ \mathrm{p}(\mathrm{I}|\mathrm{PRO})$$
$$\delta_{\langle s\rangle,PN}(1) = \delta_{\langle s\rangle,\langle s\rangle}(0)\ \mathrm{p}(\mathrm{PN}|\langle s\rangle,\langle s\rangle)\ \mathrm{p}(\mathrm{I}|\mathrm{PN})$$

$$\delta_{PRO,MD}(2) = \delta_{\langle s\rangle,PRO}(1)\ \mathrm{p}(\mathrm{MD}|\langle s\rangle,\mathrm{PRO})\ \mathrm{p}(\mathrm{can}|\mathrm{MD})$$
$$\delta_{PRO,NN}(2) = \delta_{\langle s\rangle,PRO}(1)\ \mathrm{p}(\mathrm{NN}|\langle s\rangle,\mathrm{PRO})\ \mathrm{p}(\mathrm{can}|\mathrm{NN})$$
$$\delta_{PRO,VB}(2) = \delta_{\langle s\rangle,PRO}(1)\ \mathrm{p}(\mathrm{VB}|\langle s\rangle,\mathrm{PRO})\ \mathrm{p}(\mathrm{can}|\mathrm{VB})$$

$$\delta_{PN,MD}(2) = \delta_{\langle s\rangle,PN}(1)\ \mathrm{p}(\mathrm{MD}|\langle s\rangle,\mathrm{PN})\ \mathrm{p}(\mathrm{can}|\mathrm{MD})$$
$$\delta_{PN,NN}(2) = \delta_{\langle s\rangle,PN}(1)\ \mathrm{p}(\mathrm{NN}|\langle s\rangle,\mathrm{PN})\ \mathrm{p}(\mathrm{can}|\mathrm{NN})$$
$$\delta_{PN,VB}(2) = \delta_{\langle s\rangle,PN}(1)\ \mathrm{p}(\mathrm{VB}|\langle s\rangle,\mathrm{PN})\ \mathrm{p}(\mathrm{can}|\mathrm{VB})$$

$$\begin{aligned}\delta_{MD,MD}(3) = \max(&\delta_{PRO,MD}(2)\ \mathrm{p}(\mathrm{MD}|\mathrm{PRO},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{MD}),\\ &\delta_{PN,MD}(2)\ \mathrm{p}(\mathrm{MD}|\mathrm{PN},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{MD}))\end{aligned}$$
$$\begin{aligned}\delta_{MD,NN}(3) = \max(&\delta_{PRO,MD}(2)\ \mathrm{p}(\mathrm{NN}|\mathrm{PRO},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{NN}),\\ &\delta_{PN,MD}(2)\ \mathrm{p}(\mathrm{NN}|\mathrm{PN},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{NN}))\end{aligned}$$
$$\begin{aligned}\delta_{MD,VB}(3) = \max(&\delta_{PRO,MD}(2)\ \mathrm{p}(\mathrm{VB}|\mathrm{PRO},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{VB}),\\ &\delta_{PN,MD}(2)\ \mathrm{p}(\mathrm{VB}|\mathrm{PN},\mathrm{MD})\ \mathrm{p}(\mathrm{can}|\mathrm{VB}))\end{aligned}$$

$$\begin{aligned}\delta_{NN,MD}(3) = \max(&\delta_{PRO,NN}(2)\ \mathrm{p}(\mathrm{MD}|\mathrm{PRO},\mathrm{NN})\ \mathrm{p}(\mathrm{can}|\mathrm{MD}),\\ &\delta_{PN,NN}(2)\ \mathrm{p}(\mathrm{MD}|\mathrm{PN},\mathrm{NN})\ \mathrm{p}(\mathrm{can}|\mathrm{MD}))\end{aligned}$$

...

Rechenzeit-Komplexität

Der Viterbi-Algorithmus verwendet 3 Schleifen.

- Die äußerste Schleife iteriert über n+1 Wortpositionen.
- Die mittlere Schleife iteriert über alle m Tags.
- Die innerste Schleife iteriert über alle Kontexte mit k Tags
  (1 Tag beim Bigramm-Tagger und 2 Tags beim Trigramm-Tagger).

Damit ist die Rechenzeit-Komplexität $O((n+1)mm^k) = O(nm^{k+1})$.

Evaluierung

Für die Evaluierung eines Wortart-Taggers brauchen wir ein manuell annotiertes **Testkorpus**, das nicht Teil der Trainingsdaten war.

Schritte:

1. Training des Taggers auf den Trainingsdaten
2. Taggen der Wortfolge des Testkorpus mit dem Tagger
3. Vergleich der Taggerausgabe mit den **Goldstandard**-Tags
4. Genauigkeit = Anzahl korrekte Tags / Anzahl Wörter

Signifikanztest

Beim Vergleich zweier Systeme ist es wichtig zu wissen, ob der Unterschied zwischen ihren Evaluierungsergebnissen **signifikant** (also bedeutsam) ist, oder ob er Zufall sein könnte.

Dazu wenden wir einen **Signifikanztest** an. Wir werden den Vorzeichentest nehmen.

Vorzeichentest zur Taggerevaluierung

Beim Vorzeichentest interessieren nur die Wörter, die **genau einer** der Tagger falsch annotiert hat. Alle anderen werden ignoriert. Angenommen es gibt **60** Wörter in den Testdaten, die NewTagger richtig taggt und OldTagger falsch, und **40** Wörter, die OldTagger richtig taggt und NewTagger falsch.

Nullhypothese: NewTagger ist nicht besser als OldTagger.

$\Rightarrow$ Bei jedem der 100 Wörter ist die Wahrscheinlichkeit, dass NewTagger richtig taggte, maximal 0.5.

Dies ist ein einseitiger Binomialtest, weil uns nur interessiert, ob NewTagger signifikant besser ist, nicht aber, ob er signifikant schlechter ist.

Wir summieren die Werte der Binomialfunktion:

$$b(\geq 60, 0.5, 100) \approx 0.03$$

$\Rightarrow$ Der Unterschied ist signifikant mit einer Fehlerwk. von etwa 3 %.

Optimierung von Metaparametern

Der beschriebene HMM-Tagger hat eine Reihe von **Metaparametern** (Ordnung des Modelles n, maximale Suffixlänge l), die beliebig gewählt werden können.

Um gute Werte dafür auszuwählen, kann der Tagger für verschiedene mögliche Werte der Metaparameter trainiert und dann jeweils auf Testdaten evaluiert werden. Diese Testdaten dürfen aber nicht mit den Testdaten für die eigentliche Evaluierung identisch sein.

Solche Daten für die Parameter-Optimierung nennt man **Entwicklungsdaten** (Developmentdaten).

In der Regel braucht man für Experimente in der Computerlinguistik **Trainingsdaten**, **Developmentdaten** und **Testdaten**.

Unüberwachtes Training

Können wir ein HMM auch ohne manuell annotierte Trainingsdaten trainieren?

- Wenn wir ein annotiertes Korpus haben, können wir ein HMM trainieren.
- Wenn wir ein trainiertes HMM haben, können wir ein Korpus annotieren.

$\Rightarrow$ Henne-Ei-Problem

EM-Training

Lösung: Expectation-Maximization-Training

(maximiert iterativ die Wahrscheinlichkeit der Trainingsdaten)

1. gegeben
   - ein nicht annotiertes Trainingskorpus
   - ein Lexikon mit möglichen Wortarten von Wörtern
2. Initialisiere das HMM uniform (abgesehen davon, dass $p(w|t) = 0$ falls w im Lexikon enthalten und t keines seiner möglichen Tags ist)
3. Annotiere[2] das Trainingskorpus mit dem HMM und extrahiere die Taghäufigkeiten (E-Schritt)
4. Schätze die HMM-Parameter aus den Taghäufigkeiten neu (M-Schritt)
5. weiter mit E-Schritt (bis irgendein Abbruchkriterium erfüllt ist)

[2] Wir werden später sehen, wie das genau gemacht wird.

EM-Training: Beispiel

Sätze: *eine Katze jagt eine Maus* | *die entkommt der Katze* | *der Hund bellt*

context	prediction	f	p	f	p	f	p	f	p	f	p	f	p
ART	eine	1.14	0.49	1.91	0.62	2.0	0.62	2.0	0.56	2.0	0.52	2.0	0.5
VVFIN	eine	0.86	0.22	0.09	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PDS	die	0.6	0.33	0.77	0.42	0.96	0.55	1.0	0.7	1.0	0.87	1.0	0.98
ART	die	0.4	0.17	0.23	0.08	0.04	0.01	0.0	0.0	0.0	0.0	0.0	0.0
PDS	der	1.2	0.67	1.07	0.58	0.79	0.45	0.43	0.3	0.15	0.13	0.02	0.02
ART	der	0.8	0.34	0.93	0.3	1.21	0.37	1.57	0.44	1.85	0.48	1.98	0.5
VVFIN	jagt	1.0	0.26	1.0	0.32	1.0	0.33	1.0	0.33	1.0	0.33	1.0	0.33
VVFIN	entkommt	1.0	0.26	1.0	0.32	1.0	0.33	1.0	0.33	1.0	0.33	1.0	0.33
VVFIN	bellt	1.0	0.26	1.0	0.32	1.0	0.33	1.0	0.33	1.0	0.33	1.0	0.33
NN	Katze	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5
NN	Maus	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25
NN	Hund	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25	1.0	0.25
$\langle s \rangle$	ART	1.37	0.46	1.61	0.54	1.53	0.51	1.65	0.55	1.86	0.62	1.98	0.66
$\langle s \rangle$	PDS	1.2	0.4	1.35	0.45	1.47	0.49	1.35	0.45	1.14	0.38	1.02	0.34
$\langle s \rangle$	VVFIN	0.43	0.14	0.04	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ART	NN	1.94	0.83	2.84	0.92	3.21	0.99	3.57	1.0	3.85	1.0	3.98	1.0
ART	VVFIN	0.4	0.17	0.23	0.08	0.04	0.01	0.0	0.0	0.0	0.0	0.0	0.0
VVFIN	$\langle /s \rangle$	1.0	0.26	1.0	0.32	1.0	0.33	1.0	0.33	1.0	0.33	1.0	0.33
VVFIN	ART	0.97	0.25	1.46	0.47	1.71	0.57	1.91	0.64	1.99	0.66	2.0	0.67
VVFIN	NN	0.86	0.22	0.09	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VVFIN	PDS	0.6	0.16	0.49	0.16	0.29	0.1	0.09	0.03	0.01	0.0	0.0	0.0
VVFIN	VVFIN	0.43	0.11	0.05	0.02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NN	VVFIN	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5
NN	$\langle /s \rangle$	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5	2.0	0.5
PDS	NN	1.2	0.67	1.07	0.58	0.79	0.45	0.43	0.3	0.15	0.13	0.02	0.02
PDS	VVFIN	0.6	0.33	0.77	0.42	0.96	0.55	1.0	0.7	1.0	0.87	1.0	0.98

Das Modell wurde wie früher beschrieben uniform initialisiert. Dann wurden die Häufigkeiten f mit dem Forward-Backward-Algorithmus geschätzt. Aus den Häufigkeiten wurden die neuen Wahrscheinlichkeiten p des HMM geschätzt. Dann wurde iteriert. context-prediction-Paare mit Häufigkeit 0 werden in der Tabelle nicht gezeigt.

EM-Training

Variante 1: Wir benutzen den Viterbi-Algorithmus zum Taggen.

⇒ Nur die wahrscheinlichste Tagfolge wird berücksichtigt.
Alle anderen Tagfolgen werden ignoriert.

⇒ Am Anfang des Trainings gibt es aber noch keine eindeutige beste Tagfolge.

⇒ Das Training funktioniert deshalb so nicht.

Lösung

- Alle Tagfolgen bei der Extraktion der Taghäufigkeiten berücksichtigen.
- Jede Tagfolge wird dabei mit ihrer Wahrscheinlichkeit gewichtet, so dass doppelt so wahrscheinliche Tagfolgen doppelt so viel zu den extrahierten Häufigkeiten beitragen.

Gewichtung der Tagfolgen

Die Tagfolgen sollen so **gewichtet** werden, dass die Summe der Gewichte 1 ergibt ($\Rightarrow$ Wahrscheinlichkeitsverteilung)

Wir berechnen daher die Aposteriori-Wahrscheinlichkeit jeder Tagfolge T ($=$ bedingte Wahrscheinlichkeit von T gegeben die Wortfolge W)

$$p(T|W) = \frac{p(T,W)}{p(W)} = \frac{p(T,W)}{\sum_{T'} p(T',W)}$$

Aus jeder Tagfolge werden die Tag-Tag-Häufigkeiten und die Tag-Wort-Häufigkeiten extrahiert, mit dem Gewicht der Tagfolge (Aposteriori-Wahrscheinlichkeit) multipliziert und aufsummiert.

Mit den so erhaltenen **erwarteten Häufigkeiten** werden die HMM-Parameter neu geschätzt.

Forward-Backward-Algorithmus

Da die Aufzählung aller möglichen Tagfolgen ineffizient ist, verwenden wir wie beim Viterbi-Algorithmus *dynamische Programmierung*

$\Rightarrow$ Forward-Backward-Algorithmus

Dynamische Programmierung

Ein komplexes Problem wird gelöst, indem zunächst Lösungen von Teilproblemen gesucht und daraus die Lösung des komplexen Problems errechnet wird. Die Lösungen der Teilprobleme werden rekursiv nach derselben Methode berechnet.

Die Effizienz der dynamischen Programmierung ergibt sich dadurch, dass die Lösung identischer Teilprobleme nur einmal berechnet und dann mehrfach verwendet wird.

Forward-Backward-Algorithmus

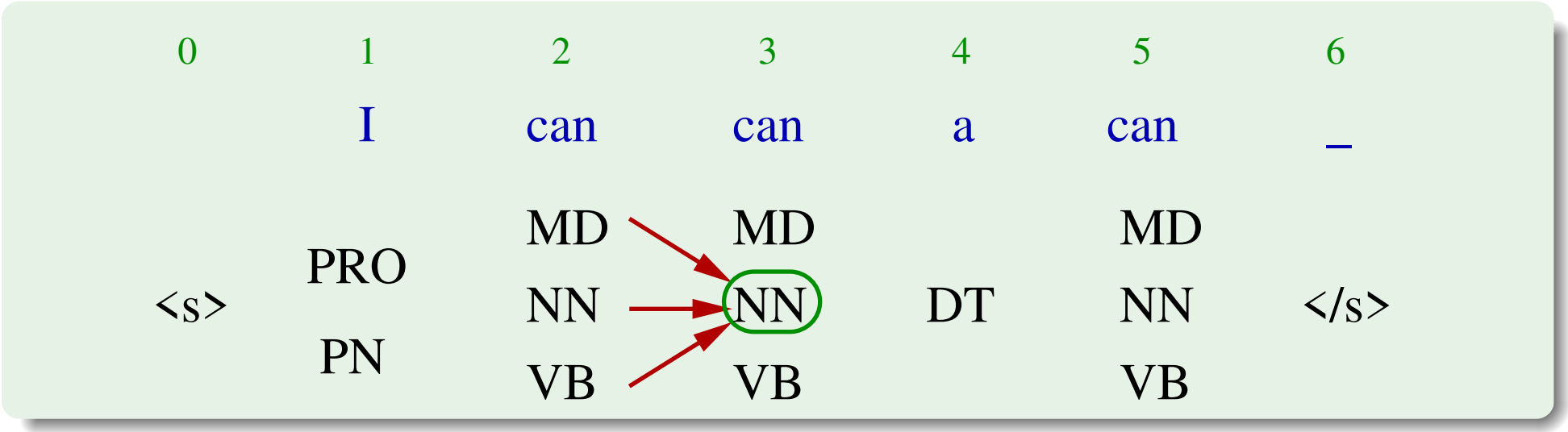
Die Forward-Wahrscheinlichkeit $\alpha_t(k)$ des Tags t an Position k ist die Summe der Wahrscheinlichkeiten aller Tagfolgen t_0^k für die (Teil-)Wortfolge w_1^k, die mit dem Tag $\langle s \rangle$ beginnen und dem Tag t enden.

Formeln für die rekursive Berechnung im Fall des Bigramm-Taggers:

$$\begin{aligned} \alpha_t(0) &= \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \alpha_t(k) &= \sum_{t' \in T} \alpha_{t'}(k-1)\, p(t|t')\, p(w_k|t) \quad \text{für } 0 < k \leq n+1 \end{aligned}$$

Der Forward-Algorithmus unterscheidet sich vom Viterbi-Algorithmus durch die Summe statt der Max-Operation.

Beispiel: Forward-Algorithmus



Berechnung der Forward-Wahrscheinlichkeit des Tags NN an Position 3:

$$\begin{aligned}\alpha_{NN}(3) = & \alpha_{MD}(2)\ p(NN|MD)\ p(can|NN) + \\ & \alpha_{NN}(2)\ p(NN|NN)\ p(can|NN) + \\ & \alpha_{VB}(2)\ p(NN|VB)\ p(can|NN)\end{aligned}$$

Forward-Backward-Algorithmus

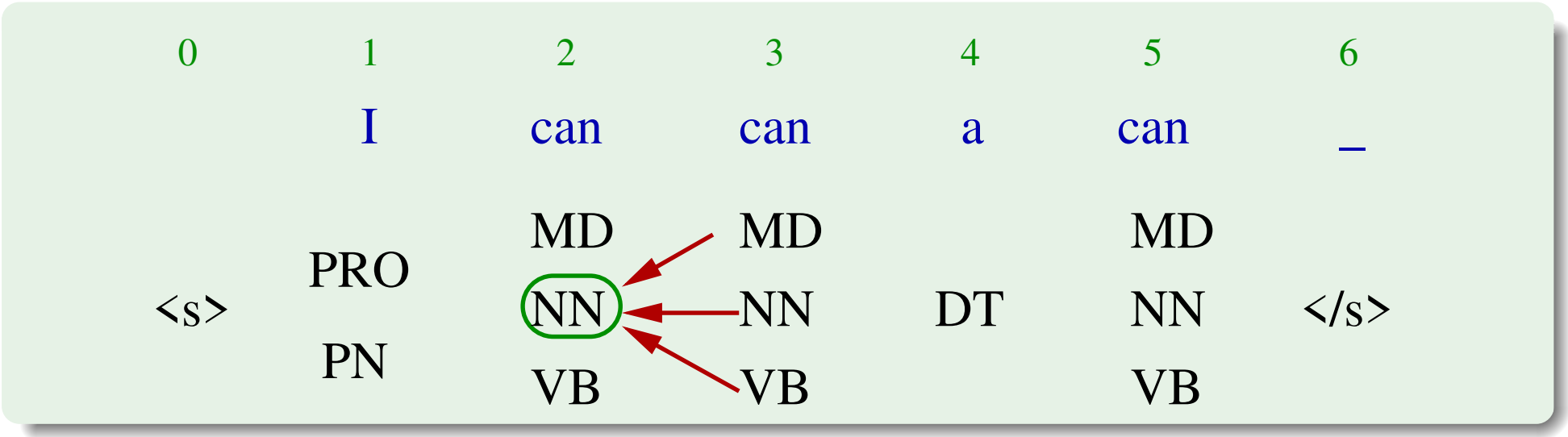
Die Backward-Wahrscheinlichkeit $\beta_t(k)$ des Tags t an Position k ist die Summe der Wahrscheinlichkeiten aller Tagfolgen t_k^{n+1} für die Teil-Wortfolge w_{k+1}^n, die mit dem Tag t beginnen und dem Tag $\langle/s\rangle$ enden.

Die lexikalische Wk. $p(w_k|t_k)$ ist in $\beta_t(k)$ nicht enthalten!

Formeln für die rekursive Berechnung im Fall des Bigramm-Taggers:

$$\begin{aligned}
\beta_t(n+1) &= \begin{cases} 1 & \text{falls } t = \langle/s\rangle \\ 0 & \text{sonst} \end{cases} \\
\beta_t(k-1) &= \sum_{t' \in T} p(t'|t)\, p(w_k|t')\, \beta_{t'}(k) \qquad \text{für } 0 < k \leq n+1
\end{aligned}$$

Beispiel: Backward-Algorithmus



Berechnung der Backward-Wahrscheinlichkeit des Tags NN an Position 2:

$$\begin{aligned}\beta_{NN}(2) = & \beta_{MD}(3)\ p(MD|NN)\ p(can|MD) + \\ & \beta_{NN}(3)\ p(NN|NN)\ p(can|NN) + \\ & \beta_{VB}(3)\ p(VB|NN)\ p(can|VB)\end{aligned}$$

Forward-Backward-Algorithmus

Die (Aposteriori-)Wahrscheinlichkeit $p(t_k = t|w_1^n) = \gamma_t(k)$ der Wortart t an Position k ist die Summe der Wahrscheinlichkeiten aller möglichen Tagfolgen t_1^n mit Tag t an Position k geteilt durch die Gesamtwk. aller Tagfolgen.

$$\gamma_t(k) = \frac{\sum_{t_1^n : t_k = t} p(t_1^n, w_1^n)}{\sum_{t_1^n} p(t_1^n, w_1^n)}$$

Sie kann aus den Forward- und Backward-Wahrscheinlichkeiten berechnet werden:

$$\gamma_t(k) = \frac{\alpha_t(k)\ \beta_t(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

$\alpha_t(k)$ ist die Wahrscheinlichkeit aller partiellen Tagfolgen, die an Position 0 mit $\langle s \rangle$ starten und an Position k mit dem Tag t enden.

$\beta_t(k)$ ist die Wahrscheinlichkeit aller partiellen Tagfolgen, die an Position k mit dem Tag t starten und an Position n+1 mit $\langle /s \rangle$ enden.

$\alpha_t(k) \cdot \beta_t(k)$ ist die Wahrscheinlichkeit aller Tagfolgen des Satzes mit dem Tag t an Position k.

$\alpha_{\langle /s \rangle}(n+1)$ ist die Wahrscheinlichkeit aller Tagfolgen insgesamt, da jede Tagfolge an Position n+1 mit $\langle /s \rangle$ endet.

Forward-Backward-Algorithmus

0	1	2	3	4	5	6
	I	can	can	a	can	_
<s>	PRO PN	MD NN VB	MD NN VB	DT	MD NN VB	</s>

$$\gamma_{NN}(3) = \frac{\alpha_{NN}(3)\,\beta_{NN}(3)}{\alpha_{\langle/s\rangle}(6)}$$

$\alpha_{NN}(3)$: Wahrscheinlichkeit, dass die Wörter "I can can" mit beliebigen Tags generiert werden, wobei das 3. Tag jedoch "NN" ist.

$\beta_{NN}(3)$: Wahrscheinlichkeit, dass die Wörter "a can" mit beliebigen Tags generiert werden, falls das 3. Tag "NN" ist. (Das Endetag wird ebenfalls generiert.)

$\alpha_{\langle/s\rangle}(6)$: Forward-Wahrscheinlichkeit des Ende-Tags und damit die Gesamtwk. des Satzes summiert über alle möglichen Tagfolgen.

Forward-Backward-Algorithmus

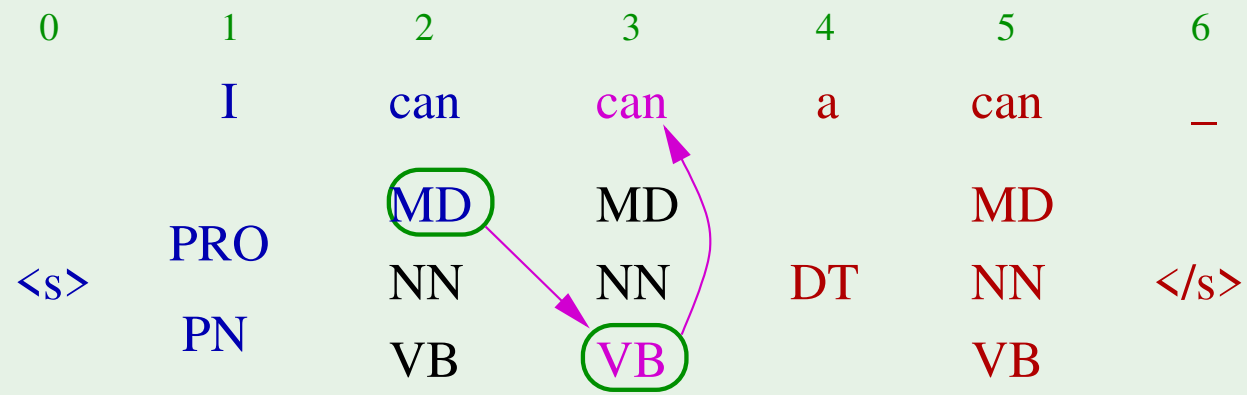
Analog wird die Aposteriori-Wahrscheinlichkeit

$$p(t_{k-1} = t, t_k = t' | w_1^n) = \gamma_{tt'}(k)$$

des Tagpaares t, t' an Position k berechnet:

$$\begin{aligned} \gamma_{tt'}(k) &= \frac{\sum_{t_1^n : t_{k-1}=t, t_k=t'} p(t_1^n, w_1^n)}{\sum_{t_1^n} p(t_1^n, w_1^n)} \\ &= \frac{\alpha_t(k-1)\ p(t'|t)\ p(w_k|t')\ \beta_{t'}(k)}{\alpha_{\langle /s \rangle}(n+1)} \end{aligned}$$

Forward-Backward-Algorithmus



$$\gamma_{MD,VB}(3) = \frac{\alpha_{MD}(2)\ p(VB|MD)\ p(can|VB)\ \beta_{VB}(3)}{\alpha_{\langle/s\rangle}(6)}$$

Forward-Backward-Algorithmus

Summation der Aposteriori-Wahrscheinlichkeiten über alle Sätze **w** im Korpus C und über alle Wortpositionen i im Satz zu erwarteten Häufigkeiten:

$$f_{tw} = \sum_{\mathbf{w} \in C} \sum_{1 \leq i \leq |\mathbf{w}| : w_i = w} \gamma_t(i, \mathbf{w})$$

$$f_{tt'} = \sum_{\mathbf{w} \in C} \sum_{i=1}^{|\mathbf{w}|+1} \gamma_{tt'}(i, \mathbf{w})$$

Der Ausdruck $\sum_{1 \leq i \leq |\mathbf{w}| : w_i = w} \gamma_t(i)$ summiert über alle Positionen $i \in \{1, 2, \ldots, |\mathbf{w}|\}$ mit $w_i = w$. Man kann unter Verwendung der Indikatorfunktion auch schreiben: $\sum_{1 \leq i \leq |\mathbf{w}|} \gamma_t(i, \mathbf{w}) \mathbb{1}_{w_i = w}$

$\gamma_{tt'}(i, \mathbf{w})$ ist der Wert von $\gamma_{tt'}(i)$ für den Satz **w**. $\gamma_t(i, \mathbf{w})$ ist analog der Wert von $\gamma_t(i)$ für den Satz **w**.

Neuschätzung der HMM-Parameter (M-Schritt)

$$p(w|t) = \frac{f_{tw}}{\sum_{w'} f_{tw'}} \qquad p(t'|t) = \frac{f_{tt'}}{\sum_{t''} f_{tt''}}$$

EM-Training eines Wortart-Taggers

1. Uniforme Initialisierung aller $p(t|t')$
2. Uniforme Initialisierung aller $p(w|t)$, die im Lexikon auftauchen
3. Berechnung der erwarteten Wort-Tag- und Tag-Tag-Häufigkeiten mit dem Forward-Backward-Algorithmus (E-Schritt)
4. Neuschätzung der HMM-Wahrscheinlichkeiten aus den erwarteten Häufigkeiten (M-Schritt)
5. weiter mit Schritt 3 bis das Stoppkriterium erfüllt ist

Beispiele von Stoppkriterien:

- N Trainings-Iterationen wurden durchlaufen.
- Die Genauigkeit auf Development-Daten hat abgenommen.

Warum funktioniert das EM-Training?

- Das Lexikon schränkt die möglichen Tags eines Wortes ein.
- Das ermöglicht dem Tagger, etwas über **wahrscheinliche Tagfolgen** zu lernen.
- Die Information über wahrscheinliche Tagfolgen erlaubt dem Tagger, Information über **wahrscheinliche Tags der Wörter** zu lernen.
- Mit dieser Information kann der Tagger sein Wissen über **wahrscheinliche Tagfolgen** verfeinern usw.

Wortart-Tagging

- Typische Tagsetgröße: 50 - 150
- Typische Größe des Trainingskorpus: 100,000 - 1,000,000
- Typische Tagginggenauigkeit: 95%-98%

Überwachtes Training auf einem manuell annotierten Korpus ist meist besser als unüberwachtes EM-Training, weil das annotierte Korpus explizite Information darüber enthält, in welchem Kontext welches Tag verwendet werden muss.

Außer HMMs gibt es weitere Taggingmethoden wie Conditional Random Fields (CRFs) oder neuronale Netze.

Kurzzusammenfassung: Hidden-Markow-Modelle

Ein **Hidden-Markow-Modell** definiert die gemeinsame Wahrscheinlichkeit einer Tagfolge t_1^n und einer Wortfolge w_1^n:

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1}) p(w_i|t_i)$$

Die beste Tagfolge (für $k = 1$) wird mit dem **Viterbi-Algorithmus** berechnet:

$$\begin{aligned} \delta_t(0) &= 1 \text{ falls } t = \langle s \rangle \text{ sonst } 0 \\ \delta_t(i) &= \max_{t'} \delta_{t'}(i-1) p(t|t') p(w_i|t) \end{aligned}$$

Beispiel ($k = 1$): p(PRO V, Er liest) =
p(PRO|$\langle s \rangle$) p(Er|PRO) p(V|PRO) p(liest|V) p($\langle /s \rangle$|V) p(ϵ|$\langle /s \rangle$)

Kurzzusammenfassung: Hidden-Markow-Modelle

Unüberwachtes Training mit dem EM-Algorithmus

1. gegeben: Ein nicht annotiertes Korpus und ein Lexikon
2. Uniforme Initialisierung der Modell-Parameter
3. E-Schritt: Schätzung der erwarteten Tagpaar-Häufigkeiten f(t',t) und Wort-Tag-Häufigkeiten f(w,t) mit dem Forward-Backward-Algorithmus
4. M-Schritt: Neuschätzung der Modellparameter

$$p(t|t') = \frac{f(t',t)}{\sum_{t''} f(t',t'')} \qquad p(w|t) = \frac{f(w,t)}{\sum_{w'} f(w',t)}$$

5. Weiter mit Schritt 3 bis ein Endekriterium erfüllt ist

Kurzzusammenfassung: Hidden-Markow-Modelle

Forward-Backward-Algorithmus

Forward-Wahrscheinlichkeiten

$$\begin{aligned}\alpha_t(0) &= 1 \text{ falls } t = \langle s \rangle \text{ sonst } 0\\ \alpha_t(i) &= \sum_{t'} \alpha_{t'}(i-1) p(t|t') p(w_i|t)\end{aligned}$$

Backward-Wahrscheinlichkeiten

$$\begin{aligned}\beta_t(n+1) &= 1 \text{ falls } t = \langle /s \rangle \text{ sonst } 0\\ \beta_t(i-1) &= \sum_{t'} \beta_{t'}(i) p(t'|t) p(w_i|t')\end{aligned}$$

Aposteriori-Wahrscheinlichkeiten der Tags

$$\begin{aligned}\gamma_t(i) &= \frac{\alpha_t(i)\beta_t(i)}{\alpha_{\langle /s \rangle}(n+1)}\\ \gamma_{t',t}(i) &= \frac{\alpha_{t'}(i-1) p(t|t') p(w_i|t) \beta_t(i)}{\alpha_{\langle /s \rangle}(n+1)}\end{aligned}$$

Kurzzusammenfassung: Hidden-Markow-Modelle

Modifiziertes Hidden-Markow-Modell zur besseren Behandlung **unbekannter Wörter**

$$\hat{t}_1^n = \arg\max_{t_1^n} \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1}) \frac{p(t_i|w_i)}{p(t_i)}$$

Die lexikalische Wahrscheinlichkeit $p(w_i|t_i)$ wird hier durch $p(t_i|w_i)/p(t_i)$ ersetzt.

$p(t|w)$ wird mit Backoff-Glättung geschätzt:

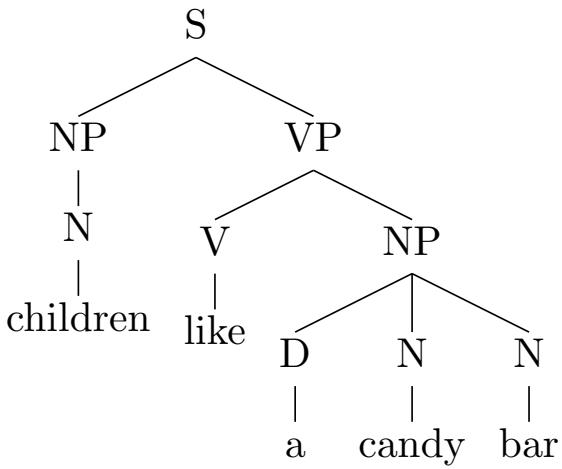
$$p(t|w) = r(t|w) + \alpha(w)p(t|\text{suff}(w))$$

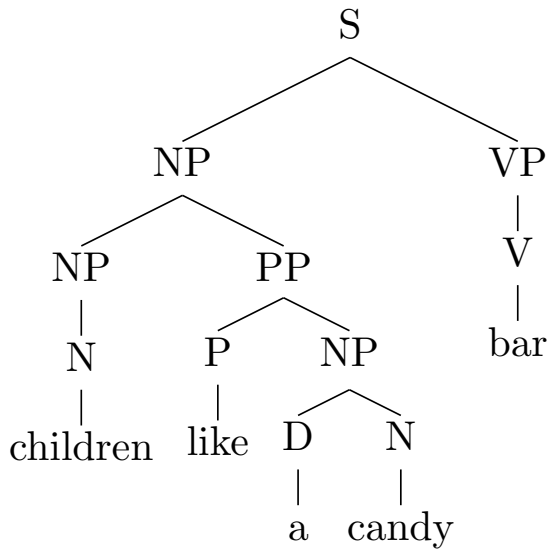
suff(w) berechnet das Suffix der festen Länge L von w.
$p(t|\text{suff}(w))$ wird rekursiv mit Backoff geglättet.

PCFGs: Statistisches Parsen

Syntaktische Ambiguitäten

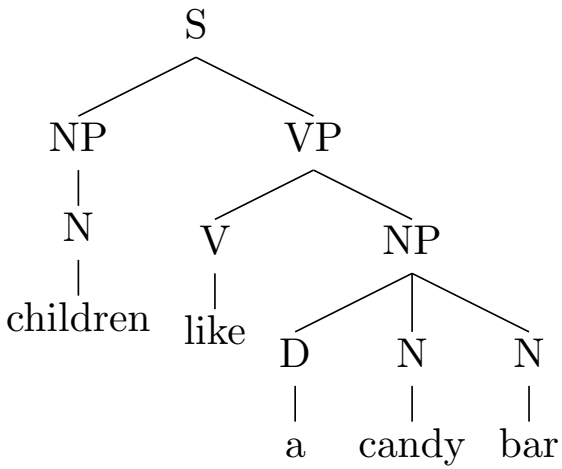
Problem: (Symbolische) Parser liefern für viele Sätze mehrere mögliche syntaktische Analysen (= Parsebäume).

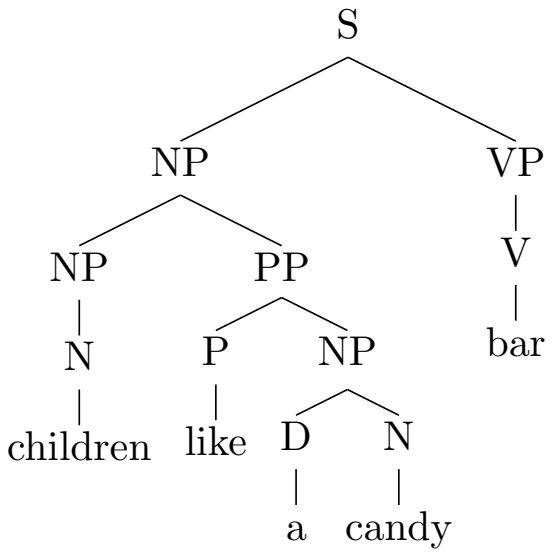




Lösung: Statistische Desambiguierung durch Auswahl des wahrscheinlichsten Parsebaumes.

Statistisches Modell





Wir werden ein statistisches Modell herleiten, welches als Information die Häufigkeiten von Grammatikregeln im Trainingskorpus nutzt.

To Do

Fragen, die wir bei der Entwicklung einer Methode zur statistischen Desambiguierung von syntaktischen Analysen beantworten müssen.

1. Zwischen welchen Alternativen muss desambiguiert werden?
   hier: Parsebäume
2. Wie wird die Wahrscheinlichkeit eines Parsebaumes definiert?
3. Wie werden die Parameter geschätzt?
4. Wie wird die beste Analyse effizient berechnet?

Kontextfreie Grammatiken

Eine kontextfreie Grammatik $G = (V, \Sigma, S, P)$

- leitet aus dem Startsymbol S
- unter Verwendung von Regeln der Form $A \rightarrow \alpha$,
- eine Folge von Terminalsymbolen (z.B. Wörtern) ab.

Dabei ist

- Σ die Menge der Terminalsymbole
- V die Menge der Nichtterminalsymbole
- P die Menge der Grammatikregeln
- S das Startsymbol mit $S \in P$
- α die rechte Seite einer Regel mit $\alpha \in (V \cup \Sigma)^*$

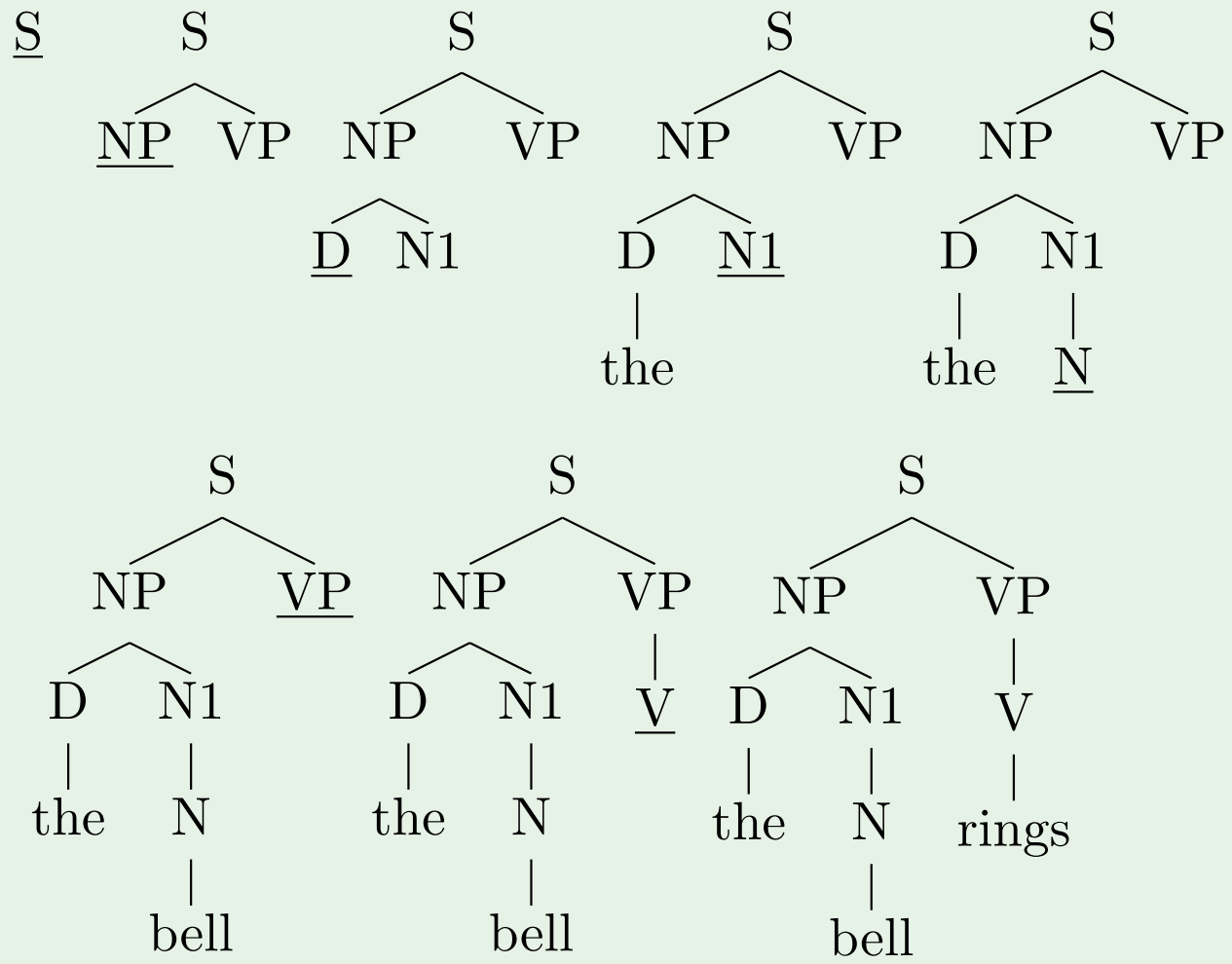
Beispiel einer Ableitung

Grammatik	**Satzform**	**Aktion**
S → NP VP	S	S → NP VP
VP → V NP	NP VP	NP → D N1
VP → V	D N1 VP	D → the
NP → D N1	the N1 VP	N1 → N
N1 → A N1	the N VP	N → bell
N1 → N	the bell VP	VP → V
D → the	the bell V	V → rings
N → bell	the bell rings	
V → rings ...		

Hier handelt es sich um eine Linksableitung, weil immer das am weitesten links stehende Nichtterminal der Satzform expandiert wird.

Analog gibt es Rechtsableitungen.

Folge der partiellen Parsebäume für die Ableitung



Ableitungen

- Für jeden kontextfreien Parsebaum gibt es genau eine **Linksableitung** und umgekehrt.
- Eine Linksableitung ist eindeutig bestimmt durch die Folge der angewandten Regeln $\mathbf{r} = r_1, r_2, ..., r_n$
- Die Wahrscheinlichkeit der Regelfolge wird zerlegt in ein Produkt bedingter Wahrscheinlichkeiten:

$$\begin{aligned} p(r_1, r_2, ..., r_n) &= p(r_1)p(r_2|r_1)...p(r_n|r_1, ..., r_{n-1}) \\ &= \prod_{i=1}^{n} p(r_i|r_1, ..., r_{i-1}) \end{aligned}$$

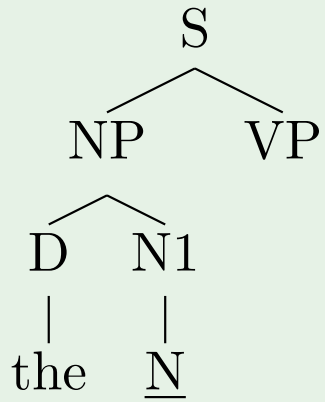
- Der Kontext $r_1, ..., r_{i-1}$ entspricht einem **partiellen Parsebaum** (=Parsebaum, der noch Nichtterminal-Symbole als Blätter haben kann).

Partieller Parsebaum

Zu jeder (partiellen) Linksableitung gibt es einen eindeutigen (partiellen) Parsebaum und umgekehrt.

Beispiel: Regelfolge und partieller Parsebaum

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{NP} &\rightarrow \text{D N1} \\
\text{D} &\rightarrow \text{the} \\
\text{N1} &\rightarrow \text{N}
\end{aligned}
$$



Vereinfachende Annahme: Die Wahrscheinlichkeit einer Regel N $\rightarrow$ bell hängt nur von der **Kategorie** der expandierten Konstituente (hier: N) ab.

$\Rightarrow$ Probabilistische kontextfreie Grammatik (PCFG)

Probabilistische kontextfreie Grammatiken

Wahrscheinlichkeit eines Parsebaumes T mit Linksableitung $r_1, ..., r_n$

$$p(T) = p(r_1, .., r_n) = \prod_{i=1}^{n} p(r_i | r_1, ..., r_{i-1})$$

$$= \prod_{i=1}^{n} p(r_i | nextcat(r_1, ..., r_{i-1}))$$

$nextcat(r_1, ..., r_k)$: Kategorie des Nichtterminalknotens, der als nächster expandiert wird im partiellen Baum von $r_1, ..., r_{i-1}$

Es gilt: $p(A \to \alpha | B) = 0$ falls $A \neq B$ für $A, B \in V$ und $\alpha \in (V \cup \Sigma)^*$

Daraus folgt: $p(r_1, ..., r_k) = 0$ falls $r_1, ..., r_k$ keine (partielle) Linksableitung ist.

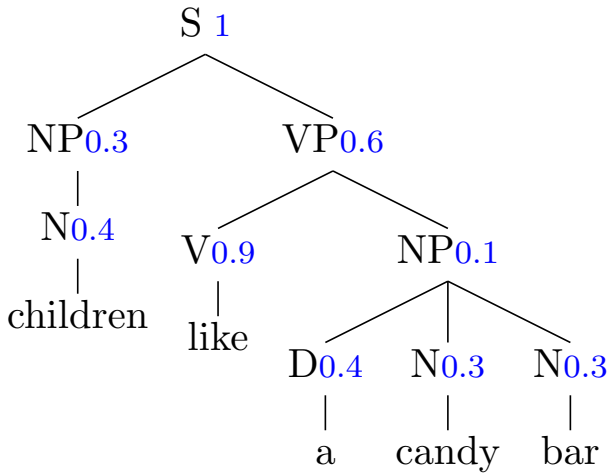
Meistens wird einfach $p(A \to \alpha)$ statt $p(A \to \alpha | A)$ geschrieben. Damit vereinfacht sich die obige Definition zu:

$$p(T) = p(r_1, ..., r_n) = \prod_{i=1}^{n} p(r_i)$$

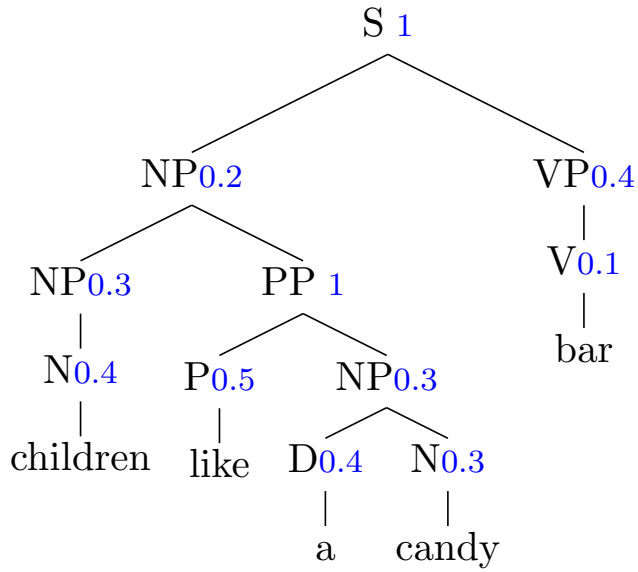
Diese Formel gilt aber nur, wenn $r_1, ..., r_n$ tatsächlich eine Linksableitung ist!

Beispiele für Parsebaum-Wahrscheinlichkeiten

S	→	NP VP	1
NP	→	NP PP	0.2
NP	→	D N	0.3
NP	→	N	0.3
NP	→	D N N	0.1
NP	→	N N	0.1
VP	→	V NP	0.6
VP	→	V	0.4
PP	→	P NP	1
D	→	the	0.6
D	→	a	0.4
N	→	children	0.4
N	→	candy	0.3
N	→	bar	0.3
V	→	bar	0.1
V	→	like	0.9
P	→	like	0.5
P	→	for	0.5



$p_1 = 0.0002333$



$p_2 = 0.0000173$

PCFG-Zusammenfassung

- PCFG = CFG + Regelwahrscheinlichkeiten
- Die Wahrscheinlichkeiten aller Regeln mit derselben linken Seite summieren zu 1, d.h.

$$\sum_{\alpha: A \rightarrow \alpha \in P} p(A \rightarrow \alpha) = 1 \text{ für alle } A \in V$$

- Parsebaum-Wahrscheinlichkeit = Produkt der Regelwahrscheinlichkeiten
- syntaktische Desambiguierung durch Auswahl der wahrscheinlichsten Analyse
- Die Wahrscheinlichkeit eines Satzes S definieren wir als Summe der Wahrscheinlichkeiten all seiner Parsebäume.

Parameterschätzung

Die Regelwahrscheinlichkeiten werden aus Regelhäufigkeiten geschätzt.
Die Regelhäufigkeiten können auf zwei Arten erhalten werden:

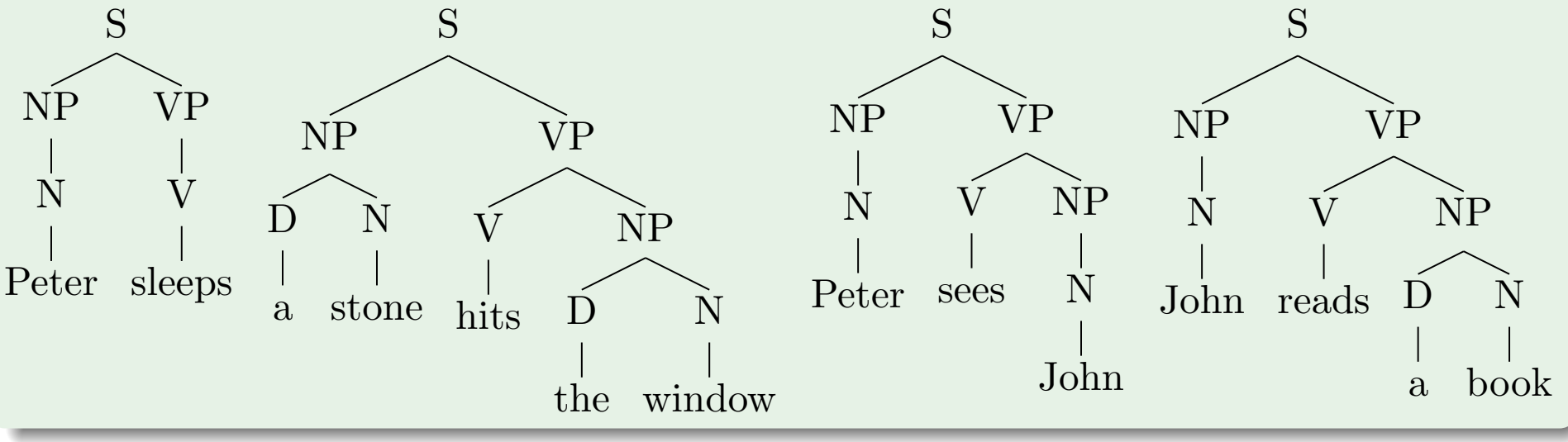
1. durch Zählen der Regeln in einem manuell mit Parsebäumen annotierten Korpus (Baumbank)
2. aus einem automatisch geparsten und desambiguierten Korpus (EM-Training)

Schätzung der Regelwahrscheinlichkeiten mit relativen Häufigkeiten

$$p(A \to \alpha) = \frac{f_{A\to\alpha}}{\sum_\beta f_{A\to\beta}}$$

Eine Parameterglättung ist nicht unbedingt notwendig, da meist alle Regeln beobachtet sind.

Baumbank-Training



Extraktion der Grammatikregeln und ihrer Häufigkeiten:

S → NP VP	4	1
VP → V NP	3	0.75
VP → V	1	0.25
NP → D N	3	0.43
NP → N	4	0.57

D → a	2	0.67
D → the	1	0.33
V → sleeps	1	0.25
V → hits	1	0.25
V → sees	1	0.25
V → reads	1	0.25

N → Peter	2	0.29
N → John	2	0.29
N → stone	1	0.14
N → window	1	0.14
N → book	1	0.14

Baumbanken

- Baumbanken müssen manuell erstellt werden.
- Der Aufbau einer Baumbank ist mühsam und teuer.
- Was tun, wenn keine Baumbank zur Verfügung steht?

Training auf unannotierten Daten

Ich werde nun 3 Ansätze zum unüberwachten Training präsentieren.
Die beiden ersten Ansätze funktionieren nicht gut.

Ansatz 1

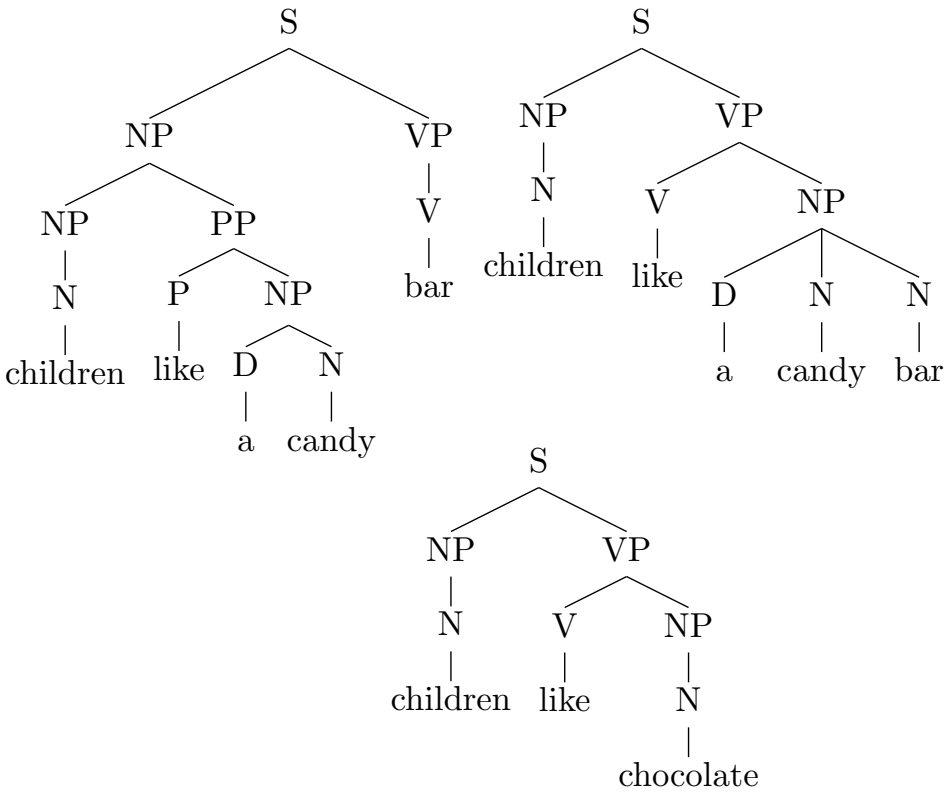
1. Die Sätze des Trainingskorpus werden (symbolisch) geparst.
2. Alle Analysen aller Sätze werden zusammen als "Baumbank" gespeichert.
3. Die Regelwahrscheinlichkeiten werden aus der Baumbank geschätzt.

Problem: Je mehr Analysen ein Satz besitzt, desto höher ist sein Einfluss auf die Grammatik.

Beispiel: Ein nicht ambiger Satz liefert nur einen Parsebaum für die Baumbank. Ein hoch-ambiger Satz kann dagegen Tausende Parsebäume liefern, die sich oft ähneln. Damit hat der ambige Satz einen tausendmal höheren Einfluss auf die Grammatikwahrscheinlichkeiten als der eindeutige Satz.

Training auf unannotierten Daten

children like a candy bar
children like chocolate



Regel			f	p
S	→	NP VP	3	1
NP	→	D N	1	0.14
NP	→	D N N	1	0.14
NP	→	N	4	0.57
NP	→	NP PP	1	0.14
VP	→	V	1	0.33
VP	→	V NP	2	0.67
PP	→	P NP	1	1
D	→	a	2	1
D	→	the	0	0
N	→	children	3	0.43
N	→	bar	1	0.14
N	→	candy	2	0.29
N	→	chocolate	1	0.14
V	→	bar	1	0.33
V	→	like	2	0.67
P	→	like	1	1

"candy" ist doppelt so wahrscheinlich wie "chocolate", obwohl beide in den Sätzen nur einmal auftauchten.

Training auf unannotierten Daten

Ansatz 2: ähnlich zu Ansatz 1 aber:

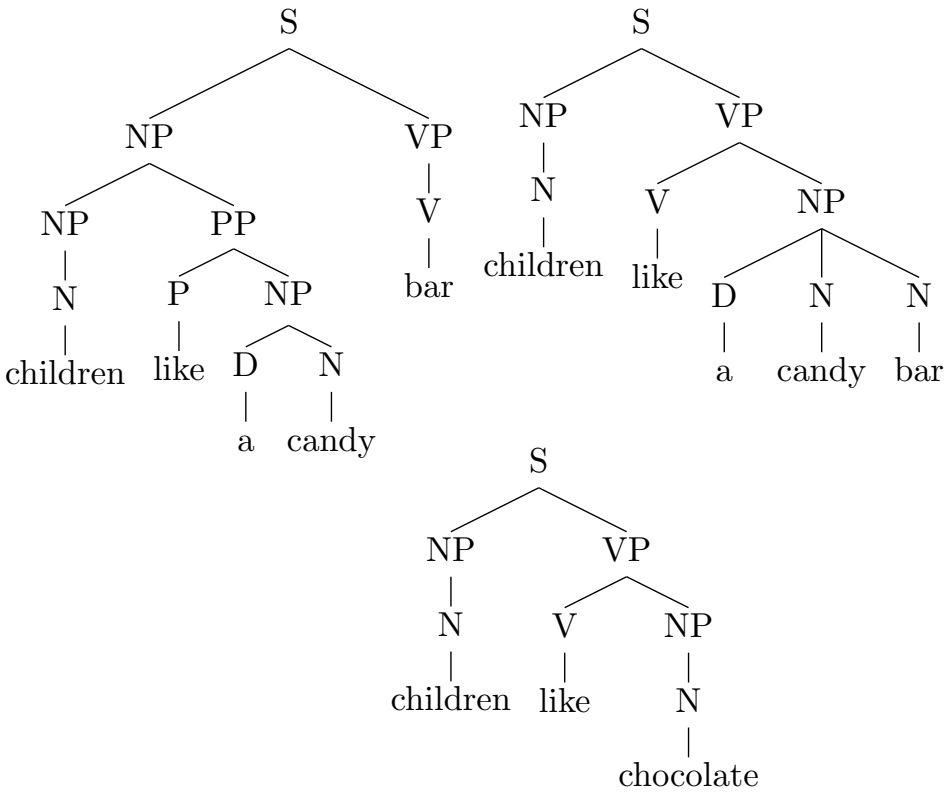
Die Regelhäufigkeiten, die aus den Analysen eines Satzes extrahiert wurden, werden durch die Zahl der Analysen geteilt.

- $\Rightarrow$ Alle Sätze erhalten dasselbe Gewicht.
- $\Rightarrow$ Alle Analysen eines Satzes erhalten dasselbe Gewicht.

Problem: Gute Analysen erhalten dasselbe Gewicht wie schlechte.

Training auf unannotierten Daten

children like a candy bar
children like chocolate



Regel	f	p
S → NP VP	2	1
NP → D N	0.5	0.11
NP → D N N	0.5	0.11
NP → N	3	0.67
NP → NP PP	0.5	0.11
VP → V	0.5	0.25
VP → V NP	1.5	0.75
PP → P NP	0.5	1
D → a	1	1
D → the	0	0
N → children	2	0.44
N → bar	0.5	0.11
N → candy	1	0.22
N → chocolate	1	0.22
V → bar	0.5	0.25
V → like	1.5	0.75
P → like	0.5	1

+ Die Wahrscheinlichkeiten von “candy” und “chocolate” stimmen jetzt.
− Die gute und die schlechte Analyse des ersten Satzes haben dasselbe Gewicht.

Training auf unannotierten Daten

Ansatz 3: Gewichtung der Analysen mit ihrer Güte.

Als Maß für die Güte der Analyse t dient ihre Wahrscheinlichkeit gegeben den Satz s:

$$p(t|s) = \frac{p(s|t)p(t)}{p(s)} = \frac{p(t)}{p(s)} = \frac{p(t)}{\sum_{t' \in T(s)} p(t')}$$

$p(s|t) = 1$, weil die Wortfolge s durch die Terminalsymbole des Parsebaum t gegeben ist.

$T(s)$ ist hier die Menge der Analysen des Satzes s.

Die aus t extrahierten Regelhäufigkeiten werden mit $p(t|s)$ multipliziert.

Aber: Zur Berechnung der Regelwahrscheinlichkeiten $p(t)$ brauchen wir schon die Regelhäufigkeiten, die hier erst bestimmt werden sollen.

⇒ Henne-Ei-Problem

⇒ EM-Training

Training auf unannotierten Daten

Ansatz 3: (Forts.)

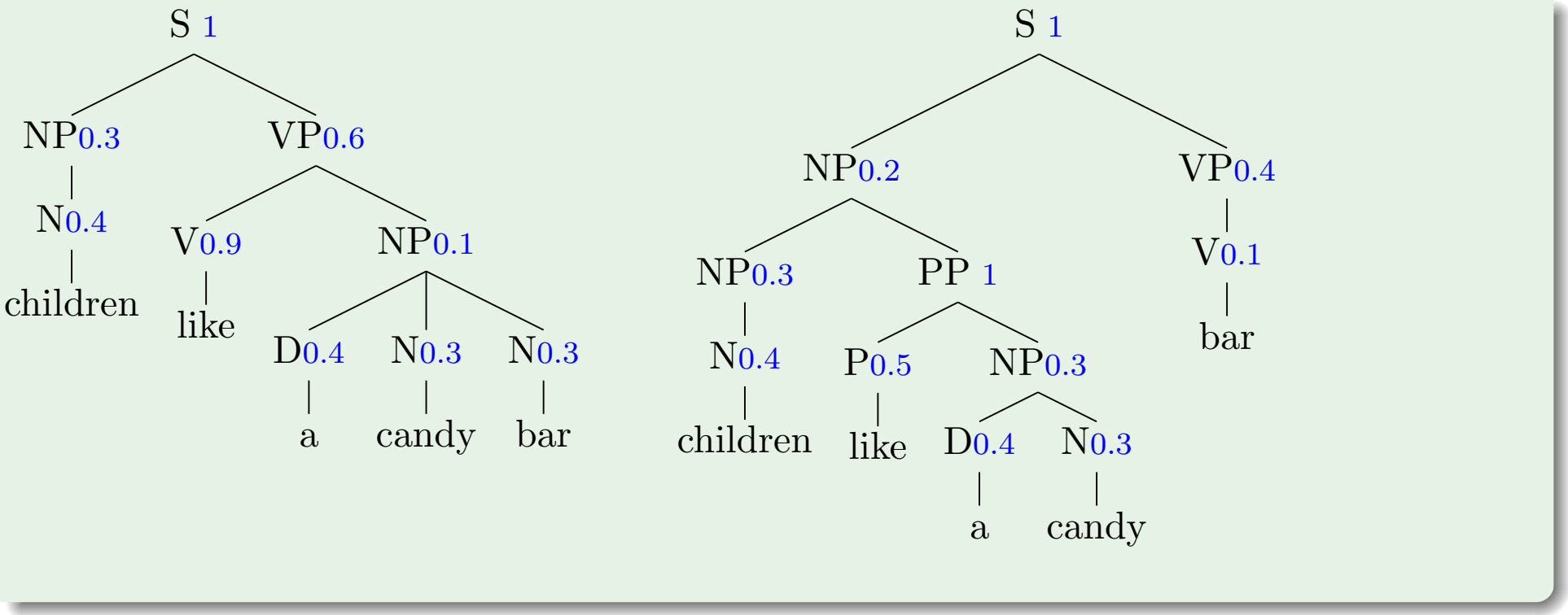
EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

Problem: Wie können die gewichteten Regelhäufigkeiten effizient für hochambige Sätze berechnet werden?

Lösung: Inside-Outside-Algorithmus (wird später vorgestellt)

Beispiel für die Berechnung der Parsebaumgewichte



Zur Berechnung von $p(t_1)$ und $p(t_2)$ siehe Folie 184.

$$p(t_1) = 0.0002333 \qquad p(t_2) = 0.0000173$$

$$p(t_1|s) = \frac{p(t_1)}{p(t_1)+p(t_2)} = 0.93 \qquad p(t_2|s) = \frac{p(t_2)}{p(t_1)+p(t_2)} = 0.07$$

Training auf unannotierten Daten

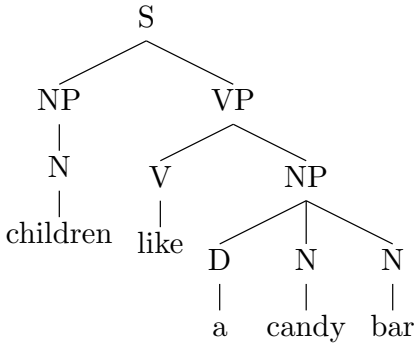
Wichtige Eigenschaft des EM-Algorithmus:

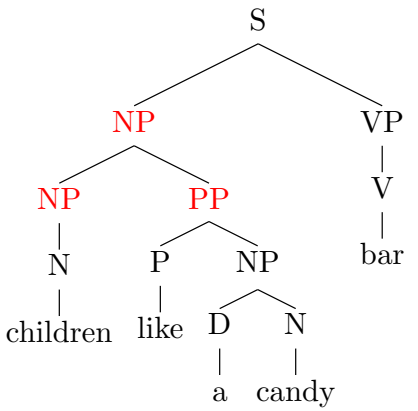
Die Wahrscheinlichkeit der Trainingsdaten steigt bei jeder Iteration, bis sie zu einem (eventuell nur lokalen) Maximum konvergiert.

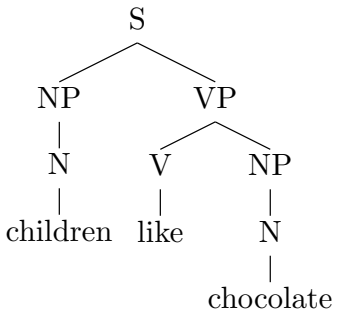
Korpuswahrscheinlichkeit	$p(C) = \prod_{s \in C} p(s)$
Satzwahrscheinlichkeit	$p(s) = \sum_{t \in T(s)} p(t)$
Parsewahrscheinlichkeit	$p(t) = p(r_1, ..., r_n) = \prod_{i=1}^{n} p(r_i)$

$T(s)$ sind die Analysen des Satzes s.

$r_1, \ldots, r_n$ sind die Regeln der Linksableitung von t.







Regel	p_0	f_1	p_1	f_2	p_2	f_3	p_3
S → NP VP	1.00	2.00	1.00	2.00	1.00	2.00	1.00
NP → D N	0.25	0.50	0.11	0.10	0.02	0.00	0.00
NP → D N N	0.25	0.50	0.11	0.90	0.22	1.00	0.25
NP → N	0.25	3.00	0.67	3.00	0.73	3.00	0.75
NP → NP PP	0.25	**0.50**	0.11	**0.10**	0.02	**0.00**	0.00
VP → V	0.50	0.50	0.25	0.10	0.05	0.00	0.00
VP → V NP	0.50	1.50	0.75	1.90	0.95	2.00	1.00
PP → P NP	1.00	0.50	1.00	0.10	1.00	0.00	1.00
D → a	0.50	1.00	1.00	1.00	1.00	1.00	1.00
D → the	0.50	0.00	0.00	0.00	0.00	0.00	0.00
N → bar	0.25	0.50	0.11	0.90	0.18	1.00	0.20
N → candy	0.25	1.00	0.22	1.00	0.20	1.00	0.20
N → children	0.25	2.00	0.44	2.00	0.41	2.00	0.40
N → chocolate	0.25	1.00	0.22	1.00	0.20	1.00	0.20
V → bar	0.50	0.50	0.25	0.10	0.05	0.00	0.00
V → like	0.50	1.50	0.75	1.90	0.95	2.00	1.00
P → like	1.00	0.50	1.00	0.10	1.00	0.00	1.00
-logprob		15.2		11.3		9.3	

Im Trainingsverlauf wird die Wahrscheinlichkeit (und die erwartete Häufigkeit) der Regel NP → NP PP immer kleiner. Der erste Satz wird dadurch zugunsten seiner ersten Analyse desambiguiert, die der Analyse des zweiten Satzes ähnlicher ist.

Die Wahrscheinlichkeit der Trainingsdaten steigt monoton. (Der negative Logarithmus der Wk. sinkt entsprechend.)

Zusammenfassung

- Baumbanktraining
  - benötigt eine manuell erstellte Baumbank.
  - Die Grammatik wird aus dem Baumbank extrahiert.
  - Die Regelhäufigkeiten werden gezählt.
  - Die Regelwahrscheinlichkeiten werden mit relativen Häufigkeiten geschätzt:

$$p(A \to \alpha) = \frac{f_{A\to\alpha}}{\sum_\beta f_{A\to\beta}}$$

- EM-Training
  - Training auf nicht annotierten Texten
  - Eine kontextfreie Grammatik ist gegeben.
  - Die Regelwahrscheinlichkeiten werden uniform initialisiert.
  - Ein symbolischer Parser liefert die möglichen Analysen.
  - Die aus jedem Parsebaum t extrahierten Regelhäufigkeiten werden gewichtet mit

$$p(t|s) = \frac{p(t)}{\sum_{t'\in T(s)} p(t')}$$

  - danach Neuschätzung der Regelwahrscheinlichkeiten und Iteration

Berechnung der besten Analyse eines Satzes

Naiver Ansatz

1. Zähle alle möglichen Parsebäume auf.
2. Berechne die Wahrscheinlichkeit jedes Parsebaumes.
3. Gib den wahrscheinlichsten Parsebaum zurück.

Problem: Aufzählung aller Parsebäume ist zu ineffizient.

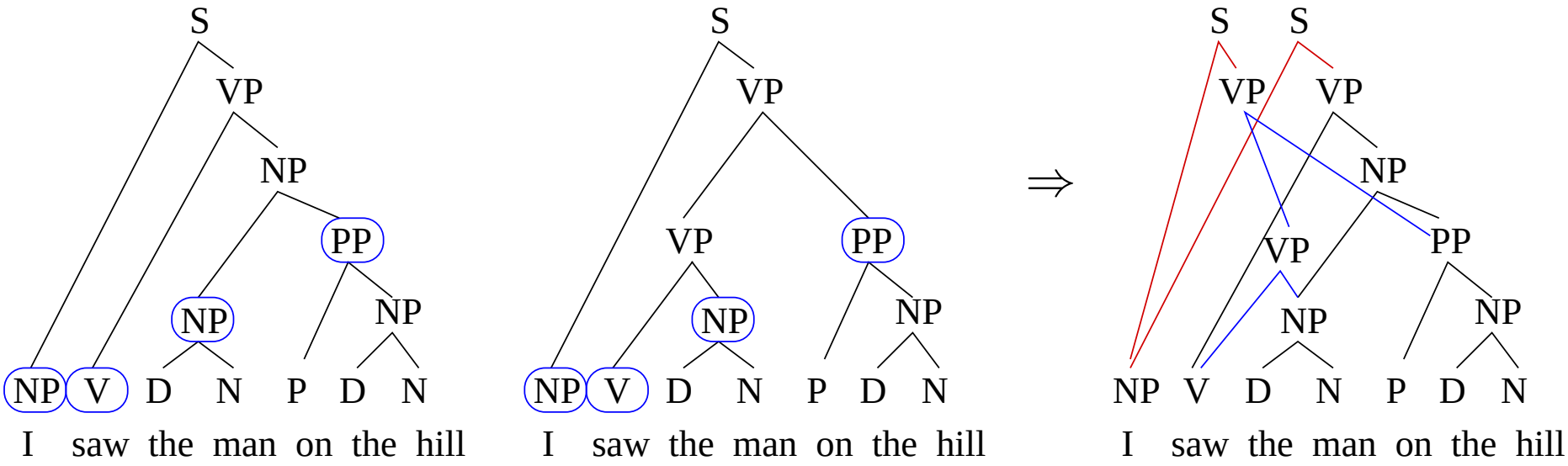
Lösung: Extraktion des besten Parsebaumes aus einer kompakten Repräsentation der Menge aller Parsebäume (genannt **Parsewald**)

Parsewald

- eine kompakte Repräsentation der Menge aller Analysen eines Satzes
- ergibt sich aus der Menge der Parsebäume durch 2 Operationen:
  1. Zusammenfassen gemeinsamer Teilbäume
  2. Zusammenfassen von Parsebäumen, die sich nur in einem Teilbaum unterscheiden.

Beispiel

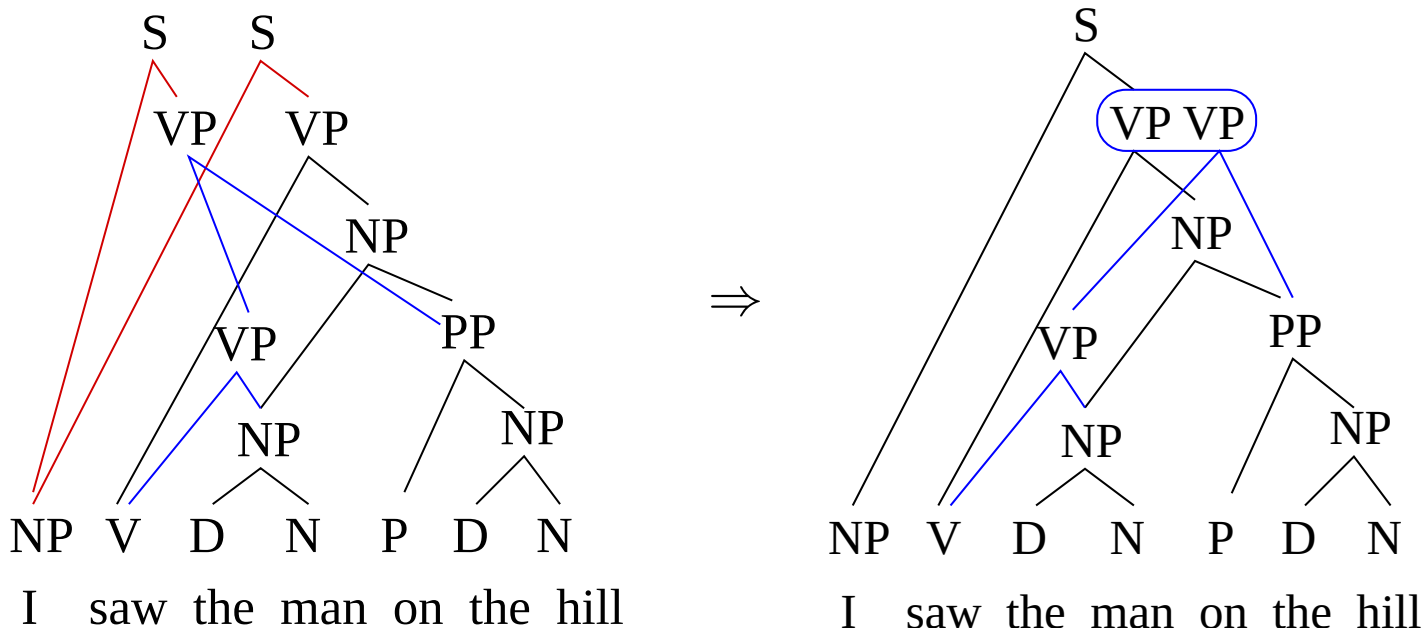
Zusammenfassen (maximaler) gemeinsamer Teilbäume



Die blauen Knoten der beiden Parsebäume werden jeweils zu einem Knoten zusammengefasst.

Beispiel

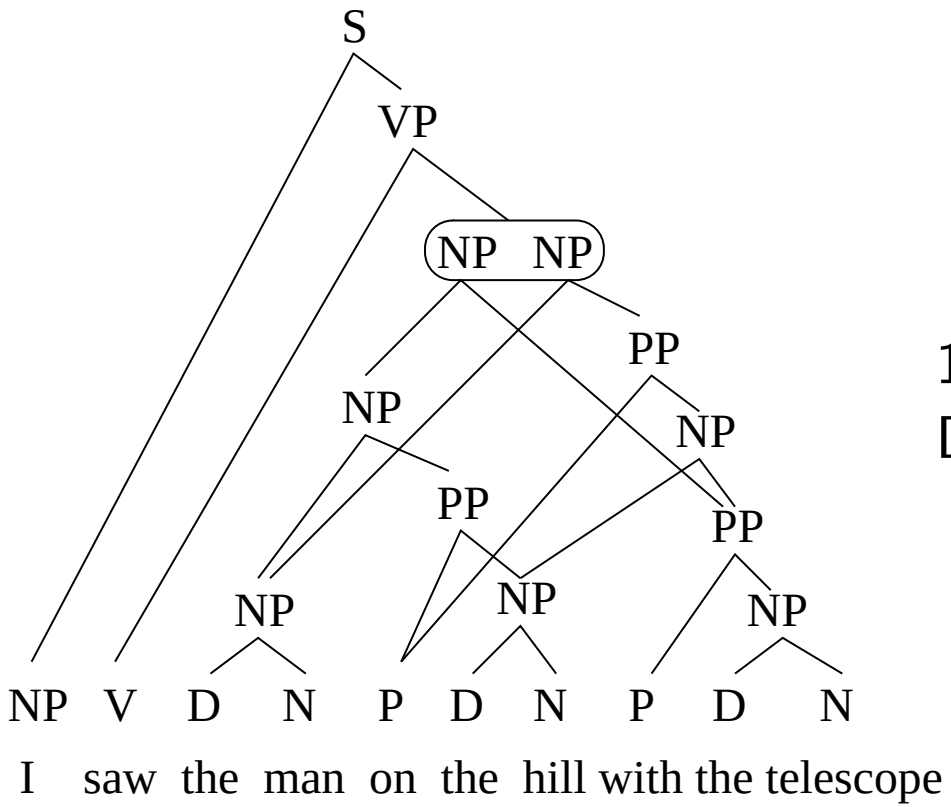
Zusammenfassen von Parsebäumen, die sich nur in einem Teilbaum unterscheiden



Hier werden die beiden roten Teilbäume zusammengefasst.
Die beiden VP-Teilbäume werden in einem ambigen VP-Knoten vereinigt.
Die blauen Kanten gehören zum 2. Parsebaum.

Parsewälder als spezialisierte Grammatiken

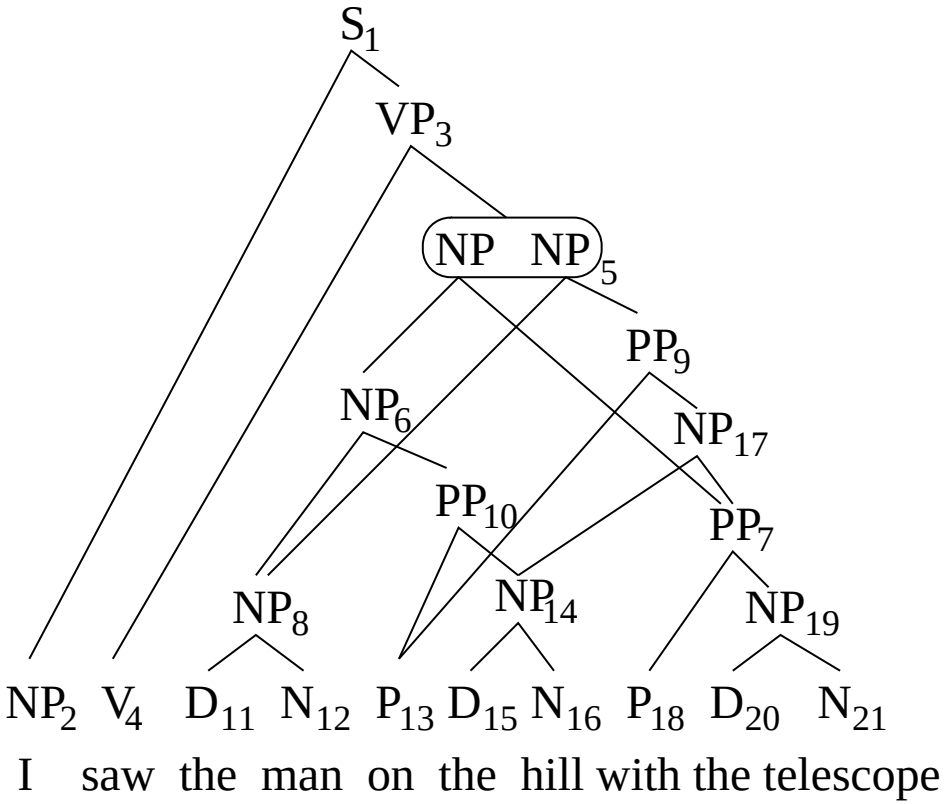
Wir definieren nun für jeden Parsewald eine Grammatik, die genau die darin enthaltenen Parsebäume generiert.



1. Schritt:
Durchnumerierung der Knoten

Parsewald als spezialisierte Grammatik

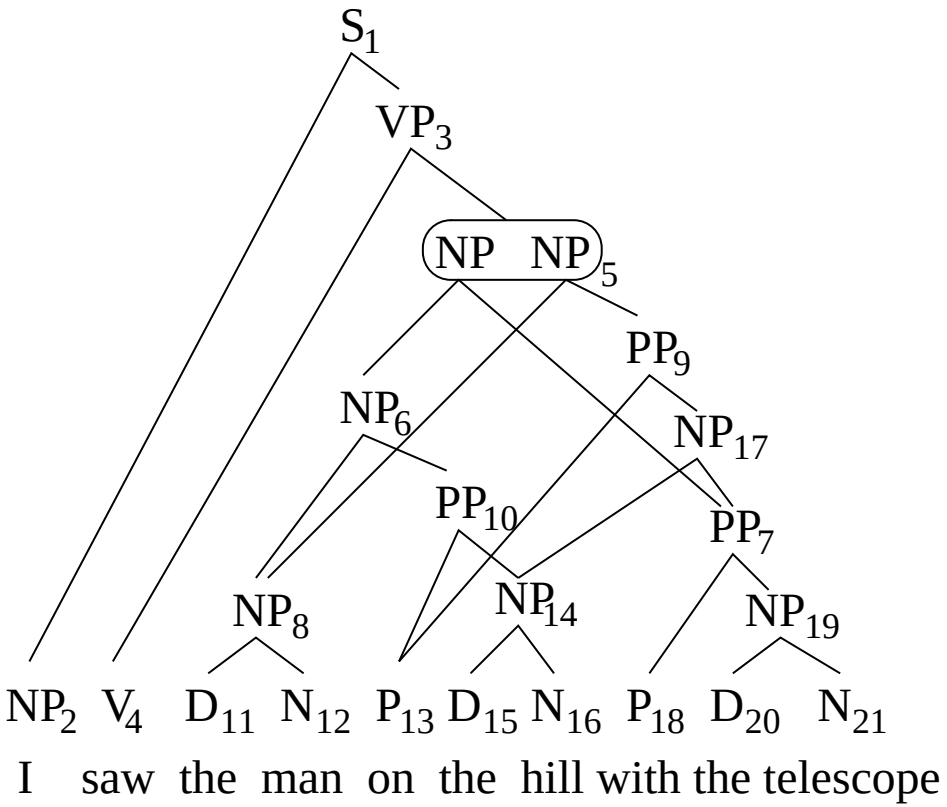
2. Schritt: Extraktion der Grammatikregeln



$$
\begin{aligned}
S_1 &\rightarrow NP_2\ VP_3 \\
NP_2 &\rightarrow I \\
VP_3 &\rightarrow V_4\ NP_5 \\
V_4 &\rightarrow saw \\
NP_5 &\rightarrow NP_6\ PP_7 \\
NP_5 &\rightarrow NP_8\ PP_9 \\
NP_6 &\rightarrow NP_8\ PP_{10} \\
NP_8 &\rightarrow D_{11}\ N_{12} \\
D_{11} &\rightarrow the \\
N_{12} &\rightarrow man \\
PP_{10} &\rightarrow P_{13}\ NP_{14} \\
NP_{14} &\rightarrow \ldots
\end{aligned}
$$

Diese Grammatik generiert genau die Parsebäume, die zu dem Parsewald zusammengefasst wurden (wenn man ignoriert, dass die Symbole noch Indizes tragen).

Grundidee des Viterbi-Algorithmus



- Die Konstituenten (Knoten) des Parsewaldes werden **bottom-up** durchwandert.
- Für jede Konstituente wird ihre **wahrscheinlichste Analyse** berechnet.
- Am Wurzelknoten wird die wahrscheinlichste **Satzanalyse** ausgegeben.

Viterbi-Algorithmus und Parsewald-Grammatiken

- Wir berechnen für jede Regel und jedes Nichtterminal der Parsewald-Grammatik eine **Viterbi-Wahrscheinlichkeit**.
- Die Viterbi-Wahrscheinlichkeit einer **Parsewaldregel** ist das Produkt aus der Wahrscheinlichkeit der entsprechenden PCFG-Regel und den Viterbi-Wahrscheinlichkeiten der Nichtterminale auf der rechten Seite:

$$\delta(\mathrm{NP}_5 \rightarrow \mathrm{NP}_6\ \mathrm{PP}_7) = \mathrm{p}(\mathrm{NP} \rightarrow \mathrm{NP}\ \mathrm{PP})\ \delta(\mathrm{NP}_6)\ \delta(\mathrm{PP}_7)$$

- Die Viterbi-Wahrscheinlichkeit eines **Parsewald-Nichtterminals** ist die maximale Wahrscheinlichkeit aller Parsewald-Regeln mit diesem Nichtterminal als linker Seite:

$$\delta(A) = \max_{A \rightarrow \alpha} \delta(A \rightarrow \alpha)$$

Berechnung der Viterbi-Wahrscheinlichkeiten

Viterbi-Algorithmus

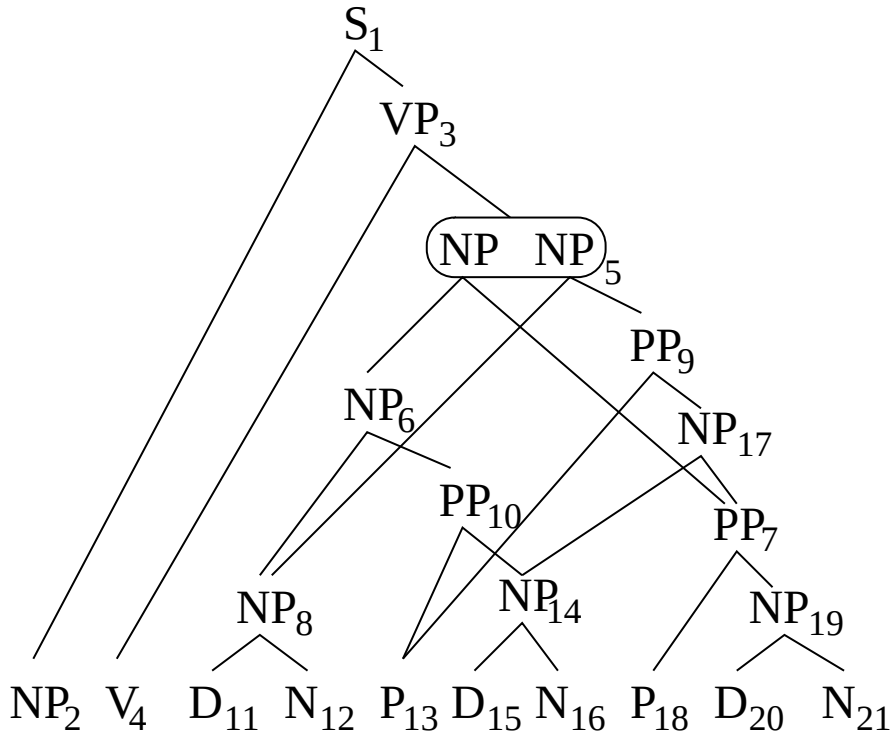
$$
\begin{aligned}
\delta(a) &= 1 \quad \text{für jedes Terminalsymbol a} \\
\delta(A \rightarrow X_1 \ldots X_n) &= p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i) \quad \text{für Parsewald-Regeln} \\
\delta(A) &= \max_{\alpha} \delta(A \rightarrow \alpha) \quad \text{für Nichtterminale A} \\
\psi(A) &= \arg\max_{\alpha} \delta(A \rightarrow \alpha) \quad \text{beste Analyse von A}
\end{aligned}
$$

$p(S_1 \rightarrow NP_2\ VP_3)$ ist die Wahrscheinlichkeit $p(S \rightarrow NP\ VP)$ der entsprechenden PCFG-Regel.

Die ψ-Variable speichert die beste Parsewald-Regel für jedes Nichtterminal.

Viterbi-Beispiel

$$\delta(I) = 1$$
$$\delta(saw) = 1$$
$$\ldots$$
$$\delta(NP_2) = \delta(NP_2 \rightarrow I)$$
$$= p(NP \rightarrow I)\ \delta(I)$$
$$\delta(V_4) = p(V \rightarrow saw)\ \delta(saw)$$
$$\ldots$$
$$\delta(NP_8) = p(NP \rightarrow D\ N)\ \delta(D_{11})\ \delta(N_{12})$$
$$\ldots$$
$$\delta(NP_6) = p(NP \rightarrow NP\ PP)\ \delta(NP_8)\ \delta(PP_{10})$$
$$\ldots$$
$$\delta(NP_5) = \max(p(NP \rightarrow NP\ PP)\ \delta(NP_6)\ \delta(PP_7), p(NP \rightarrow NP\ PP)\ \delta(NP_8)\ \delta(PP_9))$$
$$\ldots$$
$$\delta(S_1) = p(S \rightarrow NP\ VP)\ \delta(NP_2)\ \delta(VP_3)$$



Inside-Outside-Algorithmus

Der **Inside-Outside-Algorithmus**

- berechnet effizient die **erwarteten Regelhäufigkeiten** beim EM-Training
- und entspricht damit dem Forward-Backward-Algorithmus bei den HMMs.
- Er berechnet bottom-up **Inside**-Wahrscheinlichkeiten (ähnlich dem Viterbi-Algorithmus)
- und top-down **Outside**-Wahrscheinlichkeiten.
- Aus den Inside- und Outside-Wahrscheinlichkeiten berechnet er die **erwarteten Häufigkeiten**.

Inside-Wahrscheinlichkeiten

- Der **Viterbi**-Algorithmus berechnet die Wahrscheinlichkeit der **besten Analyse** jedes Parsewaldknotens
- Der **Inside**-Algorithmus berechnet die Gesamtwahrscheinlichkeit **aller Analysen** für jeden Parsewaldknoten.

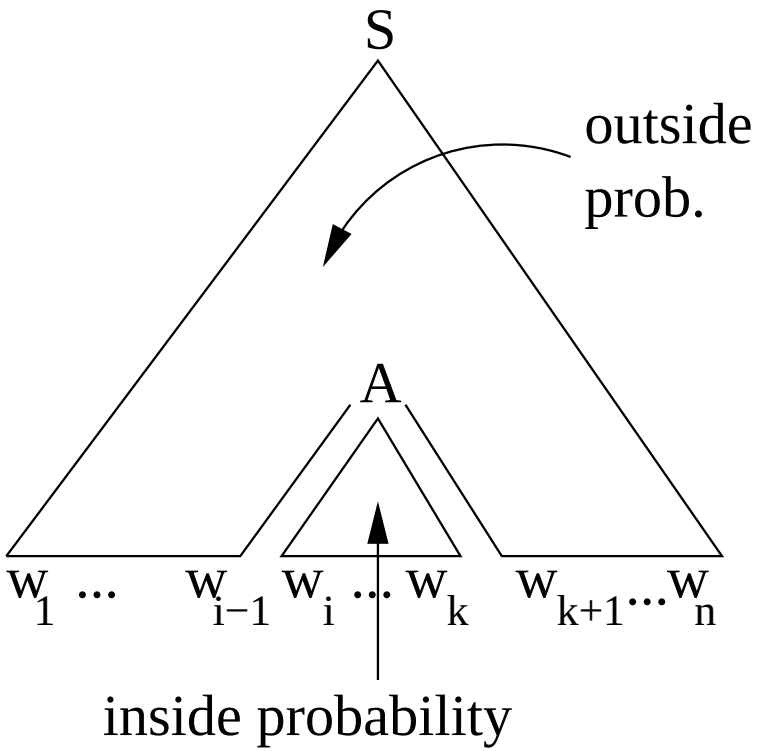
Inside-Algorithmus

$$\begin{aligned}
\alpha(a) &= 1 \quad \text{für Terminalsymbol a} \\
\alpha(A \rightarrow X_1 ... X_n) &= p(A \rightarrow X_1 ... X_n) \prod_{i=1}^{n} \alpha(X_i) \text{ für Parsewaldregeln} \\
\alpha(A) &= \sum_{A \rightarrow \gamma} \alpha(A \rightarrow \gamma) \quad \text{für Nichtterminale A}
\end{aligned}$$

Der Unterschied zum Viterbi-Algorithmus ist die Summen-Operation statt der max-Operation.

Outside-Wahrscheinlichkeiten

Angenommen der Parsewald für den Satz $w_1 \ldots w_n$ enthält eine Konstituente der Kategorie A, die zu $w_i \ldots w_k$ expandiert.



Inside-Wahrscheinlichkeit von A: Gesamtwahrscheinlichkeit aller Ableitungen

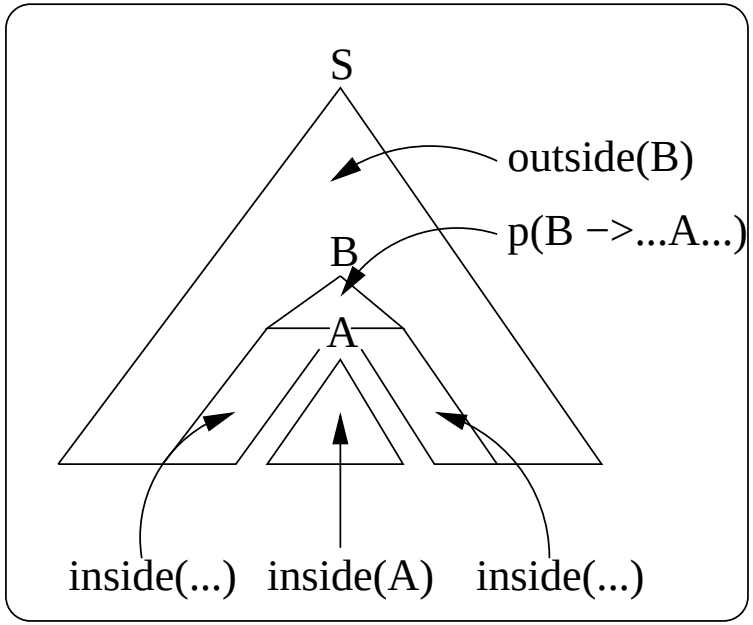
$$A \Rightarrow \ldots \Rightarrow w_i \ldots w_k$$

Outside-Wahrscheinlichkeit von A: Gesamtwahrscheinlichkeit aller Ableitungen

$$S \Rightarrow \ldots \Rightarrow w_1 \ldots w_{i-1} A w_{k+1} \ldots w_n$$

Berechnung der Outside-Wahrscheinlichkeiten

Outside-Algorithmus



$$\beta(S) = 1 \quad \text{für Startsymbol S}$$

$$\beta(A) = \sum_{B \to \gamma A \delta} \beta(B \to \gamma \underline{A} \delta)$$

$$\beta(B \to X_1 ... X_m \underline{A} X_{m+1} ... X_n) = \beta(B) p(B \to X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i)$$

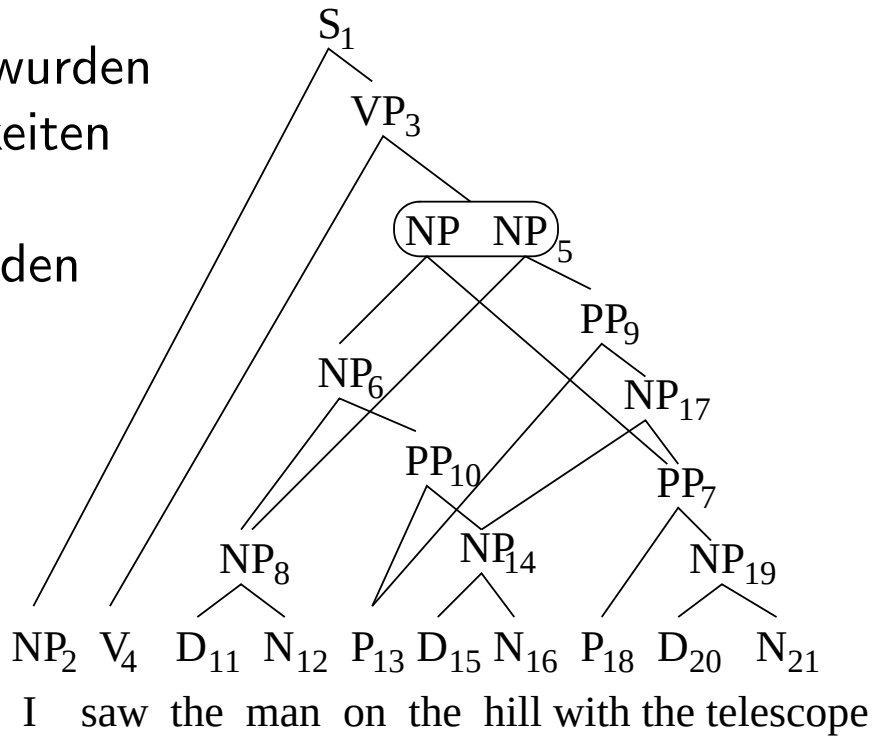
$$\beta(B \to \gamma \underline{A} \delta) = \beta(B) \frac{\alpha(B \to \gamma A \delta)}{\alpha(A)} \quad \text{mit } \gamma = X_1 ... X_m \text{ und } \delta = X_{m+1}, , , X_n$$

$\beta(B \to \gamma \underline{A} \delta)$: Beitrag der Regel $B \to \gamma A \delta$ zur Outside-Wahrscheinlichkeit von A

Beachte, dass $p(B \to X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i) = \alpha(B \to X_1 ... X_m A X_{m+1} ... X_n) / \alpha(A)$

Outside-Beispiel

Die Inside-Wahrscheinlichkeiten $\alpha(.)$ wurden schon berechnet. Die Regelwahrscheinlichkeiten $p(S \rightarrow NP\ VP)$ etc. sind gegeben.
Die Outside-Wahrscheinlichkeiten $\beta(.)$ werden top-down berechnet.



$$\beta(S) = 1$$

$$\beta(NP_2) = \beta(S_1)\, p(S \rightarrow NP\ VP)\, \alpha(VP_3)$$

$$\beta(VP_3) = \beta(S_1)\, p(S \rightarrow NP\ VP)\, \alpha(NP_2)$$

...

$$\beta(NP_8) = \beta(NP_6)\, p(NP \rightarrow NP\ PP)\, \alpha(PP_{10}) + \beta(NP_5)\, p(NP \rightarrow NP\ PP)\, \alpha(PP_9)$$

...

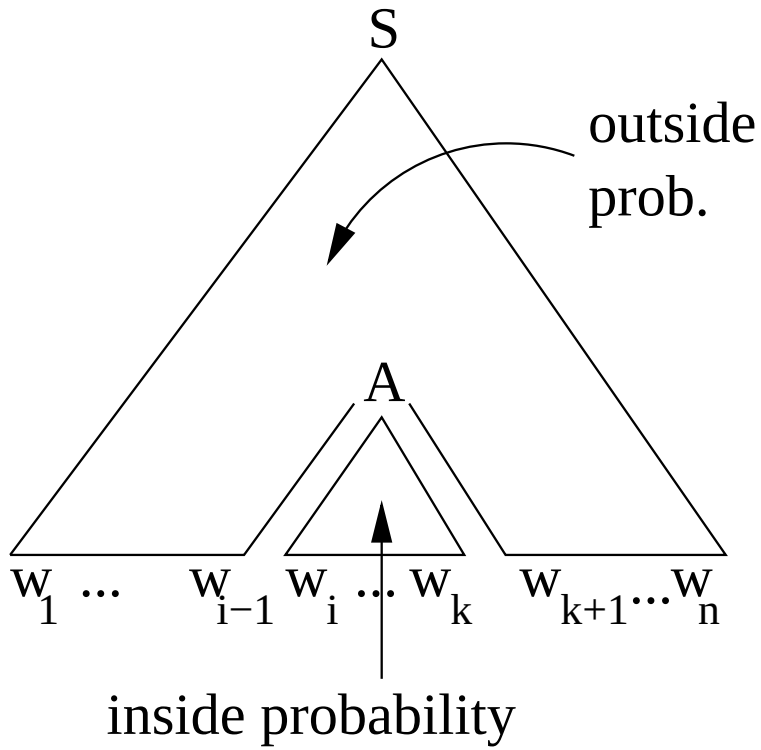
Inside-Outside-Algorithmus

- Das Produkt der Inside- und Outside-Wahrscheinlichkeiten eines Parsewaldknotens ergibt die **Gesamtwahrscheinlichkeit aller Parsebäume mit diesem Knoten**.
- Durch Division dieses Produktes mit der Wahrscheinlichkeit aller Analysen erhält man die **erwartete Häufigkeit** des Parsewaldknotens.

$$\gamma(A) = \alpha(A)\beta(A)/\alpha(S)$$

- erwartete Regelhäufigkeit

$$\gamma(A \to \delta) = \alpha(A \to \delta)\,\beta(A)/\alpha(S)$$



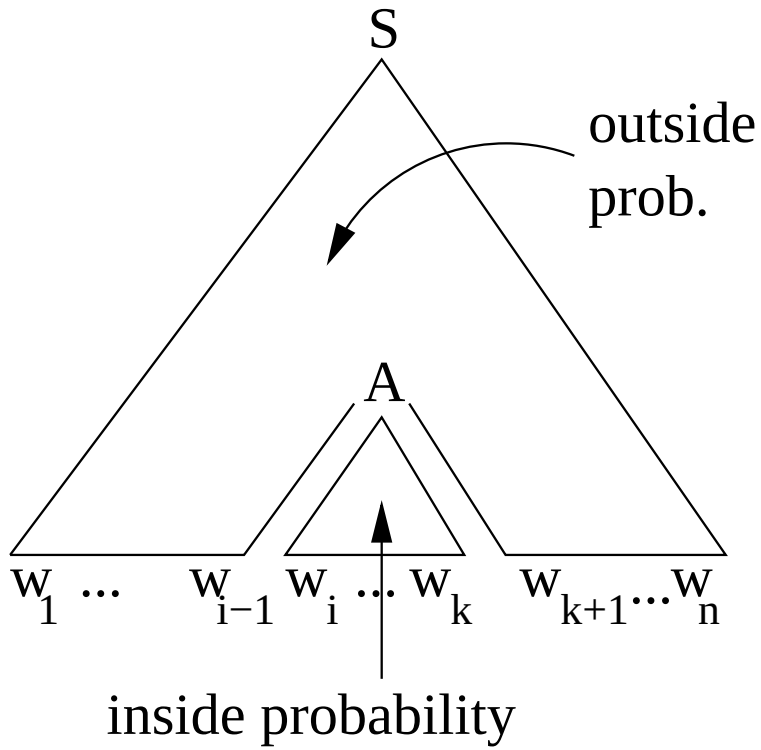
Inside-Outside-Algorithmus

- Das Produkt der Inside- und Outside-Wahrscheinlichkeiten eines Parsewaldknotens ergibt die **Gesamtwahrscheinlichkeit aller Parsebäume mit diesem Knoten**.
- Durch Division dieses Produktes mit der Wahrscheinlichkeit aller Analysen erhält man die **erwartete Häufigkeit** des Parsewaldknotens.

$$\gamma(A) = \alpha(A)\beta(A)/\alpha(S)$$

- erwartete Regelhäufigkeit

$$\gamma(A \rightarrow \delta) = \alpha(A \rightarrow \delta)\ \beta(A)/\alpha(S)$$

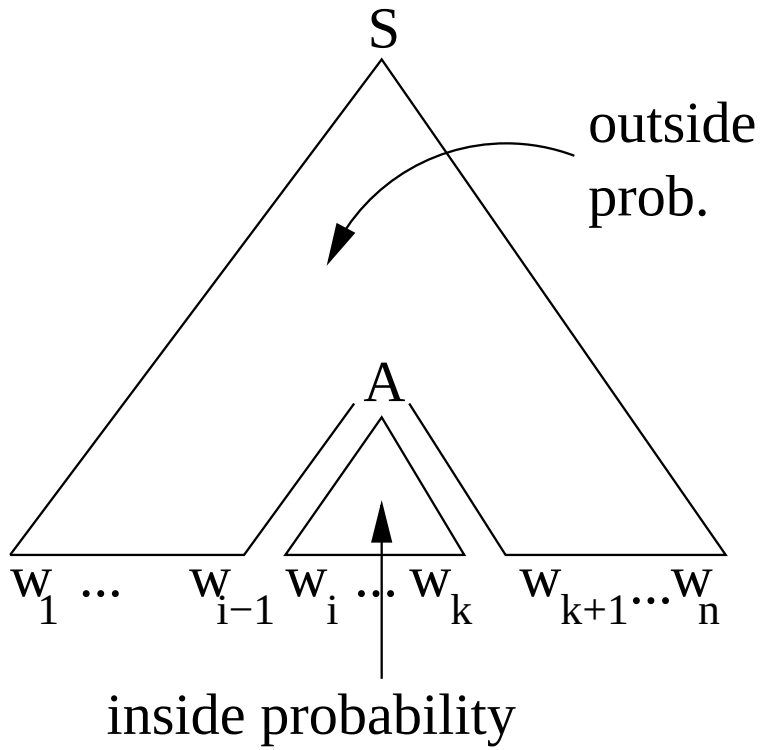


Inside-Outside-Algorithmus

- Das Produkt der Inside- und Outside-Wahrscheinlichkeiten eines Parsewaldknotens ergibt die **Gesamtwahrscheinlichkeit aller Parsebäume mit diesem Knoten**.
- Durch Division dieses Produktes mit der Wahrscheinlichkeit aller Analysen erhält man die **erwartete Häufigkeit** des Parsewaldknotens.

  $\gamma(A) = \alpha(A)\beta(A)/\alpha(S)$
- erwartete Regelhäufigkeit

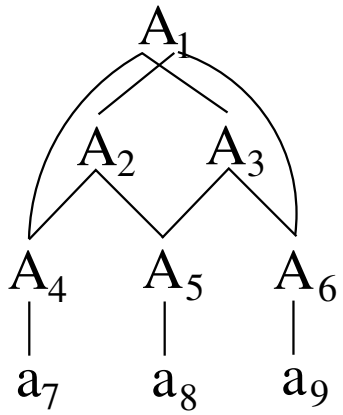
  $\gamma(A \to \delta) = \alpha(A \to \delta)\ \beta(A)/\alpha(S)$



Inside-Outside-Beispiel

PCFG:

$A \rightarrow A\ A$	0.6
$A \rightarrow a$	0.4



$$
\begin{aligned}
\alpha(a_7) &= \mathbf{1} \quad = \alpha(a_8) = \alpha(a_9) \\
\alpha(A_4) &= p(A \rightarrow a)\ \alpha(a_7) = 0.4 * 1 = \mathbf{0.4} \ = \alpha(A_5) = \alpha(A_6) \\
\alpha(A_2) &= p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_5) = 0.6 * 0.4 * 0.4 = \mathbf{0.096} \\
\alpha(A_3) &= \alpha(A_2) \\
\alpha(A_1) &= p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_3) + p(A \rightarrow A\ A)\ \alpha(A_2)\ \alpha(A_6) \\
&= 0.6 * 0.4 * 0.096 * 2 = \mathbf{0.04608} \\
\beta(A_1) &= 1 \\
\beta(A_2) &= \beta(A_1)\ p(A \rightarrow A\ A)\ \alpha(A_6) = 1 * 0.6 * 0.4 = \mathbf{0.24} = \beta(A_3) \\
\beta(A_4) &= \beta(A_1)\ p(A \rightarrow A\ A)\ \alpha(A_3) + \beta(A_2)\ p(A \rightarrow A\ A)\ \alpha(A_5) \\
&= 1 * 0.6 * 0.096 + 0.24 * 0.6 * 0.4 = \mathbf{0.1152} = \beta(A_6) \\
\beta(A_5) &= \beta(A_2)\ p(A \rightarrow A\ A)\ \alpha(A_4) + \beta(A_3)\ p(A \rightarrow A\ A)\ \alpha(A_6) \\
&= 0.24 * 0.6 * 0.4 * 2 = \mathbf{0.1152} \\
\beta(a_7) &= \beta(A_4)\ p(A \rightarrow a) = 0.1152 * 0.4 = \mathbf{0.04608}
\end{aligned}
$$

$$
\begin{aligned}
\gamma(A_2 \rightarrow A_4\ A_5) &= \beta(A_2) p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_5) / \alpha(A_1) \\
&= 0.24 * 0.6 * 0.4 * 0.4 / 0.04608 = 0.5
\end{aligned}
$$

Neuschätzung der Regel-Wahrscheinlichkeiten

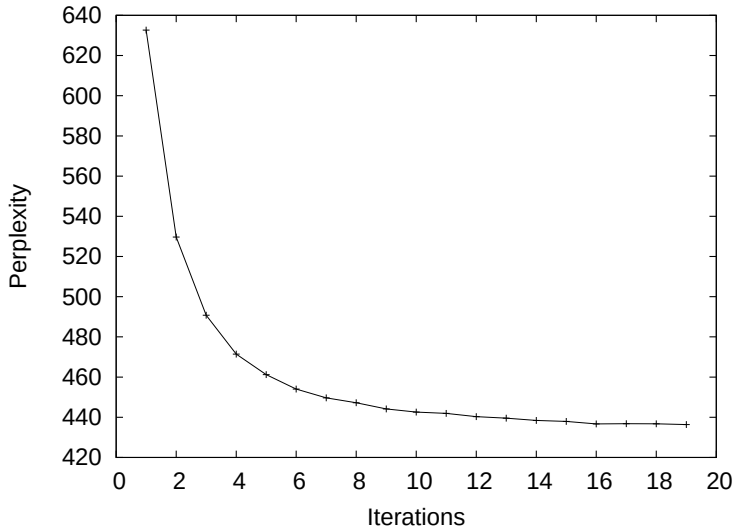
EM-Algorithmus: wiederholte Ausführung der beiden Schritte

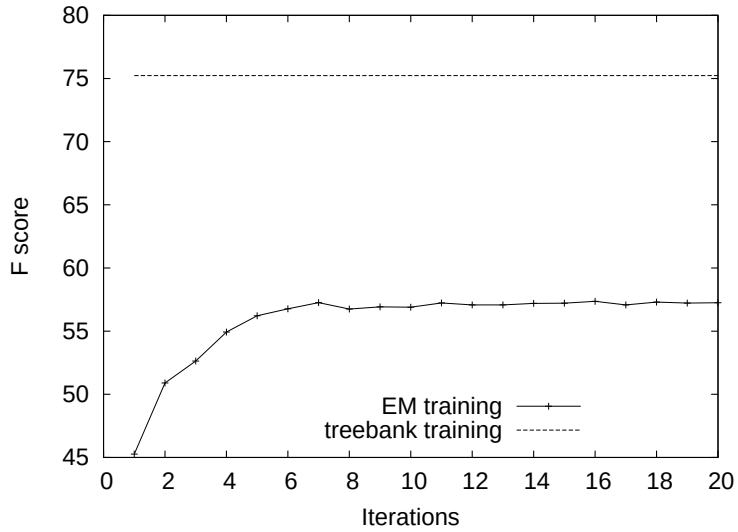
1. E-Schritt
   - Jeder Satz des Trainingskorpus wird geparst und ein Parsewald erstellt.
   - Die erwartete Häufigkeit $\gamma(A \rightarrow \delta)$ jeder Parsewaldregel $A \rightarrow \delta$ wird berechnet.
   - Die erwarteten Häufigkeiten $\gamma(A \rightarrow \delta)$ werden für jede CFG-Regel über alle Trainingssätze zu $f(A' \rightarrow \delta')$ summiert, wobei sich A' und δ' aus A und δ durch Entfernen der Knotennummern ergeben.
2. M-Schritt
   Die Regel-Wahrscheinlichkeiten werden aus den erwarteten Häufigkeiten neu geschätzt:

$$p(A \rightarrow \delta) = \frac{f(A \rightarrow \delta)}{\sum_{\delta'} f(A \rightarrow \delta')}$$

Vor- und Nachteile des EM-Algorithmus

- EM passt die Parameter an die Trainingsdaten an.
- Die Wahrscheinlichkeit der Trainingsdaten steigt monoton und konvergiert zu einem (lokalen) Maximum.





- Die Maximierung der Wahrscheinlichkeit der Trainingsdaten ist nicht dasselbe wie die Maximierung der Parsinggenauigkeit.
- Wenn eine genügend große Baumbank verfügbar ist, liefert das Baumbank-Training bessere Ergebnisse als EM-Training.

Parser-Evaluierung

Option 1

- Testsätze werden geparst und manuell evaluiert.
- Genauigkeit = Prozentsatz der korrekten Parsebäume

Probleme

1. Jede Evaluierung erfordert großen manuellen Aufwand.
2. Die Genauigkeit wird überschätzt, wenn Fehler übersehen werden.

Evaluierung mit Goldstandard-Daten

Option 2

- Eine Menge von Testsätzen wird manuell geparst.
- Die manuell erstellten Parsebäume werden mit der Ausgabe des Parsers für die Testsätze verglichen.

Vorteile

- Es müssen nur einmal Daten annotiert werden.
- Die Evaluierung ist objektiver (keine übersehenen Fehler).

Problem: Kurze und lange Sätze haben dasselbe Gewicht.

Precision

Es werden korrekte Konstituenten statt korrekter Parsebäume gezählt

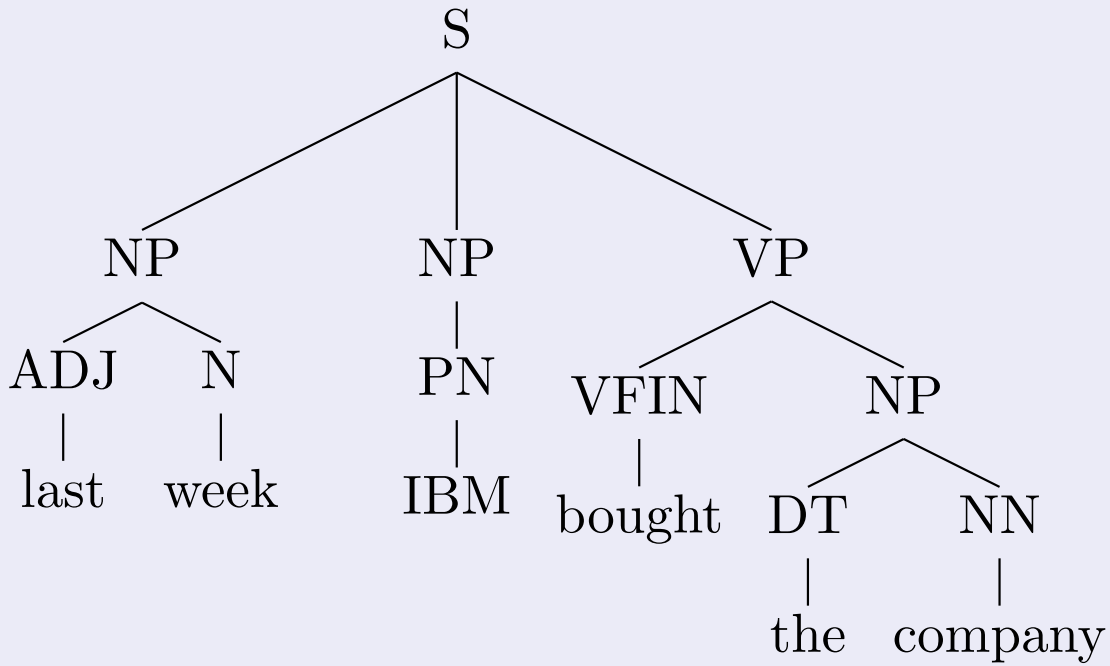
Eine Konstituente ist korrekt, wenn der Goldstandard-Parse eine Konstituente mit derselben Start- und Endposition und derselben Kategorie enthält.

$$\text{Precision} = \frac{\text{Anzahl korrekte Konstituenten}}{\text{Anzahl ausgegebene Konstituenten}}$$

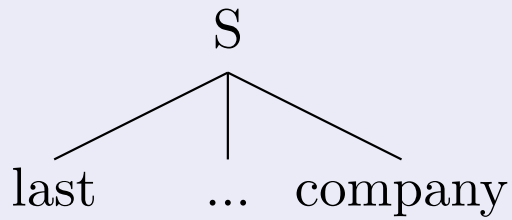
Precision

Problem Der Precision-Wert sagt nicht, wieviele der Goldstandard-Konstituenten der Parser ausgegeben hat.

Goldstandard



Parse



Der rechte Parsebaum expandiert das Startsymbol direkt zu der Folge der Wörter und erzielt eine Precision von 100%, weil er nur eine Konstituente enthält und diese korrekt ist!

Berechnung von Precision und Recall

TP (True Positives): Zahl der ausgegebenen Konstituenten, die korrekt sind

FP (False Positives): Zahl der ausgegebenen Konstituenten, die falsch sind

FN (False Negatives): Zahl der Goldstandard-Konstituenten, die fehlten

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

Mit dem **Recall** messen wir zusätzlich, wieviel Prozent der Goldstandard-Konstituenten im Parse enthalten waren.

Zusammen ergeben Precision und Recall ein genaues Bild der Parsing-Genauigkeit.

F-Score

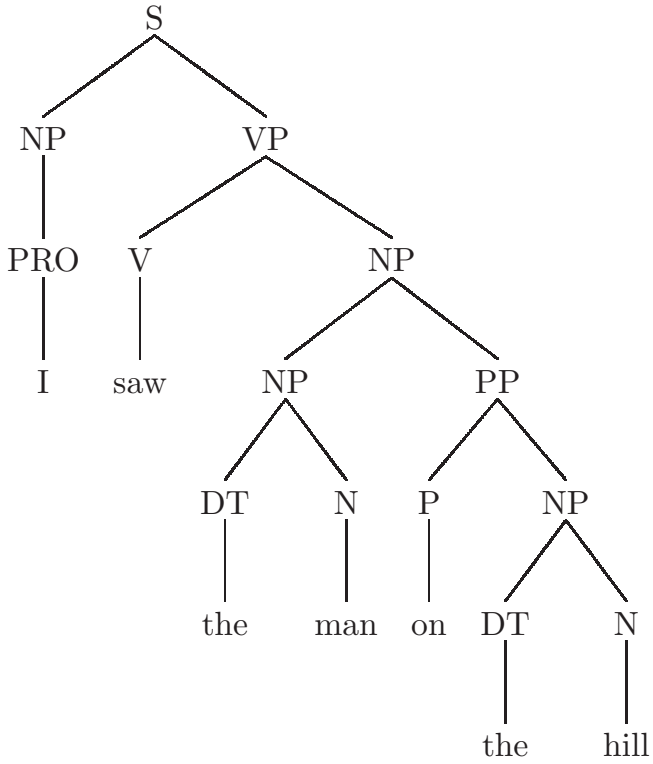
F-Score: harmonisches Mittel aus Precision und Recall

$$F_1 = \frac{1}{\frac{1}{2}(\frac{1}{P} + \frac{1}{R})} = \frac{2}{\frac{R}{PR} + \frac{P}{PR}} = \frac{2PR}{P+R}$$

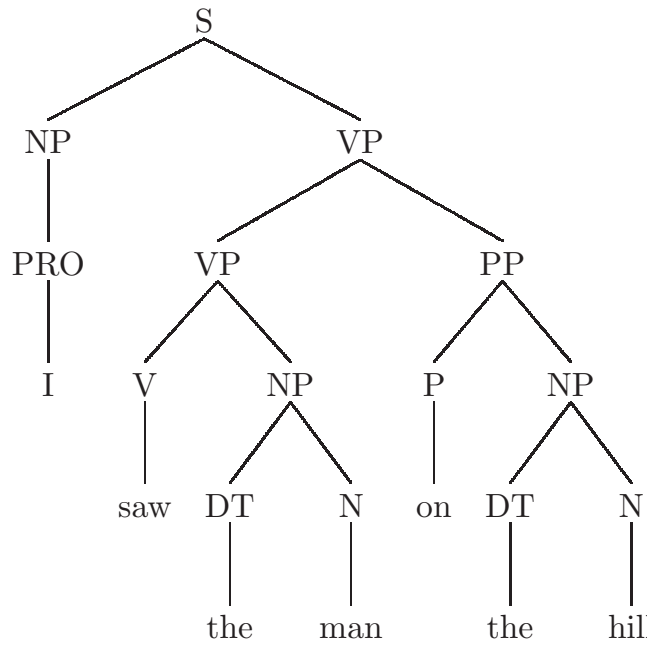
Der gewichtete F-Score gibt Precision β-mal mehr Gewicht als Recall:

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

Beispiel



Goldstandard



Parserausgabe

Goldstandard-Konstituenten:
(S,0,7) (NP,0,1) (VP,1,7) (NP,2,7) (NP,2,4) (PP,4,7) (NP,5,7)

Konstituenten in Parserausgabe:
(S,0,7) (NP,0,1) (VP,1,7) (VP,1,4) (NP,2,4) (PP,4,7) (NP,5,7)

True Positives = 6
False Positives = 1
False Negatives = 1

Precision = TP/(TP+FP) = 6/7
Recall = TP/(TP+FN) = 6/7
F-Score = 2 P R / (P+R) = 2*6/7*6/7/(6/7+6/7)

Trainings- vs. Testdaten

Beim statistischen Parsen braucht man manuell erstellte Parsebäume für

- Training
- Evaluierung

Können für beides dieselben Daten verwendet werden?

Nein!

Nach dem Training sind die Modell-Parameter (d.h. Wahrscheinlichkeiten) an die Trainingsdaten angepasst und funktionieren auf neuen Daten meistens weniger gut.

Wenn Sie auf den Trainingsdaten evaluieren, haben Sie beispielsweise keine Probleme mit unbekannten Wörtern, weil ja alle Wörter im Training gesehen wurden.

Systemoptimierung

Statistische Parser haben oft Parameter, deren Werte irgendwie gesetzt werden müssen.

Gute Werte können gefunden werden, indem der Parser für verschiedene mögliche Werten jeweils evaluiert wird, um die beste Variante zu bestimmen.

Könnten die Trainings- oder Evaluierungsdaten hierfür verwendet werden?

Nein, weder noch!!!

Wegen der Optimierung der Hyperparameter auf den Testdaten wurden solche Parameter ausgewählt, die dort besonders gut funktionieren. Daher würde eine Evaluierung auf denselben Daten die Genauigkeit auf neuen Daten tendenziell überschätzen.

Zur Optimierung der Parameter sollte ein Teil der Trainingsdaten beiseitegelegt werden ($\Rightarrow$ Entwicklungs-Daten)

Kurzzusammenfassung zu den PCFGs

PCFG = CFG + Regelwahrscheinlichkeiten $p(A \rightarrow \alpha)$

Die Wahrscheinlichkeiten aller A-Regeln addieren zu 1:

$$\sum_{\alpha} p(A \rightarrow \alpha) = 1$$

Die Wk. eines Parsebaumes ist das Produkt der Regelwahrscheinlichkeiten:

$$p(T) = p(r_1, ..., r_n) = \prod_{i=1}^{n} p(r_i)$$

Schätzung der Regelwahrscheinlichkeiten mit:

$$p(A \rightarrow \alpha) = \frac{f(A \rightarrow \alpha)}{\sum_{\beta} f(A \rightarrow \beta)}$$

Die Häufigkeiten $f(A \rightarrow \alpha)$ können aus einer **Baumbank** extrahiert oder im E-Schritt des **EM-Algorithmus** geschätzt werden.

Die Menge der Parsebäume eines Satzes wird kompakt durch einen **Parsewald** repräsentiert.

Kurzzusammenfassung zu den PCFGs

Der **Viterbi-Algorithmus** berechnet den besten Parsebaum im Parsewald.

Initialisierung der Wken der Terminalsymbole mit 1:

$$\delta(a) = 1 \quad \text{für jedes Terminalsymbol a}$$

Viterbi-Wk. von Parsewaldregeln:

$$\delta(A \rightarrow X_1 ... X_n) = p(A \rightarrow X_1 ... X_n) \prod_{i=1}^{n} \delta(X_i) \text{ für Parsewald-Regeln}$$

Viterbi-Wk. von Parsewaldknoten:

$$\delta(A) = \max_{\alpha} \delta(A \rightarrow \alpha)$$

Verweis auf die beste Analyse eines Knotens:

$$\psi(A) = \arg\max_{\alpha} \delta(A \rightarrow \alpha)$$

Kurzzusammenfassung zu den PCFGs

Extraktion des besten Parsebaumes mit der rekursiven Funktion build_tree(ψ, S):

```
build_tree(ψ, A)
    subtree = [A]
    for X in ψ(A) do:
        if X is terminal
            subtree.append(X)
        else
            subtree.append(build_tree(ψ, X))
    return subtree
```

Kurzzusammenfassung zu den PCFGs

Inside-Outside-Algorithmus zur Berechnung **erwarteter Regelhäufigkeiten**

Inside-Wahrscheinlichkeiten:

$$\begin{aligned}
\alpha(a) &= 1 \quad \text{für Terminalsymbol a} \\
\alpha(A \rightarrow X_1 \ldots X_n) &= p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \alpha(X_i) \text{ für Parsewaldregeln} \\
\alpha(A) &= \sum_{A \rightarrow \gamma} \alpha(A \rightarrow \gamma) \quad \text{für Nichtterminale A}
\end{aligned}$$

Outside-Wahrscheinlichkeiten:

$$\begin{aligned}
\beta(S) &= 1 \quad \text{für Startsymbol S} \\
\beta(B \rightarrow X_1 \ldots X_m \underline{A} X_{m+1} \ldots X_n) &= \beta(B) p(B \rightarrow X_1 \ldots X_m A X_{m+1} \ldots X_n) \prod_{i=1}^{n} \alpha(X_i) \\
\beta(A) &= \sum_{B \rightarrow \gamma A \delta} \beta(B \rightarrow \gamma \underline{A} \delta)
\end{aligned}$$

Kurzzusammenfassung zu den PCFGs

Wahrscheinlichkeit einer Parsewald-Regel gegeben den Satz

$$\gamma(A \to \delta) = \frac{\beta(A)\,\alpha(A \to \delta)}{\alpha(S)}$$

Falls $\gamma(NP_3 \to NP_4\ PP_5) = 0.3$, so wird 0.3 zur erwarteten Häufigkeit der Regel NP $\to$ NP PP hinzuaddiert.

Die γ-Werte werden so über die Parsewälder aller Sätze aufsummiert, um die erwarteten Regelhäufigkeiten zu erhalten.

Im M-Schritt werden die Wahrscheinlichkeiten neuberechnet.

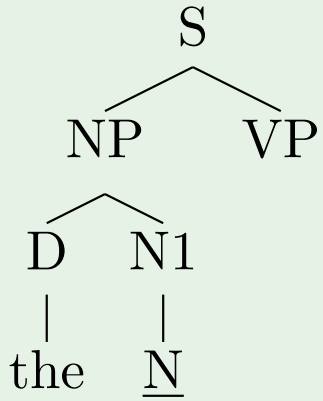
Evaluierung mit Precision, Recall und F-Score

Berkeley-Parser

Unabhängigkeitsannahmen der PCFGs

Regelfolge und partielle Parsebäume

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{NP} &\rightarrow \text{D N1} \\
\text{D} &\rightarrow \text{the} \\
\text{N1} &\rightarrow \text{N}
\end{aligned}
$$



Die Wahrscheinlichkeit der Regel N → bell hängt in diesem Kontext nur von der Kategorie N der expandierten Konstituente ab.

Schwachpunkte einfacher Baumbank-Grammatiken

Die Unabhängigkeitsannahmen sind zu stark:

- NPs in Subjektposition unterscheiden sich von NPs in Objektposition.
- Argument-PPs unterscheiden sich von Adjunkt-PPs.
- Adverbien, die Verben modifizieren, unterscheiden sich von Adverbien, die Adjektive modifizieren.
- Transitive Verben unterscheiden sich von intransitiven Verben.
- Verben, die PP-Argumente nehmen, unterscheiden sich in den selegierten Präpositionen.
- Nichtlokale Abhängigkeiten wie WH-Bewegungen werden nicht angemessen behandelt..
- Verben haben unterschiedliche Präferenzen bei ihren Subjekt- und Objekt-NPs.
- etc.

Baumbank-Annotation

Idee: Nützliche Information wird automatisch zu den Baumbank-Annotationen hinzugefügt (mit einem Computerprogramm).

$\Rightarrow$ Die Baumbank-Kategorien werden verfeinert.

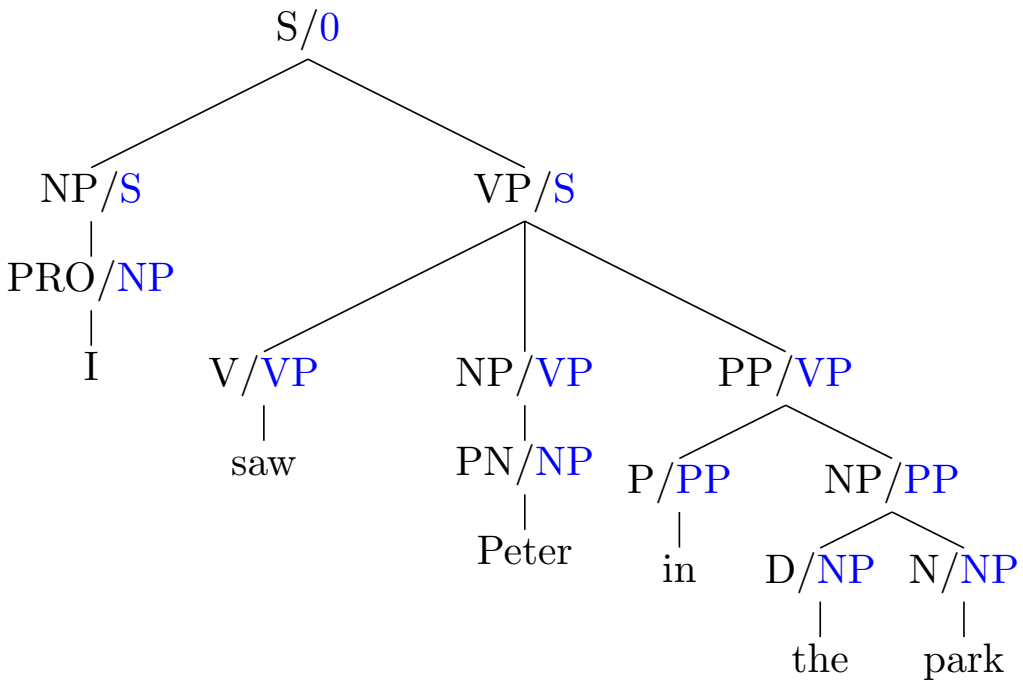
$\Rightarrow$ Kontextinformation wird in den Kategorien repräsentiert.

$\Rightarrow$ Die Unabhängigkeitsannahmen werden abgeschwächt.

$\Rightarrow$ PCFGs, die aus der verfeinerten Baumbank extrahiert werden, können besser zwischen verschiedenen Analysen desambiguieren.

Elternannotation (Mark Johnson)

Jeder Parseknoten wird zusätzlich mit der Kategorie des Elternknotens beschriftet.



Diese Annotation unterscheidet zwischen

- Subjekten und Objekten
- verschiedenen VP-Typen
- verschiedenen Arten von Adjunktanbindungen
- prädikativen und attributiven Adjektiven
- ...

Beispielregeln mit Annotationen

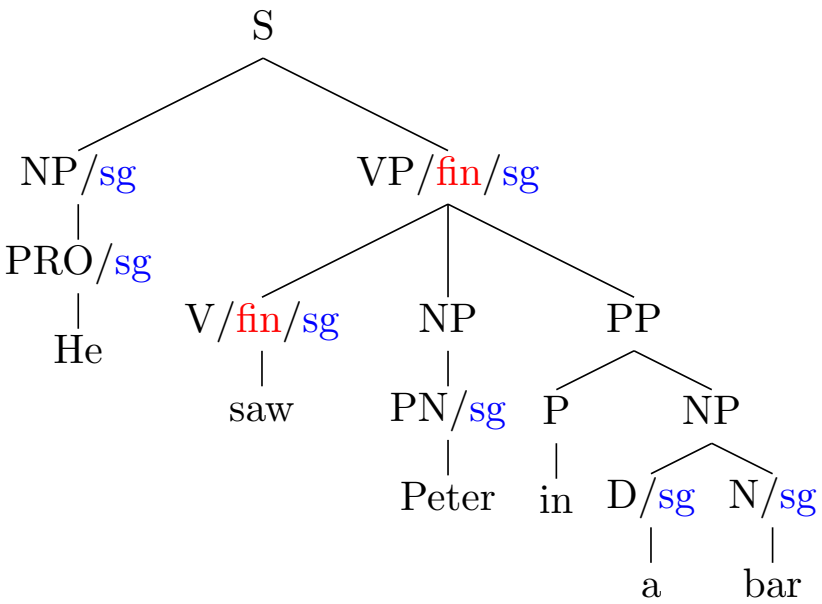
(aus der Penn-Treebank)

- NP/S → PRO/NP ist wahrscheinlicher als die Originalregel NP → PRO
- VP/VP → VB/VP NP/VP ist wahrscheinlicher als VP → VB NP
- VBG/PP → *including* ist wahrscheinlicher als VBG → *including*

Annotation mit morphosyntaktischer Information

VP: finites Verb, Infinitiv, Gerund, Partizip

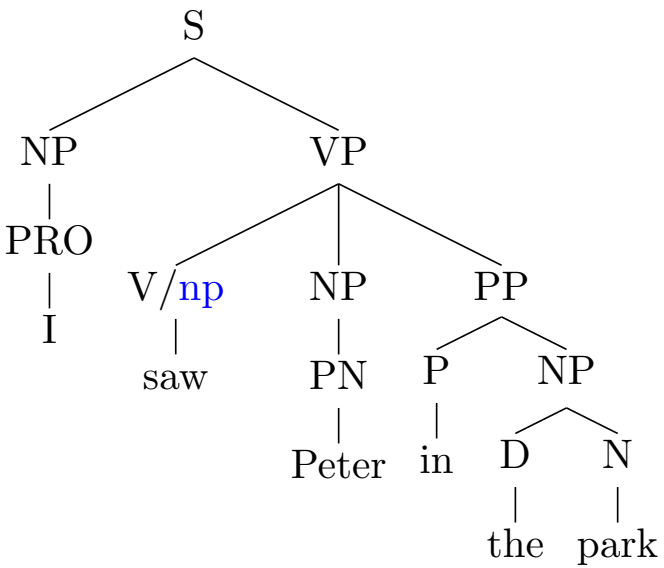
N: Numerus (Genus, Kasus)



- erfordert lexikalische Information
- unterscheidet zwischen verschiedenen Arten von Verben und VPs
- prüft Agreement innerhalb von NPs (wichtig für Deutsch)
- prüft Subjekt-Verb-Agreement im Numerus

Annotation mit Argument-Information

Kodierung der Argumentinformation: np = NP + PP



Trainingsdaten müssen enthalten

- Argument/Adjunkt-Unterscheidungen
- Information über Argument-"Spuren"
  (mehr dazu auf der nächsten Folie)

Vor- und Nachteile

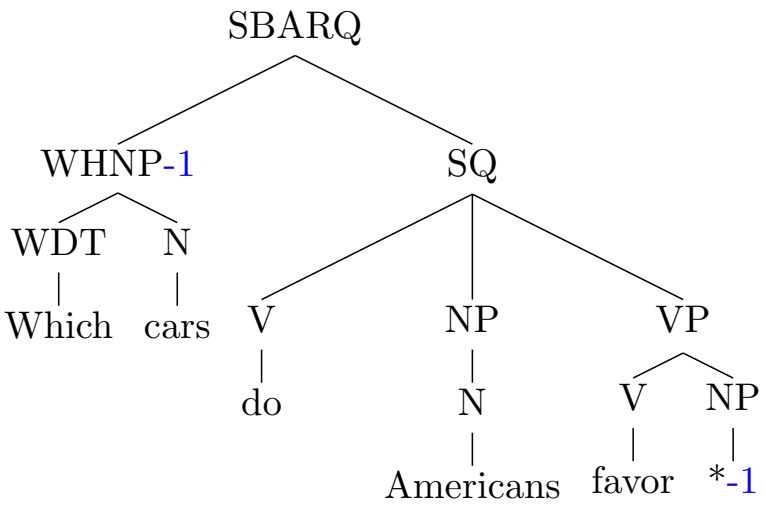
+ erlaubt eine Unterscheidung zwischen Argumenten und Adjunkten
+ löst manche PP-Anbindungsprobleme
– mehr Verbeinträge im Lexikon
– dadurch Sparse-Data-Probleme

Umgang mit Konstituentenbewegungen

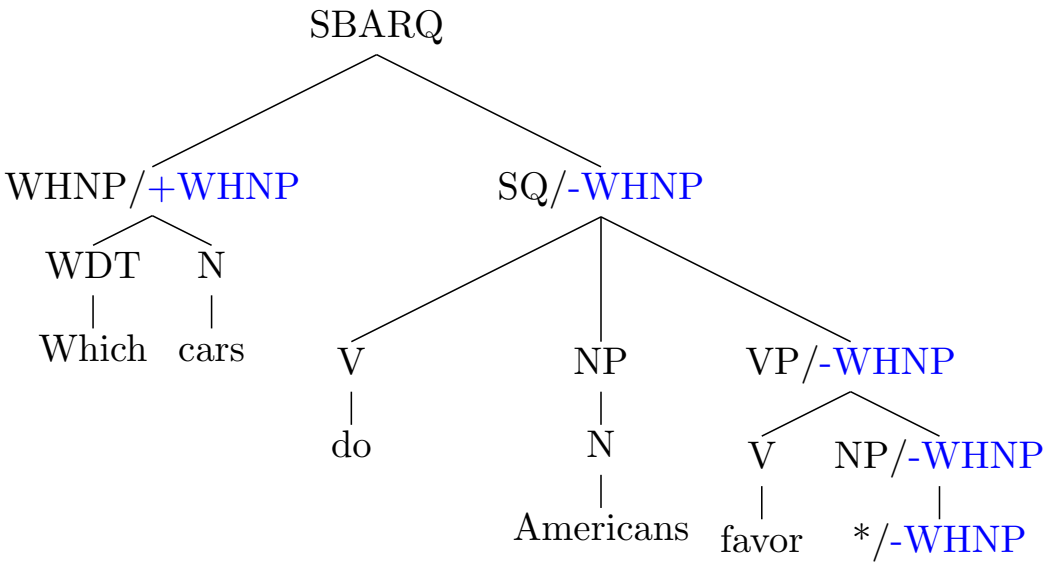
Alle Knoten auf dem Weg von der Spur zu ihrem Füller werden mit einem Spurmerkmal annotiert.

Koreferenz-Indizes

(verwendet in der Penn Treebank)



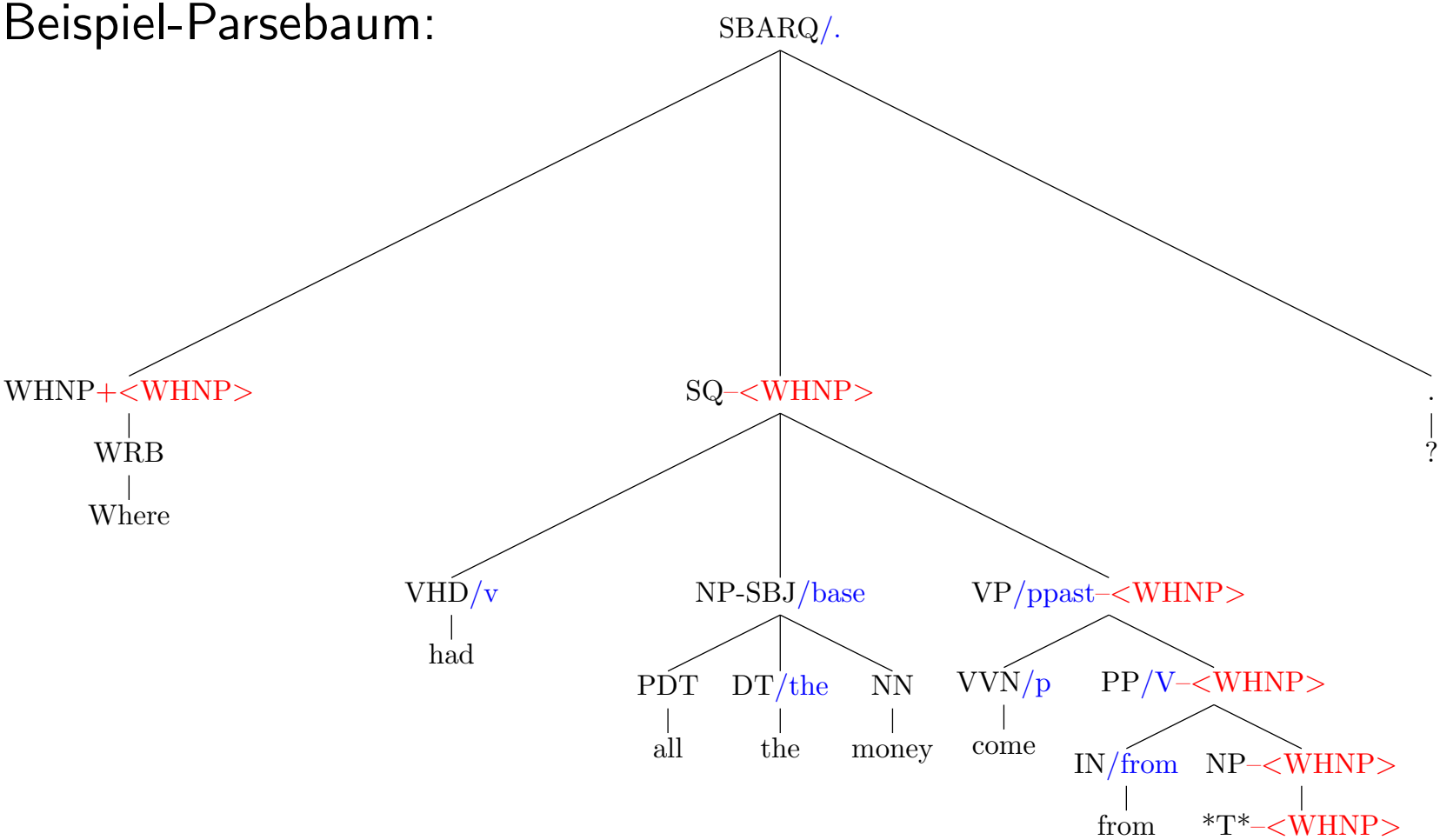
Annotation mit Spurmerkmalen



Die Spurmerkmale stellen sicher, dass für jede Spur ein Füller vorhanden ist und umgekehrt.

Mehrfache Annotationen

Beispiel-Parsebaum:



Beispielregel:

SQ-<WHNP> → VHD/v NP-SBJ/base VP/ppast-<WHNP>

Mehrfache Annotationen

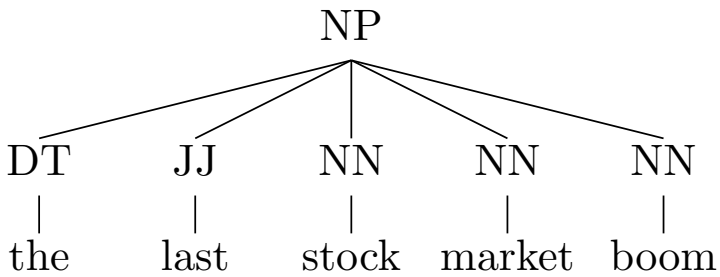
Zusätzliche Annotation erhöhen die Zahl der Nichtterminale und Regeln.

- ⇒ Sparse Data Probleme:

  Viele mögliche Regeln tauchen nicht in den Trainingsdaten auf.
- ⇒ Welche Menge von Annotationen optimal ist, hängt von der Baumbankgröße und anderen Faktoren ab.

Probleme mit flachen Baumbankannotationen

- Viele Baumbanken verwenden flache Strukturen.



- Die extrahierten Grammatiken enthalten viele Regeln mit langen rechten Seiten, die nur einmal auftauchen.

$$\text{NP} \rightarrow \text{DT JJ NN NN NN}$$

- Andere ähnliche Regeln fehlen, werden aber für das Parsen mancher Sätze benötigt.

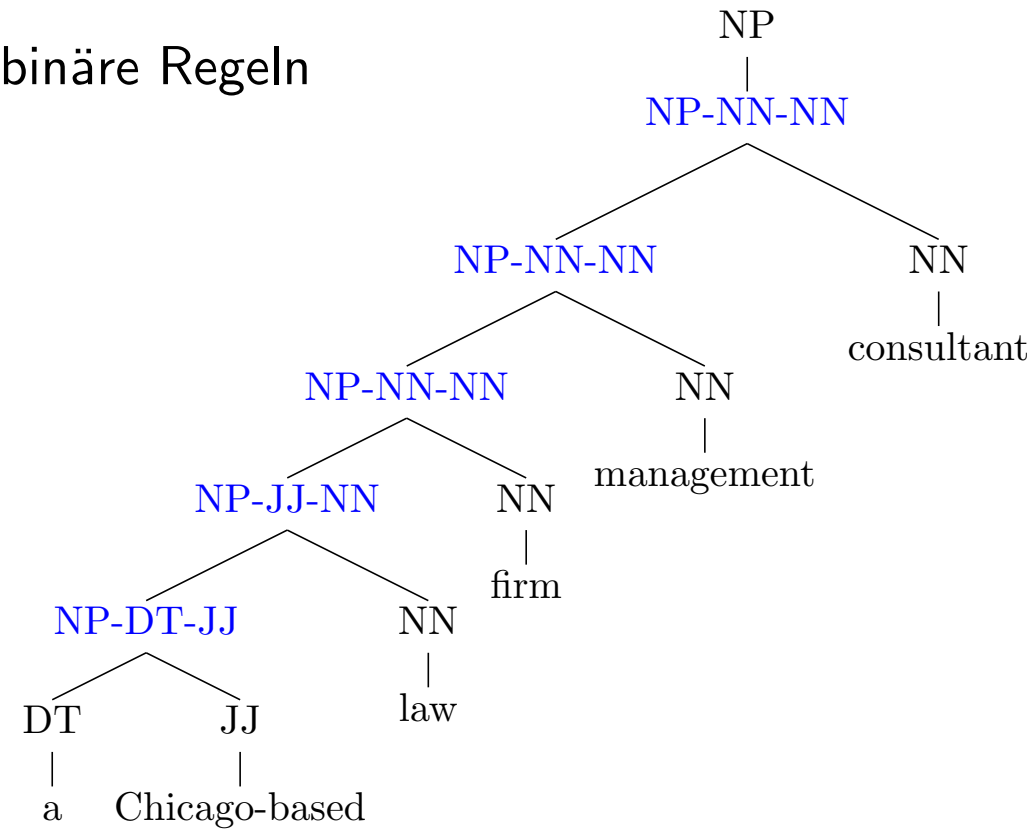
$$\text{NP} \rightarrow \text{DT JJ NN NN NN NN}$$

Markowisierung

- Aufspaltung von langen Regeln in binäre Regeln
- Neue Hilfskategorien mit z.B.
  - der Elternkategorie und
  - den Kategorien der beiden letzten Tochterknoten
- Das Entfernen der Hilfsknoten liefert wieder den Originalparse

$\Rightarrow$ Reduktion der Grammatikgröße

$\Rightarrow$ Verbesserung ihrer Abdeckung



Die gezeigte Markowisierung ist äquivalent dazu, dass wir die rechten Seiten (und Wahrscheinlichkeiten) von bspw. NP-Regeln mit einem Markow-Modell 2. Ordnung erzeugen.

Automatisch gelernte Merkmale

Nachteile der Merkmalsannotation per Programm

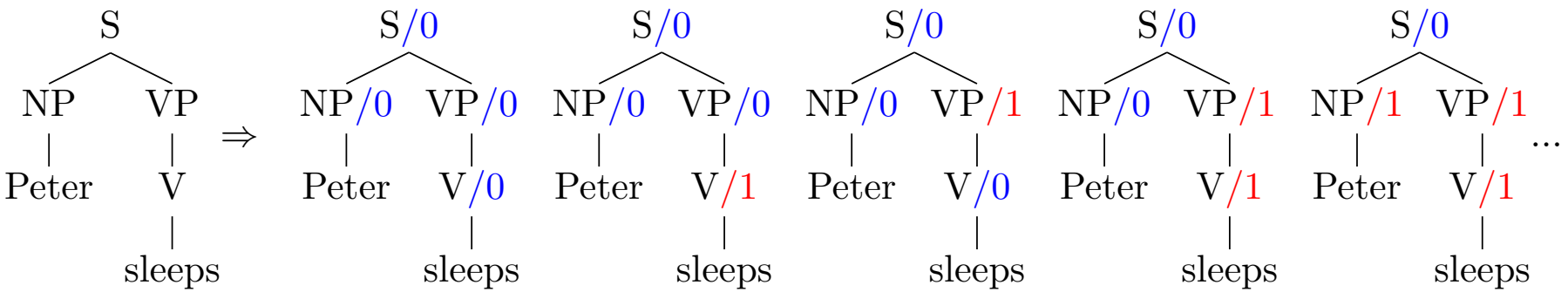
- komplexer ,,Trial and Error"-Prozess
- aufwändige Programmierarbeit
- muss für jede Baumbank wiederholt werden

Könnten gute Merkmale automatisch gelernt werden?

Synthetische Merkmale

Grundidee (von Petrov/Klein)

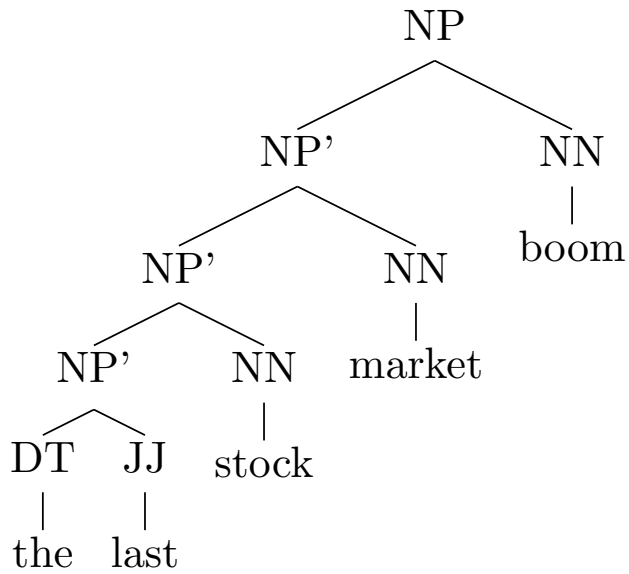
- Alle Kategorien werden durch ein synthetisches Merkmal mit den Werten 0 und 1 aufgespalten.
- Jeder Parse der Baumbank kann von der neuen Grammatik auf viele unterschiedliche Arten generiert werden.
- Durch EM-Training wird die neue Grammatik an die Baumbank angepasst.



Die modifizierte Grammatik liefert für den alten Parsebaum $2^4 = 16$ neue Parsebäume.

Vorverarbeitung

1. Binarisierung der Parsebäume
2. Extraktion einer Grammatik mit Häufigkeiten



$$\begin{array}{lll} \text{NP} \rightarrow \text{NP' NN} & 1 \\ \text{NP'} \rightarrow \text{NP' NN} & 2 \\ \text{NP'} \rightarrow \text{DT JJ} & 1 \end{array}$$

3. Aufspaltung jeder Kategorie in 2 neue Kategorien
   Uniforme Verteilung der Häufigkeiten über die neuen Regeln
   (mit kleinen Abweichungen, um die Symmetrie zu brechen.
   Sonst bleibt das EM-Training im Anfangszustand stecken.)

$$\begin{array}{lll} \ldots & & \\ \text{NP'}/0 \rightarrow \text{NP'}/0\ \text{NN}/0 & 0.24 & \\ \text{NP'}/0 \rightarrow \text{NP'}/0\ \text{NN}/1 & 0.26 & \\ \text{NP'}/0 \rightarrow \text{NP'}/1\ \text{NN}/0 & 0.25 & \\ \ldots & & \end{array}$$

Hier wird die Häufigkeit von 2 der Originalregel auf 8 neue Regeln verteilt.

EM-Training

1. Aus den Regelhäufigkeiten werden Maximum-Likelihood-Schätzungen der Regelwahrscheinlichkeiten berechnet (M-Schritt).

$$p(A \to \alpha) = \frac{f(A \to \alpha)}{\sum_{A \to \beta} f(A \to \beta)}$$

2. Für jeden Parsebaum der Baumbank wird der Parsewald mit Analysen entsprechend der verfeinerten Grammatik berechnet.
3. Mit dem Inside-Outside-Algorithmus werden erwartete Häufigkeiten für die Parsewaldregeln berechnet (E-Schritt).
4. Die erwarteten Regelhäufigkeiten werden über alle Sätze summiert.
5. Weiter mit 1)

EM-Training

- Wegen der leicht zufälligen Initialisierung der Wahrscheinlichkeiten unterscheiden sich die synthetisch erzeugten Unterkategorien etwas.
- Diese Unterschiede verstärken sich während des EM-Trainings.

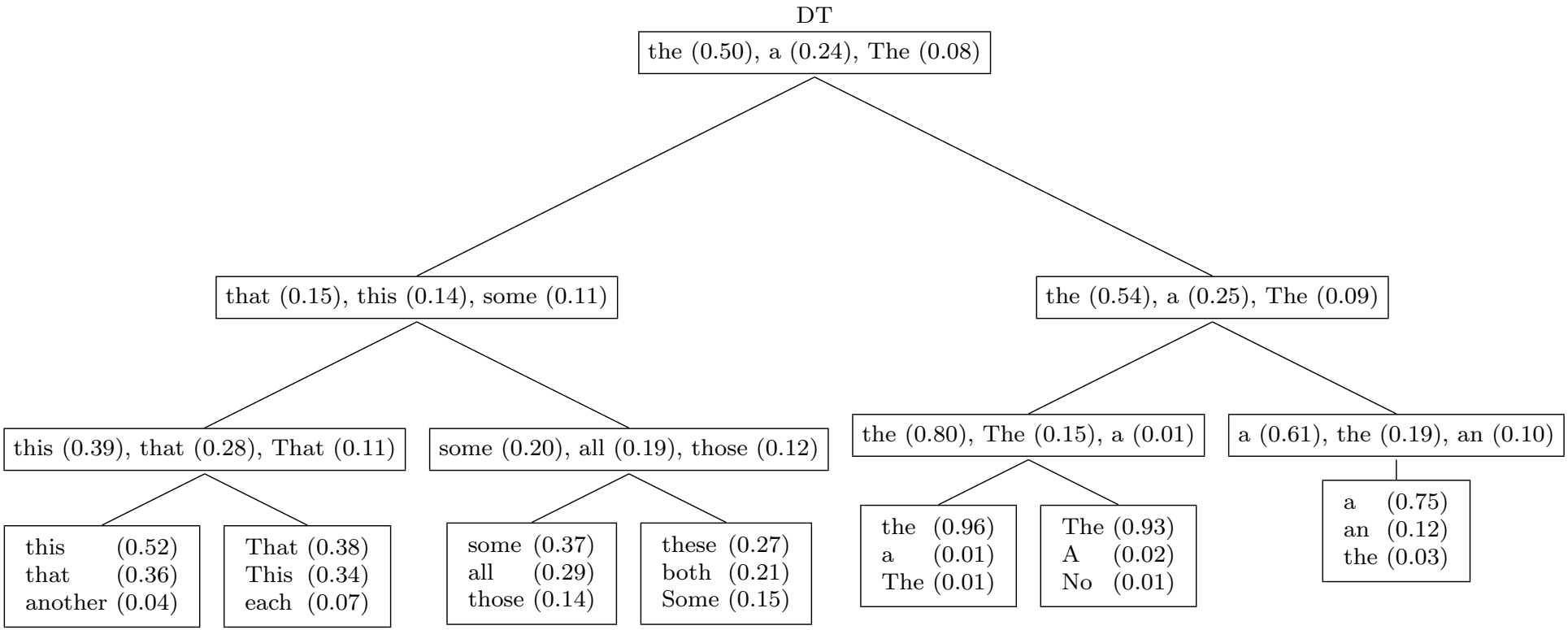
$\Rightarrow$ Die Unterkategorien spezialisieren sich.

Beispiel: wahrscheinlichste Expansionen der DT-(Unter-)Kategorien

DT	the (0.50)	a (0.24)	The (0.08)	...
DT/0	that (0.15)	this (0.14)	some (0.11)	...
DT/1	the (0.54)	a (0.25)	The (0.09)	...

Rekursive Verfeinerung

Rekursives Aufspalten der Kategorien gefolgt von mehreren EM-Iterationen verfeinert die Kategorien immer mehr.



Es werden jeweils nur die 3 wahrscheinlichsten Expansionen gezeigt.

Vereinigung

- Ab einem bestimmten Punkt ist eine weitere Aufspaltung der Kategorien nicht mehr vorteilhaft, weil sie zu Sparse-Data-Problemen führt.

  Die Penn-Treebank verwendet bspw. eine eigene Kategorie "," für Kommas, deren Aufspaltung nichts nützt.

⇒ Unterkategorien, die sich zu ähnlich sind, werden daher nach dem EM-Training wieder zusammengefasst.

Strategie:

- Nach jedem EM-Training und vor dem Aufspalten der Kategorien, werden 50% der vorherigen Aufspaltungen rückgängig gemacht.
- Für jede Aufspaltung wird berechnet, wie stark sich die Wahrscheinlichkeit der Trainingsdaten verringert, wenn die Aufspaltung rückgängig gemacht wird. (Details im Originalartikel)
- Die Aufspaltungen mit der kleinsten Differenz werden rückgängig gemacht.

Experimentelle Ergebnisse

- "Berkeley"-Parser von Slav Petrov und Dan Klein
- Der Parser liefert für viele verschiedene Sprachen und Baumbanken gute Ergebnisse
- Deutsch (Precision=80.1%, Recall=80.1%)
- Englisch (Precision=90.2%, Recall=89.9%)
- Chinesisch (Precision=84.8%, Recall=81.9%)

Aber: Neuere Parser auf Basis neuronaler Netze sind noch besser.

Kurzzusammenfassung: Berkeley-Parser

1. gegeben: eine **Baumbank**
2. **Binarisierung** der Parsebäume durch Einfügen von Hilfsknoten
3. Extraktion einer Grammatik mit Regelhäufigkeiten
4. für mehrere Iterationen (Split&Merge-Schritte)
   1. Aufspaltung jeder Kategorie X in zwei neue Kategorien X1 und X2 (Split-Schritt)
   2. annähernd uniforme Verteilung der Häufigkeit einer Regel auf die neuen Regeln
   3. für mehrere Iterationen (EM-Training)
      1. Schätzung der Regelwahrscheinlichkeiten aus den Regelhäufigkeiten (M-Schritt)
      2. Für jeden Parsebaum der Baumbank wird der Parsewald mit Analysen entsprechend der verfeinerten Grammatik berechnet.
      3. Mit dem Inside-Outside-Algorithmus werden erwartete Häufigkeiten für die Parsewaldregeln berechnet (E-Schritt).
      4. Die erwarteten Regelhäufigkeiten werden über alle Sätze summiert.
   4. Die schlechtere Hälfte der Aufspaltungen wird rückgängig gemacht (Merge-Schritt).

Diskriminative Modelle

Generative Modelle

Alle bisher betrachteten Modelle (Markowodell, Naïve-Bayes-Modell, HMM, PCFG) sind generative Modelle.

- Sie definieren eine gemeinsame Wahrscheinlichkeit von Sätzen und Annotationen bspw. $p(w_1^n, t_1^n)$
- Sie zerlegen die Wahrscheinlichkeit (z.B. eines getaggten Satzes) in ein Produkt von Einzel-Wahrscheinlichkeiten $\prod_{i=1}^{n+1} p(t_i|t_{i-1})p(w_i|t_i)$
- Sie modellieren einen Generierungsprozess, der die annotierten Daten Stück für Stück generiert und in jedem Schritt zwischen mehreren möglichen Aktionen auswählen kann.
- Jede mögliche Aktion (z.B. die Wahl des nächsten Tags/des nächsten Wortes) besitzt eine Wahrscheinlichkeit.
- Die Wahrscheinlichkeiten der Aktionen werden multipliziert.

Nachteile generativer Modelle

Zusätzliche Information in ein gegebenes generatives Modell zu integrieren ist schwierig und manchmal unmöglich.

Beispiel: HMM-Tagger

- kombiniert $p(t_i|t_{i-1})$ und $p(w_i|t_i)$
- Zur Integration suffix-basierter Wort-Tag-Wahrscheinlichkeiten wurde das Modell geändert und $p(w_i|t_i)$ durch $p(t_i|w_i)p(w_i)/p(t_i)$ ersetzt
- Wie könnte zusätzlich noch eine Abhängigkeit des aktuellen Tags vom vorhergehenden Wort $p(t_i|w_{i-1})$ integriert werden?

  Problem: t_i wurde schon von $p(t_i|t_{i-1})$ "generiert" und darf nicht noch einmal generiert werden. Wir könnten höchstens $p(t_i|w_{i-1})$ und $p(t_i|t_{i-1})$ interpolieren und so das Modell weiter verkomplizieren.

Diskriminative Modelle

Diskriminative Modelle definieren eine bedingte Wahrscheinlichkeit $p(c|d)$ für die Klassen c gegeben ein Objekt d.

Beispiel: Email-Klassifikation

Email: Antworten Sie auf diese Email und gewinnen Sie 1000 Euro!!!
Klassen: Spam, kein Spam (auch "Ham" genannt)
gesucht:
p(Spam | Antworten Sie auf diese Email und gewinnen Sie 1000 Euro!!!)

Beim Naïve Bayes-Modell wird jetzt das Bayes'sche Theorem angewendet, um die bedingten Wahrscheinlichkeiten umzudrehen. Dann werden Unabhängigkeits-Annahmen gemacht, um das Modell zu vereinfachen.

Diskriminative Modelle definieren die bedingte Wahrscheinlichkeit direkt.

Sie nutzen als Information beliebig definierte Merkmale der klassifizierten Objekte.

Diskriminative Modelle

Grundidee:

- Manuell erstellte Merkmalsfunktionen extrahieren relevante Information aus dem Dokument
  z.B. das Vorkommen eines Wortes wie “Lotterie”.
- Der Wert einer solchen Merkmalsfunktion ist eine Zahl
  z.B. die Häufigkeit des Wortes “Lotterie”
- Jeder Merkmalswert wird mit einem Gewicht multipliziert, das ausdrückt, ob das Merkmal auf eine bestimmte Klasse hindeutet oder dieser widerspricht.
- Die gewichteten Merkmalswerte werden zu einer Gesamtbewertung (Score) aufsummiert.
- Die Scores werden in Wahrscheinlichkeiten transformiert.

Merkmale

Arten von Merkmalsfunktionen $f_1(d), ..., f_M(d)$.

- numerische Merkmale

  $f_1(d) =$ Häufigkeit des Wortes "win" in d

- binäre Merkmale

$$f_2(d) = \begin{cases} 1 & \text{falls Länge von d} in [20, ..., 40) \\ 0 & sonst \end{cases}$$

Beispiel

Wir wollen zwischen Spam-Emails (positive Klasse) und normalen Emails (negative Klasse) unterscheiden.

Es handelt sich also um ein 2-Klassen-Problem.

Email-Text:

GRATULATION!!! Sie haben bei unserer Lotterie gewonnen! Klicken Sie hier, um Ihren Preis anzufordern.

extrahierte Merkmale (nach Stoppwortentfernung und Lemmatisierung)

Merkmal	Wert
Gratulation	1
Lotterie	1
gewinnen	1
klicken	1
Preis	1
anfordern	1

Gewichte

Für jedes Merkmal $f_i(d)$ gibt es ein entsprechendes Gewicht θ_i.

- $\theta_i > 0$:
  Das Merkmal $f_i(d)$ deutet auf die positive Klasse (Spam) hin.
- $\theta_i < 0$:
  Das Merkmal $f_i(d)$ deutet auf die negative Klasse (Ham) hin.

Die Gewichte aller Merkmale, deren Wert von 0 verschieden ist, fließen in die Klassifikation ein.

Die Gewichte sind reelle Zahlen und entsprechen in ihrer Funktion den logarithmierten Wahrscheinlichkeiten eines Naïve-Bayes-Modelles. Die optimalen Werte der Gewichte werden im Training gelernt. Sie werden aber nicht aus Häufigkeiten geschätzt, sondern anders gelernt. Zum Vergleich mit Naïve-Bayes-Modellen siehe auch Folie 266.

Lineare Klassifikatoren

Ein linearer Klassifikator multipliziert die Merkmalswerte $f_i(d)$ und Gewichte θ_i und summiert sie auf:

$$score(d) = \sum_i \theta_i f_i(d) = \theta \cdot f(d)$$

θ und $f(d)$ sind hier Vektoren und $\cdot$ ist das Vektorprodukt.

Der Ergebniswert (Score) kann direkt zur Klassifikation verwendet werden:

$$class(d) = \begin{cases} \text{positiv} & \text{falls } score(d) > 0 \\ \text{negativ} & \text{sonst} \end{cases}$$

Diese Methode wendet der Perzeptron-Algorithmus an, den wir als Nächstes behandeln.

Später werden wir sehen, wie man den Score in eine Wahrscheinlichkeit transformiert ($\Rightarrow$ Maximum-Entropy-Modelle).

Beispiel

GRATULATION!!! Sie haben bei unserer Lotterie gewonnen! Klicken Sie hier, um Ihren Preis anzufordern.

extrahierte Merkmale (nach Stoppwortentfernung und Lemmatisierung)

Merkmal	Wert	Gewicht
Gratulation	1	1.2
Lotterie	1	0.9
gewinnen	1	0.7
klicken	1	1.3
anfordern	1	-0.2
Preis	1	0.5

Summe der gewichteten Merkmale = 1 * 1.2 + 1 * 0.9 + ... = 4.4

Ergebnisklasse: Spam (da $4.4 > 0$)

Merkmalsvektoren

G	M	M1	M2
1.2	0	1:2	gewinnen:2
0.7	2	2:1	Baum:1
−1.5	1	5:1	anfordern:1
9.2	0		
−3.1	0		
−0.2	1		
2.9	0		
1.8	0		

- Wir verwenden in der Regel **tausende** Merkmale.
- Die meisten Merkmale liefern jeweils 0-Werte.
- Merkmalsvektoren und Gewichtsvektoren enthalten für **jedes** Merkmal einen Wert.
- Die Multiplikation von Gewichtsvektor (G) und Merkmalsvektor (M) umfasst daher viele **nutzlose** Multiplikationen und Additionen mit 0.
- Wir können Merkmale mit 0-Werten beim Vektorprodukt **ignorieren**.
- Wir **speichern** daher nur die von 0 verschiedenen Werte des Merkmalsvektors (M1) und nicht den ganzen Vektor.
- Zur **Identifizierung** des Merkmales müssen wir mit dem Merkmalswert auch den **Index** des Merkmales speichern.
- Statt des Indexes können wir auch einen verständlicheren **Merkmalsnamen** (M2) verwenden.

Mehrklassen-Probleme

Wenn es mehr als 2 mögliche Klassen gibt, braucht man klassenspezifische Merkmale (oder zumindest klassenspezifische Gewichte).

Beispiel: Topic-Klassifikation (Politik, Wirtschaft, Sport)

Text: Aktien der Telekom im Minus

Klassifikation durch

- Berechnung des Merkmalsvektors für jede Klasse

"Aktie Politik"	1	"Aktie Wirtschaft"	1	"Aktie Sport"	1
"Telekom Politik"	1	"Telekom Wirtschaft"	1	"Telekom Sport"	1
"Minus Politik"	1	"Minus Wirtschaft"	1	"Minus Sport"	1

- Multiplikation der Merkmalswerte mit den Gewichten und Summation
- Auswahl der Klasse mit der höchsten Bewertung

$$\hat{c} = \arg\max_c \sum_i \theta_i f_i(c, d) = \arg\max_c \theta \cdot f(c, d)$$

Wie werden die Gewichte θ_i gelernt?

Vergleich mit Naive Bayes-Modellen

Ein Naive Bayes-Modell definiert die logarithmierte gemeinsame Wahrscheinlichkeit für die Klasse c und die Wortfolge w_1^n wie folgt:

$$\log p(c, w_1^n) = \log p(c) + \sum_{i=1}^{n} \log p(w_i|c)$$

Wenn V das Wort-Inventar ist, und $f_w(w_1^n)$ die Häufigkeit des Wortes w in der Wortfolge w_1^n angibt, dann können wir auch schreiben:

$$\begin{aligned}
\log p(c, w_1^n) &= \log p(c) + \sum_{w \in V} \log p(w|c)^{f_w(w_1^n)} \\
&= \log p(c) + \sum_{w \in V} [\log p(w|c)]\, f_w(w_1^n) \\
&= \log p(c) + \sum_{c'} \sum_{w \in V} [\log p(w|c')]\, [f_w(w_1^n) \mathbf{1}_{c=c'}]
\end{aligned}$$

Wenn wir den (Bias-)Term $\log p(c)$ ignorieren, entspricht das dem Score eines linearen Klassifikators, dessen Merkmalsfunktionen $f_i(c, w_1^n)$ jeweils die Häufigkeit eines bestimmten Wortes w angeben, falls die Argumentklasse c eine bestimmte Klasse c' ist, und sonst 0:

$$score(c, w_1^n) = \sum_i \theta_i f_i(c, w_1^n)$$

Perzeptron-Algorithmus

Perzeptron-Algorithmus

- 1957 von Frank Rosenblatt erfunden
- dient zum Training linearer Klassifikatoren

$$\arg\max_y \theta \cdot f(x, y)$$

x ist ein Beispiel und y ist eine Klasse (Analyse).
θ ist der Gewichtsvektor.
$f(x, y)$ ist der Merkmalsvektor.
$\theta \cdot f(x, y)$ ist die Bewertung der Klasse y.

- Das Training ist sehr einfach und effizient
- und funktioniert in der Praxis recht gut.

In der CL muss oft eine **Struktur** (Wortartfolge, Parsebaum) statt einer einzelnen **Klasse** (wie bei der Textklassifikation) ausgegeben werden.

Im Englischen spricht man von *Structured Prediction*.

Grundidee

- Wähle ein Trainingsbeispiel (x, y)
- Wende den Klassifikator darauf an:

$$z = \arg\max_{y'} \theta \cdot f(x, y')$$

- Wenn x falsch klassifiziert wird (d.h. $z \neq y$)
  - Modifiziere θ so, dass y höher und z niedriger bewertet wird.

$$\theta \leftarrow \theta + f(x, y) - f(x, z)$$

- Wiederhole

Wenn die Trainingsdaten linear separierbar sind, findet der Algorithmus in endlicher Zeit einen Gewichtsvektor, der alle Daten richtig klassifiziert.

Beispiel

Wortart-Annotation

Wörter	the	man	bit	the	dog
korrekte Tags	DT	NN	VBD	DT	NN
vorhergesagte Tags	DT	NN	NN	DT	NN

Angenommen es gibt Merkmale für

- Tag-Tag-Paare DT-NN, NN-VBD, ...
- Wort-Tag-Paare DT:the, NN:man, ...

Um 1 erhöht werden die Gewichte für: NN-VBD, VBD-DT, VBD:bit

Um 1 verringert werden die Gewichte für: NN-NN, NN-DT, NN:bit

Alle übrigen Merkmalsgewichte bleiben unverändert.

Anmerkung: Für die Berechnung der besten Tagfolge kann der Viterbi-Algorithmus verwenden werden (siehe Folie 296).

Beispiel

Wortart-Annotation

Wörter	the	man	bit	the	dog
korrekte Tags	DT	NN	VBD	DT	NN
vorhergesagte Tags	DT	NN	NN	DT	NN

Angenommen es gibt Merkmale für

- Tag-Tag-Paare DT-NN, NN-VBD, ...
- Wort-Tag-Paare DT:the, NN:man, ...

Um 1 erhöht werden die Gewichte für: NN-VBD, VBD-DT, VBD:bit

Um 1 verringert werden die Gewichte für: NN-NN, NN-DT, NN:bit

Alle übrigen Merkmalsgewichte bleiben unverändert.

Anmerkung: Für die Berechnung der besten Tagfolge kann der Viterbi-Algorithmus verwenden werden (siehe Folie 296).

Perzeptron-Lernalgorithmus

Eingabe: Trainingsdaten (x_i, y_i) für $i = 1...n$

Initialisierung: $\theta = \vec{0}$

Algorithmus:

for $t = 1...T$ // für T Iterationen
    Wähle zufällig ein i in $1...n$
    $z \leftarrow \arg\max_y \theta \cdot f(x_i, y)$ // klassifiziere
    if $z \neq y_i$ // Fehlklassifikation?
        $\theta \leftarrow \theta + f(x_i, y_i) - f(x_i, z)$ // Gewichtsanpassung
return θ // Rückgabe des Gewichtsvektors

Stoppkriterium

- In der Theorie lässt man den Algorithmus bis zur Konvergenz laufen.
- Aber, das Training konvergiert nicht, wenn die Daten nicht linear separierbar sind.
- Man könnte sehr lange trainieren, bis die (relative) Gewichtsvektoränderung sehr klein wird.
- Aber dadurch wird der Gewichtsvektor zu stark an die Trainingsdaten angepasst (Overfitting).
- Wenn das Training früher gestoppt wird, wird der Gewichtsvektor von den zuletzt gesehenen Beispielen dominiert.
- Lösung: Berechne das Mittel aller Gewichtsvektoren während des Trainings.

Averaged-Perzeptron Trainingsalgorithmus

Eingabe: Trainingsdaten (x_i, y_i) for $i = 1...n$

Initialisierung: $\theta = 0$

Algorithmus:

for $t = 1...T$ // für T Iterationen
    Wähle zufällig ein i in $1...n$
    $z \leftarrow \arg\max_y \theta \cdot f(x_i, y)$ // klassifiziere
    if $z \neq y_i$ // Fehlklassifikation?
        $\theta \leftarrow \theta + f(x_i, y_i) - f(x_i, z)$ // Gewichtsanpassung
    $\theta_s \leftarrow \theta_s + \theta$
return θ_s / T

Stoppkriterium (2)

Die optimale **Zahl von Iterationen** wird auf **Held-Out-Daten** ermittelt:

1. Trainiere für bspw. 20 Epochen auf den Trainingsdaten.
2. Speichere den gemittelten Gewichtsvektor nach jeder Epoche.
3. Evaluiere die gespeicherten Gewichtsvektoren auf den Held-Out-Daten.
4. Wähle den Gewichtsvektor mit dem besten Ergebnis.

Anmerkung: Hier wird ein etwas modifizierter Perzeptron-Algorithmus angenommen, bei dem in einer äußeren Schleife über x Epochen iteriert wird und in einer inneren Schleife über alle Trainingsdaten.

Maximum-Entropy-Modelle: Textklassifikation

Nachteile des Perzeptron-Algorithmus

Training mit dem Perzeptron-Algorithmus funktioniert recht gut, aber:

- Das Verfahren ist eher heuristisch.
- Es gibt noch bessere Trainingsverfahren.

Maximum-Entropy-Modelle (logistische Regression)

- berechnen einen Score auf Basis der Merkmale
- konvertieren den Score dann in eine Wahrscheinlichkeit
- werden mit dem Gradientenanstiegsverfahren trainiert

Definition von Wahrscheinlichkeiten

Die "Scores" sind beliebige reelle Zahlen:

$$\sum_i \theta_i f_i(c,d) = \theta \cdot f(c,d)$$

Exponentiation der Scores liefert nicht-negative Werte:

$$e^{\theta \cdot f(c,d)}$$

Normalisierung liefert Werte, die zu 1 summieren:

$$p(c|d) = \frac{1}{Z} e^{\theta \cdot f(c,d)} \quad \text{mit } Z = \sum_{c'} e^{\theta \cdot f(c',d)}$$

Ein solches Modell heißt Maximum-Entropy-Modell.

Die Exponentiation mit nachfolgender Normalisierung wird auch als Softmax bezeichnet.

Beispiel

Text = Der Dax steigt und steigt

Index	Merkmal	Gewicht
...		
113	Dax-Wirtschaft	1.5
...		
287	steigen-Wirtschaft	0.3
...		
1023	Dax-Sport	-4.5
...		
1984	steigen-Sport	1.3
...		

$$\begin{aligned} s(\text{Wirtschaft},\text{Text}) &= f_{113}(\text{Wirtschaft},\text{Text})\ \theta_{113} + f_{287}(\text{Wirtschaft},\text{Text})\ \theta_{287} \\ &= 1 * 1.5 + 2 * 0.3 = 2.1 \end{aligned}$$

$$\begin{aligned} s(\text{Sport},\ \text{Text}) &= f_{1023}(\text{Sport},\text{Text})\ \theta_{1023} + f_{1984}(\text{Sport},\text{Text})\ \theta_{1984} \\ &= 1 * (-4.5) + 2 * 1.3 = -1.9 \end{aligned}$$

$$p(\text{Wirtschaft}|\text{Text}) = \frac{e^{2.1}}{e^{2.1}+e^{-1.9}} = 0.982$$

$$p(\text{Sport}|\text{Text}) = \frac{e^{-1.9}}{e^{2.1}+e^{-1.9}} = 0.018$$

Merkmalsfunktionen, deren Werte 0 sind, werden in den Berechnungen ignoriert.

Maximum-Entropy-Modelle

Bedingte Wahrscheinlichkeit der Klasse y gegeben das Objekt x

$$p_\theta(y|x) = \frac{1}{Z_\theta(x)} e^{\theta \cdot f(x,y)}$$

Im Training passen wir die Parameter so an, dass die bedingte Wahrscheinlichkeit (Likelihood) der Trainingsdaten D maximiert wird:

$$\hat{\theta} = \arg\max_\theta \prod_{(x,y)\in D} p_\theta(y|x)$$

Äquivalent dazu können wir auch die Log-Likelihood maximieren:

$$\hat{\theta} = \arg\max_\theta \log \prod_{(x,y)\in D} p_\theta(y|x) = \arg\max_\theta \sum_{(x,y)\in D} log\ p_\theta(y|x)$$

Umformung der Log-Likelihood

$$
\begin{aligned}
LL_D(\theta) &= \sum_{(x,y)\in D} log\ p_\theta(y|x) \\
&= \sum_{(x,y)\in D} log\left(\frac{1}{Z_\theta(x)} e^{\theta\cdot f(x,y)}\right) \\
&= \sum_{(x,y)\in D} \left(log \frac{1}{Z_\theta(x)} + log\ e^{\theta\cdot f(x,y)}\right) \\
&= \sum_{(x,y)\in D} \theta\cdot f(x,y) - \sum_{(x,y)\in D} log\ Z_\theta(x)
\end{aligned}
$$

Um das Maximum der Log-Likelihood in Abhängigkeit von den Parametern θ_i zu finden, setzen wir die partiellen Ableitungen nach θ_i gleich 0.

Ableitungsregeln

- Ableitung von Konstanten: $\frac{d}{dx} y = 0$
- Ableitung von Polynomen: $\frac{d}{dx} x^n = n x^{n-1}$
- Ableitung des Logarithmus: $\frac{d}{dx} \log x = \frac{1}{x}$
- Ableitung der Exponentialfunktion: $\frac{d}{dx} e^x = e^x$
- Ableitung einer Summe: $\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx} f(x) + \frac{d}{dx} g(x)$
- Ableitung eines Produktes: $\frac{d}{dx}(f(x)g(x)) = \frac{df(x)}{dx} g(x) + f(x) \frac{dg(x)}{dx}$
- Kettenregel: $\frac{d}{dx} f(g(x)) = \frac{df(g(x))}{dg(x)} \frac{dg(x)}{dx}$

Partielle Ableitungen

Wenn wir eine Funktion gleichzeitig nach mehreren Variablen ableiten, sprechen wir von partiellen Ableitungen. Die partiellen Ableitungen werden zu einem Vektor zusammengefasst, dem Gradienten:

Für $f(x_1, x_2, x_3) = 3x_1 + 2x_2^2 + x_3^3$ erhalten wir

$$\operatorname{grad} f(x_1, x_2, x_3) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2, x_3) \\ \frac{\partial}{\partial x_2} f(x_1, x_2, x_3) \\ \frac{\partial}{\partial x_3} f(x_1, x_2, x_3) \end{pmatrix} = \begin{pmatrix} 3 \\ 4x_2 \\ 3x_3^2 \end{pmatrix}$$

Partielle Ableitungen der Log-Likelihood

$$\frac{\partial}{\partial \theta_i} \sum_{(x,y)\in D} \theta \cdot f(x,y) - \sum_{(x,y)\in D} log\ Z_\theta(x) = 0$$

$$\frac{\partial}{\partial \theta_i} \theta \cdot f(x,y) = \frac{\partial}{\partial \theta_i} \sum_{j=1}^{m} \theta_j f_j(x,y) = f_i(x,y)\ (=\text{beobachtete Hfk.})$$

Die Ableitung von $\theta_j f_j(x,y)$ nach θ_i ist $f_j(x,y)$ für $i = j$ und sonst 0.

$$\begin{aligned}
\frac{\partial}{\partial \theta_i} log\ Z_\theta(x) &= \frac{\partial}{\partial \theta_i} log \sum_{y'} e^{\sum_{j=1}^{m} \theta_j f_j(x,y')} \\
&= \frac{1}{\sum_{y'} e^{\sum_{j=1}^{m} \theta_j f_j(x,y')}} \left(\sum_{y'} e^{\sum_{j=1}^{m} \theta_j f_j(x,y')} f_i(x,y') \right) \\
&= \sum_{y'} p_\theta(y'|x)\ f_i(x,y') \\
&= E_{p_\theta(y'|x)}(f_i(x,y'))\ (=\text{erwartete Hfk.})
\end{aligned}$$

Maximum-Entropy-Modelle

Zusammensetzen der Teilergebnisse:

$$\frac{\partial}{\partial \theta_i} LL_D(\theta) = \sum_{(x,y)\in D} f_i(x,y) - \sum_{(x,y)\in D} E_{p_\theta(y'|x)}(f_i(x,y')) = 0$$

$$\sum_{(x,y)\in D} f_i(x,y) = \sum_{(x,y)\in D} E_{p_\theta(y'|x)}(f_i(x,y'))$$

$\Rightarrow$ Die erwartete Häufigkeit jedes Merkmales (rechte Summe) soll gleich der beobachteten Häufigkeit (linke Summe) werden.

$\Rightarrow$ Die Ableitung nach dem Gewicht θ_i eines Merkmales ist die Differenz zwischen beobachteter und erwarteter Häufigkeit des Merkmales.

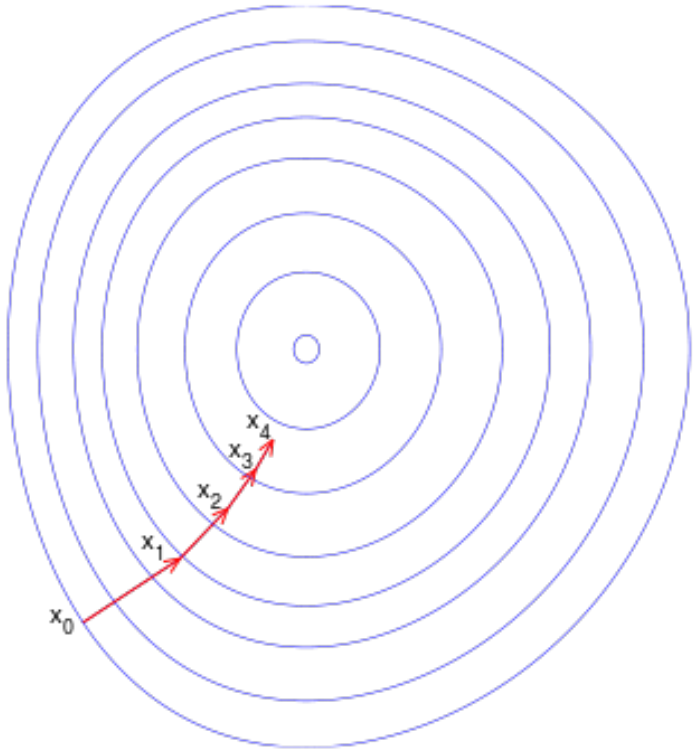
$\Rightarrow$ Der Gradient $\nabla LL_D(\theta)$ ist ein Vektor gebildet aus allen partiellen Ableitungen $\frac{\partial}{\partial \theta_i} LL_D(\theta)$.

Training mit Gradientenanstieg

Um eine Funktion f zu maximieren, müssen wir die Parameter in Richtung steigender Werte von f verändern (Gradientenanstieg).

Algorithmus

initialize θ
for T iterations do
$\theta \leftarrow \theta + \eta \nabla f(\theta)$



aus Wikipedia

Die Lernrate η bestimmt, wie groß die einzelnen Schritte sind.
Wenn eine Funktion minimiert werden muss, spricht man von Gradientenabstieg.

Dann wird der Gradient $\nabla f(\theta)$ subtrahiert.

Varianten von Gradientenanstieg

- (Batch-)Gradientenanstieg
  - berechnet die Ableitung für alle Daten D zusammen
  - Anpassung der Gewichte, nachdem alle Daten verarbeitet wurden
  - lernt langsam
- Stochastischer Gradientenanstieg
  - berechnet die Ableitung für jeden Datensatz einzeln
  - passt die Gewichte nach jedem Datensatz an
  - lernt schneller, aber die Likelihood der Daten stiegt nicht monoton
- Minibatch-Gradientenanstieg
  - Gewichtsanpassung nach der Verarbeitung von B Datensätzen (B = Minibatch-Größe)
  - Obige Methoden sind Spezialfälle davon (B=|D|) bzw. B=1

Regularisierung

Bei begrenzten Trainingsdaten besteht die Gefahr des Overfitting.

Gegenmaßnahme: Große Gewichte bestrafen

modifizierte Ziel-Funktion:

$$LL_D(\theta) - \frac{\mu}{2} \sum_i \theta_i^2 \quad (L_2\text{-Regularisierung})$$

$$LL_D(\theta) - \mu \sum_i |\theta_i| \quad (L_1\text{-Regularisierung})$$

$\Rightarrow$ Große Gewichte sind nur erlaubt, wenn sie die Likelihood stark erhöhen.

μ bestimmt, wie stark regularisiert wird. Keine Regularisierung für $\mu = 0$.
μ kann auf Development-Daten optimiert werden und hat einen Wert ≥ 0 und (normalerweise) < 1.

Partielle Ableitungen der Regularisierungsterme

L_2-Regularisierung:

$$\frac{\partial}{\partial \theta_i} \frac{\mu}{2} \sum_j \theta_j^2 = \mu \, \theta_i$$

L_1-Regularisierung:

$$\frac{\partial}{\partial \theta_i} \mu \sum_j |\theta_j| = \mu \, \mathit{sign}(\theta_i)$$

$$\mathit{sign}(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{falls } x < 0 \end{cases}$$

Regularisierung

modifizierter Trainingsalgorithmus

initialize θ
for T iterations do
$\theta \leftarrow \theta + \eta(\nabla LL_D(\theta) - \mu\,\theta)$ (falls L_2-Regularisierung)
$\theta \leftarrow \theta + \eta(\nabla LL_D(\theta) - \mu\,sign(\theta))$ (falls L_1-Regularisierung)

Alternativ können die neuen Parameterwerte auch so berechnet werden:

$\theta \leftarrow \theta(1 - \eta\mu) + \eta\nabla LL_D(\theta)$ Multiplikation eines Weight-Decay-Faktors
$\theta \leftarrow \theta - \eta\mu\,sign(\theta) + \eta\nabla LL_D(\theta)$ Subtraktion eines Weight-Decay-Terms

Regularisierung

Weitere Regularisierungsmethoden:

- Mittelung der (letzten) Gewichtsvektoren (wie beim Averaged Perceptron)
- Optimale Zahl der Trainingsiterationen mit Heldout-Daten bestimmen

Andere Lernalgorithmen

Neben (stochastischem) Gradientenanstieg/-abstieg gibt es weitere Trainingsverfahren

- Generalized Iterative Scaling (veraltet)
- Conjugate Gradient (aufwendigere Implementierung)
- L-BFGS (sehr aufwendige Implementierung, konvergiert schnell, nur Batchtraining)

Conditional Random Fields: Wortart-Tagging

Taggen mit Maximum-Entropy-Modellen

Zwei Methoden der Implementierung von Wort(art)-Taggern

- jedes Tag unabhängig von allen anderen Tags zuweisen

  Problem: inkonsistente Ausgaben
  Das ist die/Nom beste/Nom Lösung/Akk ./$.

  Nach dem Satzanfang *"Das ist"* ist eine Nominativ-NP wahrscheinlich. Am Satzende ist dagegen eine Akkusativ-NP wahrscheinlicher.

- alle Wörter gemeinsam desambiguieren (analog HMMs)
  $\Rightarrow$ (Linear Chain) Conditional Random Field
  Hier werden inkonsistente Taggingergebnisse bestraft.

Linear-Chain Conditional Random Field

definiert die bedingte Wahrscheinlichkeit einer Tagfolge gegeben eine Wortfolge

- Für jede Wortposition i wird ein Merkmalsvektor $f(t_{i-k}^{i}, w_1^n, i)$ extrahiert.
  - lexikalische Merkmale (ohne Nachbartag-Information)
    Wort+Tag, Wortsuffix+Tag, Wort"shape"+Tag, voriges Wort+Tag, nächstes Wort+Wort+Tag, Wort+Tag im Lexikon
  - Kontext-Merkmale (mit Nachbartag-Information)
    voriges Tag+Tag, letzte2Tags+Tag, voriges Tag+Wort+Tag
- Die Merkmalsvektoren $f(t_{i-k}^{i}, w_1^n, i)$ werden über alle Positionen i addiert und mit dem Gewichtsvektor multipliziert:

$$\theta \cdot \sum_i f(t_{i-k}^{i}, w_1^n, i)$$

- Mit der Softmax-Operation wird die bedingte Wk. der Tagfolge definiert:

$$p(t_1^n | w_1^n) = \frac{1}{Z(w_1^n)} e^{\theta \cdot \sum_{i=1}^{n+1} f(t_{i-k}^{i}, w_1^n, i)} \quad Z(w_1^n) = \sum_{t_1^n} e^{\theta \cdot \sum_{i=1}^{n+1} f(t_{i-k}^{i}, w_1^n, i)}$$

Conditional Random Fields

Berechnung der besten Tagfolge

$$
\begin{aligned}
\hat{t_1^n} &= \arg\max_{t_1^n} \frac{1}{Z(w_1^n)} e^{\theta \cdot \sum_{i=1}^{n+1} f(t_{i-k}^i, w_1^n, i)} \\
&= \arg\max_{t_1^n} \frac{1}{Z(w_1^n)} e^{\sum_{i=1}^{n+1} \theta \cdot f(t_{i-k}^i, w_1^n, i)} \\
&= \arg\max_{t_1^n} \frac{1}{Z(w_1^n)} \prod_{i=1}^{n+1} e^{\theta \cdot f(t_{i-k}^i, w_1^n, i)} \\
&= \arg\max_{t_1^n} \prod_{i=1}^{n+1} e^{\theta \cdot f(t_{i-k}^i, w_1^n, i)}
\end{aligned}
$$

Die Formel entspricht der HMM-Formel, wenn man $p(t_i|t_{i-k}^{i-1})\ p(w_i|t_i)$ durch $e^{\theta \cdot f(t_{i-k}^i, w_1^n, i)}$ ersetzt.

$\Rightarrow$ Wie bei den HMMs kann man mit dem Viterbi-Algorithmus taggen.

Viterbi-Algorithmus für Conditional Random Fields

$$
\begin{aligned}
s(t', t, w_1^n, i) &= e^{\theta \cdot f(t', t, w_1^n, i)} \\
\delta_t(0) &= 1 \text{ falls } t = \langle s \rangle \text{ sonst } 0 \\
\delta_t(i) &= \max_{t'} \delta_{t'}(i-1)\, s(t', t, w_1^n, i) \quad \text{für } 1 \leq i \leq n+1 \\
\psi_t(i) &= \arg\max_{t'} \delta_{t'}(i-1)\, s(t', t, w_1^n, i) \quad \text{für } 1 \leq i \leq n+1
\end{aligned}
$$

Besser noch mit logarithmierten Werten zur Vermeidung von Underflow:

$$
\begin{aligned}
s(t', t, w_1^n, i) &= \theta \cdot f(t', t, w_1^n, i) \\
\delta_t(0) &= 0 \text{ falls } t = \langle s \rangle \text{ sonst } -\inf \\
\delta_t(i) &= \max_{t'} \delta_{t'}(i-1) + s(t', t, w_1^n, i) \quad \text{für } 1 \leq i \leq n+1 \\
\psi_t(i) &= \arg\max_{t'} \delta_{t'}(i-1) + s(t', t, w_1^n, i) \quad \text{für } 1 \leq i \leq n+1
\end{aligned}
$$

Extraktion der besten Tagfolge $t_1, ..., t_n$

$$
\begin{aligned}
t_n &= \psi_{\langle /s \rangle}(n+1) \\
t_{i-1} &= \psi_{t_i}(i) \quad \text{für } 1 < i < n+1
\end{aligned}
$$

CRF-Beispiel

Wortfolge (mit nur einem Wort): Hallo

Tagset: INJ, XY, $\langle/s\rangle$

Im Folgenden schreiben wir Merkmalsvektoren als eine Liste von Key-Value-Paaren, in denen nur Merkmale mit von 0 verschiedenen Werten auftauchen.
Bei der Multiplikation mit dem Gewichtsvektor schreiben wir den Gewichtsvektor als eine Funktion, die Merkmalsnamen auf das entsprechende Gewicht abbildet.

Gewichtsvektor

TW-INJ-Hallo	3.7
TW-XY-Hallo	-2.5
TW-$\langle/s\rangle$-ϵ	11.5
...	
TT-$\langle s\rangle$-INJ	2.5
TT-$\langle s\rangle$-XY	1.9
TT-INJ-INJ	0.5
TT-INJ-XY	-0.3
TT-INJ-$\langle/s\rangle$	-0.1
TT-XY-INJ	-0.6
TT-XY-XY	3.2
TT-XY-$\langle/s\rangle$	0.2
...	

Score der Tagfolge INJ für die Wortfolge Hallo:

s([INJ], [Hallo]) = θ·f([INJ], [Hallo]) Multiplikation mit globalem Merkmalsvektor
= $\theta \cdot$ (f($\langle s\rangle$, INJ, [Hallo], 1) + f(INJ, $\langle/s\rangle$, [Hallo], 2)) Summe der lokalen Vektoren
= $\theta \cdot$ ([TT-$\langle s\rangle$-INJ:1, TW-INJ-Hallo:1] + [TT-INJ-$\langle/s\rangle$:1, TW-$\langle/s\rangle$-ϵ:1])
= $\theta \cdot$ [TT-$\langle s\rangle$-INJ:1, TW-INJ-Hallo:1, TT-INJ-$\langle/s\rangle$:1, TW-$\langle/s\rangle$-ϵ:1]
= θ(TT-$\langle s\rangle$-INJ) * 1 + θ(TW-INJ-Hallo) * 1 + θ(TT-INJ-$\langle/s\rangle$) * 1 + θ(TW-$\langle/s\rangle$-ϵ) * 1
= 2.5 * 1 + 3.7 * 1 + (-0.1) * 1 + 11.5 * 1 = 17.6
s([XY], [Hallo]) = 1.9 * 1 + (-2.5) * 1 + 0.2 * 1 + 11.5 * 1 = 11.1
Normalisierungskonstante: Z([Hallo]) = $e^{17.6} + e^{11.1}$ = 44079365
p([INJ]|[Hallo]) = $e^{17.6}$/Z([Hallo]) = 44013194/44079365 = 0.9985

Viterbi-Beispiel

Wortfolge: Hallo

Tagset: INJ, XY, $\langle /s \rangle$

Gewichtsvektor: wie auf der letzten Folie

Berechnung der **Viterbi-Scores** $\delta(i,t)$:

$$
\begin{aligned}
\delta_{\langle s \rangle}(0) &= log(1) = 0 \\
\delta_{INTJ}(1) &= max_{t \in \{\langle s \rangle\}}\ \delta_t(0) + \theta \cdot f(t, INTJ, (Hallo), 1) \\
&= \delta_{\langle s \rangle}(0) + \theta \cdot [\text{TT-}\langle s \rangle\text{-INTJ} : 1, \text{TW-INJ-Hallo} : 1] \\
&= 0 + \theta(\text{TT-}\langle s \rangle\text{-INTJ}) * 1 + \theta(\text{TW-INJ-Hallo}) * 1 \\
&= 0 + 2.5 * 1 + 3.7 * 1 = 6.2 \\
\delta_{XY}(1) &= max_{t \in \{\langle s \rangle\}}\ \delta_t(0) + \theta \cdot f(t, XY, (Hallo), 1) \\
&= 0 + 1.9 * 1 + (-2.5) * 1 = -0.6 \\
\delta_{\langle /s \rangle}(2) &= max_{t \in \{INTJ, XY\}}\ \delta_t(1) + \theta \cdot f(t, \langle /s \rangle, (Hallo), 2) \\
&= max(6.2 + (-0.1) * 1 + 11.5 * 1, -0.6 + 0.2 * 1 + 11.5 * 1) \\
&= max(17.6, 11.1) = 17.6
\end{aligned}
$$

Training von Conditional Random Fields

- Für das Training mit Gradientenanstieg werden die erwarteten Merkmalshäufigkeiten benötigt.
- Wie bei den HMMs können diese mit dem Forward-Backward-Algorithmus berechnet werden.
- Hier wird der FB-Algorithmus aber für überwachtes Training eingesetzt.

Training von Conditional Random Fields

Wahrscheinlichkeit der Tagfolge t_1^n gegeben die Wortfolge w_1^n:

$$p(t_1^n|w_1^n) = \frac{1}{Z(w_1^n)} \prod_{i=1}^{n+1} e^{\theta \cdot f(t_{i-m}^i, w_1^n, i)}$$

Zur Berechnung der erwarteten Merkmalshäufigkeiten eines lex. Merkmals muss die Aposteriori-Wahrscheinlichkeit jedes Tags berechnet werden:

$$\frac{\text{Wk. aller Tagfolgen mit Tag } t \text{ an Position } i}{\text{Wk. aller Tagfolgen insgesamt}}$$

- Die Normalisierungskonstanten $Z(w_1^n)$ können im Zähler und Nenner jeweils aus der Summe ausgeklammert und dann gekürzt werden.
- Die obere Summe kann als Produkt von Forward- und Backward-Werten berechnet werden (siehe nächste Folie)

Training von Conditional Random Fields

Forward-Algorithmus (Modell 1. Ordnung)

$$
\begin{aligned}
s(t', t, w_1^n, i) &= e^{\theta \cdot f(t', t, w_1^n, i)} \\
\alpha_{\langle s \rangle}(0) &= 1 \\
\alpha_t(i) &= \sum_{t'} \alpha_{t'}(i-1)\, s(t', t, w_1^n, i)
\end{aligned}
$$

Backward-Algorithmus

$$
\begin{aligned}
\beta_{\langle /s \rangle}(n+1) &= 1 \\
\beta_t(i-1) &= \sum_{t'} \beta_{t'}(i)\, s(t, t', w_1^n, i)
\end{aligned}
$$

Aposteriori-Wahrscheinlichkeiten

$$
\begin{aligned}
\gamma_t(i) &= \frac{\alpha_t(i)\beta_t(i)}{\alpha_{\langle /s \rangle}(n+1)} \\
\gamma_{tt'}(i) &= \frac{\alpha_t(i-1)\, s(t, t', w_1^n, i)\, \beta_{t'}(i)}{\alpha_{\langle /s \rangle}(n+1)}
\end{aligned}
$$

Training von Conditional Random Fields

- Das Training von CRFs ist iterativ.
- In jeder Iteration werden die Aposteriori-Wahrscheinlichkeiten $\gamma_{tt'}(i)$ der Tagpaare mit dem Forward-Backward-Algorithmus berechnet.
- Die $\gamma_{tt'}(i)$ werden dann mit den Merkmalsvektoren $f(t, t', w_1^n, i)$ multipliziert und über alle Sätze und Satzpositionen aufsummiert, um die erwarteten Merkmalshäufigkeiten zu berechnen.
- Außerdem werden die beobachteten Merkmalshäufigkeiten berechnet, indem für alle Positionen i die jeweiligen Merkmalsvektoren $f(t_{i-1}, t_i, w_1^n, i)$ berechnet und aufsummiert werden.

  t_i ist hier das korrekte Tag an Position i.
- Von den beobachteten Merkmalshäufigkeiten werden die erwarteten Merkmalshäufigkeiten abgezogen, um den Gradienten zu erhalten.
- Der Gradient wird mit der Lernrate multipliziert und zum Gewichtsvektor addiert. (Gradientenanstieg)

Training von Conditional Random Fields: Pseudocode

θ[f] = 0 for all features f weight vector
for N iterations:
    Choose a minibatch d from the data D
    ef[f] = 0 for each feature f expected feature values
    of[f] = 0 for each feature f observed feature values
    for each word sequence w and tag sequence t
        α = forward(w)
        β = backward(w)
        γ = posteriorprobs(α,β)
        for i=1 to n+1
            for tag t in tagset
                for previous tag t' in tagset
                    for f, v in features&values(t',t,w,i)
                        ef[f] += v $\cdot$ $\gamma_{t',t}(i)$
        for i=1 to n+1
            for f, v in features&values(t_{i-1},t_i,w,i)
                of[f] += v
    for each feature f
        θ[f] += η (of[f] - ef[f])

wobei η die Lernrate ist.

LC-CRFs

Hauptprobleme bei der Entwicklung von LC-CRFs

- gute Merkmale finden
  → Feature Engineering, linguistische Intuition hilfreich
- optimale Kombination der Merkmale bestimmen
  weniger wichtig bei gut funktionierender Regularisierung
- effiziente Berechnung
  - Die Berechnung der Merkmalsfunktionen ist aufwendig.
  - Der Zeitbedarf steigt exponentiell mit der Ordnung des CRFs.
    wie beim HMM

LC-CRFs

Effiziente Methode von Thomas Müller für LC-CRFs höherer Ordnung

Idee: Benutze ein einfaches Modell, um die Menge möglicher Lösungen einzuschränken.

Schritte:

1. wahrscheinlichste Tags für jedes Wort auf Basis der lexikalischen Merkmale berechnen
2. wahrscheinlichste Tags für jedes Wort auf Basis eines Modelles 1. Ordnung berechnen
3. wahrscheinlichste Tags für jedes Wort auf Basis des Modelles maximaler Ordnung berechnen

- Jede Stufe berücksichtigt nur die besten Tags der Vorstufe
- fast so effizient wie CRF 1. Ordnung, aber genauer

Überblick über die statistischen Modelle

- Naïve-Bayes-Modell zur Textklassifikation

$$p(c, w_1, ..., w_n) = p(c) \prod_{i=1}^{n+1} p(w_i|c)$$

Eine Klasse c bekommt eine gute Bewertung,

  - wenn sie häufig ist und
  - wenn die **Wörter** w_i häufig in Trainingstexten dieser Klasse vorkamen.

- Markowmodell zur Sprachidentifizierung

$$p_L(a_1, ..., a_n) = \prod_{i=1}^{n+1} p_L(a_i|a_{i-k}, ..., a_{i-1})$$

Eine Sprache L bekommt eine gute Bewertung, wenn die **Buchstaben-Ngramme** des Textes häufig in den Trainingstexten dieser Klasse auftauchten.

- PCFG zum Parsen $\quad p(T) = p(r_1, ..., r_n) = \prod_{i=1}^{n} p(r_i)$

Ein Parse T mit Linksableitung $r_1, ..., r_n$ bekommt eine gute Bewertung, wenn die verwendeten **Regeln** in der Trainings-Baumbank häufig waren.

Überblick über die statistischen Modelle

- Hidden-Markow-Modell für Tagginganwendungen

$$p(t_1, ..., t_n, w_1, ..., w_n) = \prod_{i=1}^{n+1} p(t_i|t_{i-k}, ..., t_{i-1})\ p(w_i|t_i)$$

Eine Tagfolge t_1^n bekommt eine gute Bewertung,

- wenn die **k+1-Tag-Gramme** häufig in den Trainingsdaten vorkamen
- wenn die **Tag-Wort-Paare** häufig in den Trainingstexten auftraten.

- Conditional Random Fields für Tagging

$$p(t_1, ..., t_n|w_1, ..., w_n) = softmax\left(\sum_{i=1}^{n+1} \theta \cdot f(t_{i-1}, t_i, i, w_1, ..., w_n)\right)$$

Eine Tagfolge t_1^n bekommt eine gute Bewertung,

- wenn die Merkmale der Tagfolge **hohe Gewichte** besitzen
- und die Merkmale anderer Tagfolgen **geringe Gewichte** besitzen.

ENDE