1. Einführung in die Statistik

Leitfragen zu Abschnitt 1

- Was soll im Kurs gelernt werden?
- Wie ist der Kurs organisiert?
- Wo sind weitere Informationen zu finden?

Worum geht es?

- Was ist Wahrscheinlichkeit?
- Wie bestimmt man Wahrscheinlichkeiten?
- Was ist Information?
- Wie plausibel sind Hypothesen?
- Stochastische Prozesse

Technisches

Schein: Fleißig zu Vorlesung und Tutorium kommen, Klausur *oder* Abschlusskolloquium.

Skript und Informationen zur Vorlesung: http://www/kurs/ws05/stat[1]

Tutorium

Statt Hausaufgaben mit Leistungskontrolle solltet ihr jeweils Lösungen und Gedanken zu den Aufgaben hier im Skript mit in die Tutorien bringen.

Literatur

- Manning, Schütze: Foundations of Statistical Language Processing, Cambridge, MA, 2001: The MIT Press – das Standardwerk, teuer
- Carstensen et al: Computerlinguistik und Sprachtechnologie, Heidelberg, 2001: Spektrum akad. Verlag – Allrounder, Statistikteil eher knapp
- Krengel: Einführung in die Wahrscheinlichkeitstheorie und Statistik, Braunschweig 2000: Vieweg – ein Mathematikbuch, für die, die es wirklich wissen wollen
- The Linguist's Guide to Statistics[2] – etwas unfertiges Werk, aber dafür umsonst

[1] http://www.cl.uni-heidelberg.de/kurs/ws05/stat

[2] http://docs.cl.uni-heidelberg.de/statcl.ps.gz

2. Wohin geht die Reise?

n-gram

word-type

word-token

Leitfragen zu Abschnitt 2

- Wo in der Computerlinguistik werden statistische Techniken verwendet?
- Was sind n-gramme?
- Was ist der Unterschied zwischen word types und word tokens?

Statistische Techniken werden mittlerweile auf fast allen Feldern der Sprachanalyse und -generierung verwendet. Ein paar Stichworte:

- Spracherkennung: HMMs (Hidden Markov Models) zur Erkennung von Phonen, n-gram-Modelle zur Beurteilung der Plausibilität von Ergebnissen
- Information Retrieval: Beste Entsprechungen für Suchanfragen, Clustering, Textkategorisierung
- Tagging: Erkennung von Wortklassen (HMMs, n-gram-Modelle, heuristische Verfahren)
- Parsing: Reduzierung des Ambiguitätsproblems mit probabilitischen Grammatiken, PP-Attachment
- Maschinelle Übersetzung: Das Übersetzungsproblem als Anwendung eines *noisy channel*
- Machine Learning
- ...

n-gramme

Ein *n-gram* ist einfach eine Folge von n Zeichen. Speziell: 2-gramme heißten Bigramme, 3-gramme Trigramme. Eine Bigrammdarstellung dieses Satzes: (., Eine), (Eine, Bigrammdarstellung), (Bigrammdarstellung, dieses), (dieses, Satzes), (Satzes, .). Der Punkt steht dabei für Satzanfang und Satzende.

word-types vs. word-tokens

Ein *word-type* ist ein bestimmtes Wort, etwa „blau“, ein *word-token* dagegen das Auftreten eines Wortes. Ein Text mit 200 word-types enthält tatsächlich 200 verschiedene Wörter und mag aus 500 word-tokens bestehen.

3. Propädeutikum: Mengen

Leitfragen zu Abschnitt 3

- Was ist eine Menge?
- Warum bestimmt die Cantor'sche Mengendefinition schon recht weitgehend Begriffe wie Gleichheit oder Teilmengen?
- Warum ist die Potenzmenge ein furchterregendes Konstrukt für Menschen, die Programme schreiben?
- Wie unterscheiden sich die Rechenregeln für Mengen von denen für natürliche Zahlen, wo entsprechen sich die Regeln?

Menge

Teilmenge

Extensionalitäts-prinzip

leere Menge

Potenzmenge

Vereinigung

Schnitt

Differenz

Komplementbildung

DeMorgan

Eine *Menge* ist eine „Zusammenfassung von [...] Objekten unserer Anschauung oder unseres Denkens“ (Cantor, 1895). Diese Definition ist schlecht, weil sie auf Widersprüche führt. Eine Definition der Menge, die tragfähig ist, ist aber kompliziert und interessiert hier nicht.

Mengen notieren wir in Großbuchstaben, ihre Elemente zählen wir in geschweifen Klammern auf: $M_1 = \{1, 3, 5\}$, $M_2 = \{\textit{rot}, \textit{grün}, \textit{gelb}\}$, $M_3 = \{x \mid x \text{ ist transitives Verb}\}$ (lies: M_3 ist die Menge aller x mit der Eigenschaft, dass x ein transitives Verb ist).

B ist *Teilmenge* von A ($B \subseteq A$), wenn jedes Element von B auch in A enthalten ist, *echte Teilmenge* von A (in Zeichen: $B \subset A$), wenn es zusätzlich mindestens ein $x \in A$ gibt, für das $x \notin B$ gilt.

Zwei Mengen sind gleich, wenn sie die gleichen Elemente haben: $B \subseteq A \wedge A \subseteq B$ (*Extensionalitätsprinzip*)

Die *leere Menge*, $\emptyset = \{x | x \neq x\}$ hat kein Element.

Die *Potenzmenge* $\mathcal{P}(A)$ ist die Menge aller Teilmengen von A. Die Potenzmenge einer n-elementigen Menge hat 2^n Elemente.
$\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

Operationen auf Mengen

Vereinigung: $A \cup B = \{x \mid x \in A \vee x \in B\}$.

Schnitt: $A \cap B = \{x \mid x \in A \wedge x \in B\}$.

Differenz: $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.

Komplementbildung: $\bar{A} = O \setminus A = \{x \in O \mid x \notin A\}$. Dabei heißt O die Obermenge.

Rechenregeln

Hier steht $\circ$ für $\cup$ oder $\cap$

- Kommutativität: $M \circ N = N \circ M$
- Assoziativität: $K \circ (M \circ N) = (K \circ M) \circ N$
- Idempotenz: $M \circ M = M$
- Distributivität: $K \cup (M \cap N) = (K \cup M) \cap (K \cup N)$ (und mit $\cap$ und $\cup$ vertauscht)

DeMorgan-Regeln:

$$\overline{M \cup N} = \overline{M} \cap \overline{N}$$
$$\overline{M \cap N} = \overline{M} \cup \overline{N}$$

Aufgaben

(3.1) Schreibt die Menge aller geraden Zahlen kleiner als 300, ohne einen Krampf in der Hand zu bekommen.

(3.2) Was ist die Potenzmenge von $\{1, 2, 3\}$?

(3.3) Macht euch klar, wie dramatisch 2^n wächst. Dazu könnt ihr die Laufzeit eines hypothetischen Algorithmus' analysieren, der alle Elemente der Potenzmenge seiner Eingabe durchgehen muss und, meinethalben, konstant eine Mikrosekunde (10^{-6} Sekunden) braucht, um eine Menge anzusehen.

Wie lange braucht er für eine Eingabe mit 10, 100, 1000, 10000 Elementen? Drückt die Laufzeiten jeweils in vernünftigen Einheiten aus (ein Jahr sind ungefähr 3×10^7 Sekunden).

4. Wahrscheinlichkeit I

Leitfragen zu Abschnitt 4

- Was ist Wahrscheinlichkeit? Was braucht man alles, um von einer Wahrscheinlichkeit reden zu können?
- Was ist der Unterschied zwischen unserer Definition und der relativen Häufigkeit, die populär meist für Wahrscheinlichkeit gehalten wird?
- Wodurch wird eine Funktion zum Wahrscheinlichkeitsmaß? Welche dieser Forderungen sind eher stark, welche eher schwach?
- Was sind die Unterschiede zwischen Wahrscheinlichkeitsraum und Wahrscheinlichkeitsmaß, zwischen Ereignis und Ergebnis?

Experiment

relativen Häufigkeit

Ergebnismenge

Stichprobenraum

sample space

Ergebnis

Stichproben

Realisierungen

sample points

basic outcomes

Experiment

Ereignis

event

unmögliche Ereignis

sichere Ereignis

Wahrscheinlich-keitsmaß

Wahrscheinlich-keitsverteilung

probablity function

Normiertheit

Positivität

Additivität

Intuitiv: Wenn ein *Experiment* bei 100 Versuchen im Mittel 30 Mal ein bestimmtes Ergebnis liefert, würden wir sagen, dieses Ergebnis habe eine Wahrscheinlichkeit von 30/100, der *relativen Häufigkeit*. Das ist axiomatisch schwierig zu fassen, deshalb Zugang von der „anderen Seite“.

Eine *Ergebnismenge* (auch *Stichprobenraum*, *sample space*) Ω ist eine endliche (tatsächlich darf sie auch abzählbar unendlich sein, aber um solche mathematischen Feinheiten kümmern wir uns erstmal nicht) Menge von *Ergebnis*sen (*Stichproben*, *Realisierungen*, *sample points*, *basic outcomes*) eines *Experiment*s. Ein Experiment kann dabei ziemlich viel sein: Das Werfen eines Würfels, die Bestimmung des Typs eines Worts, die Frage, ob ein Pixel schwarz ist oder nicht usf.

Teilmengen von Ω heißen *Ereignis* (*event*). Das Ereignis $A \subset \Omega$ ist eingetreten, wenn das Experiment ein Ergebnis $\omega \in A$ geliefert hat.

Sind A, B zwei Ereignisse, so ist

- $A \cup B$ das Ereignis, dass A oder B sich ereignen,
- $A \cap B$ das Ereignis, dass A und B sich ereignen,
- $\bar{A}$ das Ereignis, dass sich A nicht ereignet,
- $\emptyset$ das *unmögliche Ereignis*,
- Ω das *sichere Ereignis*

Wenn $A \cap B = \emptyset$, sind A und B *unvereinbar*.

Die Menge aller Ereignisse ist $\mathcal{P}(\Omega)$. Eine Abbildung

$$P : \mathcal{P}(\Omega) \to [0, 1]$$

heißt *Wahrscheinlichkeitsmaß* oder *Wahrscheinlichkeitsverteilung* (*probablity function*), wenn

- $P(\Omega) = 1$ (*Normiertheit*)
- $P(A) \geq 0$ für alle $A \in \mathcal{P}(\Omega)$ (*Positivität*)
- $P(A \cup B) = P(A) + P(B)$, wenn $A \cap B = \emptyset$ (*Additivität*)

Vorsicht: Es gibt auch eine Verteilungsfunktion (distribution function) – die ist aber was ganz anderes. Unsere Verteilungen heißen auf Englisch Distribution oder, wenn Ω etwas wie die reellen Zahlen ist, auch gern probability density function (PDF, Wahrscheinlichkeitsdichte).

$P(A)$ heißt *Wahrscheinlichkeit* von A. Das Paar (Ω, P) heißt *Wahrscheinlichkeitsraum* (*probability space*). Wahrscheinlichkeitsräume sind unsere *Modelle* für Experimente. In der Sprachverarbeitung ist die Wahl des geeigneten Modells häufig die schwierigste Aufgabe überhaupt. Einerseits ist klar, dass Sprache mit Zufallsexperimenten zunächst wenig zu tun hat, andererseits müssen meist eher gewagte Annahmen gemacht werden, um ein gegebenes Phänomen überhaupt einer statistischen Analyse zugänglich zu machen. Dass statistische Verfahren in der Sprachverarbeitung häufig so gut funktionieren, ist eigentlich eher überraschend.

In einem realen Experiment ist die *absolute Häufigkeit* eines Ereignisses $k_n(A)$ die Zahl der Versuchsausgänge in A in den ersten n Versuchen, die *relative Häufigkeit* $h_n(A) = k_n(A)/n$. Vorerst postulieren wir keinerlei Zusammenhang zwischen h_n und P.

Wahrscheinlichkeit

Wahrscheinlichkeitsraum

probability space

Modell

absolute Häufigkeit

relative Häufigkeit

Wahrscheinlichkeitsfunktion

Gleichverteilung

Laplace-Experiment

Kardinalität

Bernoulli-Experiment

Aufgaben

(4.1)* Was ist der Stichprobenraum für das Werfen eines (fairen, sechsseitigen) Würfels, für das Werfen zweier Würfel, für Sprache (die aus Buchstaben, Wörtern, Phonen, Texten oder was immer ihr wollt bestehen kann). Was sind darin jeweils denkbare Ereignisse?

(4.2) Nehmt an, wir modellieren Sprache als ein Zufallsexperiment über dem Stichprobenraum aller Wörter. Wie sieht das Ereignis „Wort fängt mit A an“ aus?

(4.3)* Was ist der Unterschied zwischen dem Ereignis „1 fällt“ und dem Ergebnis „1 fällt“ beim Würfel?

(4.4)* Sucht euch ein paar Leute und macht mit diesen folgendes Experiment: Alle würfeln 10 Mal (zwecks intellektueller Hygiene am Besten mit dem selben Würfel) und rechnen unter der naiven (und falschen) Annahme, dass Wahrscheinlichkeit die relative Häufigkeit eines Ereignisses sei, die Verteilung „ihres“ Würfels aus. Was stellt ihr fest? Was folgt daraus über die Annahme über das Verhältnis von Wahrscheinlichkeit und relativer Häufigkeit?

5. Wahrscheinlichkeit II

Leitfragen zu Abschnitt 5

- Was sind Laplace-, was sind Bernoulli-Experimente? Warum sind sie interessante Spezialfälle?
- Wie übertragen sich Rechenregeln für Mengen auf Regeln für Wahrscheinlichkeiten?

Für ein Wahrscheinlichkeitsmaß P gelten

- $P(\bar{A}) = 1 - P(A)$, speziell $P(\emptyset) = 0$.
- $A \subset B \Rightarrow P(A) \leq P(B)$.
- $P(A \setminus B) = P(A) - P(A \cap B)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.
- $P(A) = \sum_{\omega \in A} P(\omega)$

Dabei ist $P(\omega)$ eine schludrige, aber übliche Schreibweise für $P(\{\omega\})$. Die Abbildung $\omega \mapsto P(\omega)$ nennen wir *Wahrscheinlichkeitsfunktion*.

Wenn $P(\omega_i) = P(\omega_j)$ für alle $\omega_i, \omega_j \in \Omega$, heißt P *Gleichverteilung*, das zugehörige Experiment *Laplace-Experiment*. Klassisches Beispiel für ein Laplace-Experiment ist das Würfeln mit einem fairen Würfel. Aber auch hinter Urnenmodellen steht ein Laplace-Experiment – wenn es ohne zu viel Krampf geht, ist es meistens vorteilhaft, ein gegebenes Problem als Laplace-Experiment zu formulieren.In Laplace-Experimenten ist $P(\omega) = 1/|\Omega|$ und $P(A) = |A|/|\Omega|$ Die Schreibweise $|A|$ bezeichnet die *Kardinalität* der Menge A, im endlichen Fall einfach die Zahl ihrer Elemente).

Wenn $|\Omega| = 2$, heißt das Experiment *Bernoulli-Experiment*. Ein Element ω_e heißt Erfolg, die zugehörige Wahrscheinlichkeit Erfolgswahrscheinlichkeit. Klassisches Beispiel ist hier der Münzwurf – „Kopf“ könnte hier der Erfolg sein. Im Allgemeinen ist aber die Erfolgswahrscheinlichkeit ungleich $1/2$.

Urnenmodell

Binomialkoeffizient

Beispiel Münzwurf: $\Omega = \{0, 1\}$, $P(1) = 0.5$, $P(0) = 0.5$. Dabei steht 1 etwa für „Brandenburger Tor“ und 0 für „Zahl“.

Beispiel Würfel: $\Omega = \{1, 2, 3, 4, 5, 6\}$, $P(\omega) = 1/|\Omega| = 1/6$. Mögliche Ereignisse wären „Gerade Zahl“, $A = \{2, 4, 6\}$ oder „Größer als zwei“, $B = \{3, 4, 5, 6\}$.

Beispiel drei Münzwürfe:

$$\Omega = \{000, 001, 010, 011, 100, 101, 110, 111\},$$

$P(\omega) = 1/8$. Ereignis „Genau zwei Mal Tor“: $A = \{011, 101, 110\}$, $P(A) = |A|/|\Omega| = 3/8$; Ereignis „Erster und letzter Wurf Tor“: $B = \{101, 111\}$, $P(B) = 1/4$.

Aufgaben

(5.1)* Findet intuitive Erklärungen für Rechenregeln beim Wahrscheinlichkeitsmaß

(5.2)* Beweist $P(A) = P((A \setminus B) \cup (A \cap B)) = P(A \setminus B) + P(A \cap B)$ aus den Axiomen. Leitet daraus $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ und $P(\bar{A}) = 1 - P(A)$ ab.

(5.3) Wenn man Sprache als Folge von Zufallsexperimenten auffasst, in dem nacheinander etwa Zeichen oder Wörter gezogen werden – ist ein Laplaceexperiment dann ein gutes Modell?

(5.4) Im Rahmen einer linguistischen Untersuchung möchte man die Wahrscheinlichkeit wissen, dass hinter einem ‘das’ ein Nomen kommt. Probiert, das als Bernoulli-Experiment zu fassen – was müsste Ω sein?

6. Kombinatorik

Leitfragen zu Abschnitt 6

- Welche bequemen Möglichkeiten gibt es, die Größe endlicher Stichprobenräume zu berechnen?
- Wozu dienen die verschiedenen Varianten des Urnenmodells?
- Wie viele Möglichkeiten gibt es, aus einer bestimmten Menge von Zeichen Wörter zu bilden? Welche vergleichbaren Probleme tauchen in der Computerlinguistik auf und lassen sich mit Kombinatorik klären?

Das *Urnenmodell* erleichtert häufig die Berechnung von Wahrscheinlichkeitsfunktionen. In einer Urne liegen nummerierte Kugeln und werden gezogen. Nach jedem Zug legt man entweder zurück oder nicht, und die gezogenen Kugeln werden entweder angeordnet oder nicht. Fertige Formeln für $|\Omega|$ bei Ziehung von n Kugeln aus N:

	mit Rücklegen	ohne Rücklegen
angeordnet	I N^n	II $(N)_n$
nicht angeordnet	IV $\binom{n+N-1}{n}$	III $\binom{N}{n}$

mit $(N)_n = N(N-1)\cdots(N-n+1) = N!/(N-n)!$ und dem *Binomialkoeffizient*en (das ist einfach eine Größe, die immer wieder auftaucht und drum abgekürzt wird)

$$\binom{N}{n} = \frac{N!}{n!(N-n)!}.$$

Kg	3	1	4
in der Urne	1 2 3 4	1 2 3 4	1 2 3 4

Kg	3	1	4
in der Urne	1 2 3 4	1 2 4	2 4

Fig. 1

Wo kommen diese Formeln her?

In Reihenfolge mit Rücklegen: Wir ziehen n Mal, und auf jedem Platz gibt es N Möglichkeiten.

In Reihenfolge ohne Rücklegen: Wir ziehen n Mal; beim ersten Mal haben wir N Möglichkeiten, beim zweiten Mal noch $N-1$ usf., bis ich n Kugeln gezogen habe. Bei der letzten Ziehung habe ich also $N-n+1$ Möglichkeiten. – Spezialfall dazu: $n=N$, $|\Omega_{\mathsf{II}}| = N!$. Das ist gerade die Zahl der *Permutationen* (also der verschiedenen Anordnungen) von N verschiedenen Dingen.

Ohne Reihenfolge ohne Rücklegen: Das ist praktisch Ω_{II}, nur zählen alle verschiedenen Anordnungen der gleichen Zahlen als ein Element. Jede dieser Anordnungen kommt $n!$-mal vor, also $|\Omega_{\mathsf{III}}| = |\Omega_{\mathsf{II}}|/n!$.

Ohne Reihenfolge mit Rücklegen: Im Prinzip würde es reichen, alle Elemente von Ω_{I} zu nehmen, und die mit identischen Elementen miteinander zu identifizieren. Die einfache Division durch die Größe einer solchen Äquivalenzklasse wie bei der Ableitung von $|\Omega_{\mathsf{III}}|$ aus Ω_{II} geht hier nicht, da z.B. $\{0,0\}$ nur aus $(0,0)$ kommt, während $\{0,1\}$ sowohl aus $(0,1)$ also auch aus $(1,0)$ entsteht – die Klassen sind nicht alle gleich groß.

Deshalb Trick: Unser Versuchsergebnis stellen wir dar als eine Folge von n Einsen und N Nullen. Vor der i-ten Null steht für jede Ziehung der Kugel i je eine Eins. Die Stichprobe „Zwei Mal nullte Kugel, Null Mal erste Kugel, Ein Mal zweite Kugel“ würde also zu 110010. Umwandlung in Urnenmodell: Wir ziehen die Positionen der Einsen aus einer Urne mit $N+n-1$ Kugeln (die letzte Stelle darf keine Eins sein) ohne Zurücklegen und ohne Ordnung; das ist Ω_{III}, also ist $|\Omega_{\mathsf{IV}}| = \binom{N+n-1}{n}$.

Beispiel: Ziehen mit und ohne Rücklegen mit Reihenfolge
3 aus 4:

(cf. Fig. 1)

In der Zeile mit Kg steht dabei die jeweils gezogene Kugel, darunter die Kugeln, die noch gezogen werden können – mit Zurücklegen hat man immer vier Möglichkeiten, ohne Zurücklegen hat man immer weniger Auswahl. Wenn man das ein wenig formalisiert, kommt man auf die oben angegebenen Formeln.

Alternative Interpretation: Verteilung von Murmeln auf Plätze, die Nummer der gezogenen Kugel ist die Nummer des Platzes. Kann ein Platz mehrfach besetzt werden, ist ein Experiment mit Zurücklegen nötig, sind die Murmeln ununterscheidbar, spielt die Reihenfolge der Züge offenbar keine Rolle.

(cf. Fig. 2)

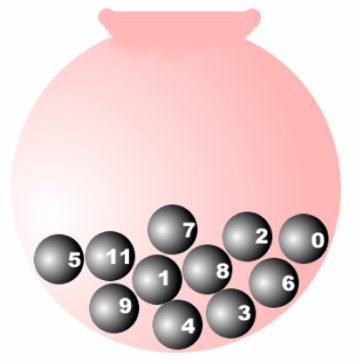

Fig. 2

Aufgaben

(6.1)* Überlegt euch, welche Modelle anwendbar sind für

(a) Zahl der Wörter mit Länge n über einem Alphabet der Länge m.

(b) Zahl der Wörter, die aus einer bestimmten Folge von Buchstaben zu bilden sind (die Zahl der Anagramme).

Denkt euch weitere Probleme dieser Art aus.

(6.2)* Wie viele mögliche Partitionen von N Gegenständen auf n Container gibt es? Diese Frage stellt sich beispielsweise, wenn ihr N Dateien habt, die ihr auf n CDs so verteilen wollt, dass der „Verschnitt“, also der Platz, der ungenutzt bleibt, minimal ist (dabei ist n natürlich variabel, aber das ist noch eine andere Frage).

(6.3) Die Fakultät ist eine sehr schnell wachsende Funktion – und das hat durchaus konkrete Folgen. Nehmt an, ihr hättet eine Folge von Objekten und wolltet eine zufällige Permutation davon ziehen. Nehmt weiter an, ihr hättet eine Funktion, die eine ganze Zahl zwischen 0 und $n!-1$ auf eine dieser Permutationen abbildet und wolltet „zufällig“ eine davon ziehen, indem ihr einen Zufallszahlengenerator eine Zahl ausspucken lasst.

Übliche Zufallszahlengeneratoren geben 32-bit-Zahlen zurück, also Zahlen zwischen 0 und (wenn sie nicht vorzeichenbehaftete Zahlen zurückgeben) $2^{32} = 4\,294\,967\,296$. Bis zu welchem n deckt ihr mit solchen Zufallszahlen alle möglichen Permutationen ab? Welchen Anteil von Permutationen deckt ihr bei $n=20$ ab, welchen bei $n=100$? Fallen euch Auswege aus diesem Problem ein? **(L)**

7. Unabhängigkeit und bedingte Wahrscheinlichkeit

Leitfragen zu Abschnitt 7

- Was sind unabhängige Ereignisse, woran sind sie zu erkennen?
- Warum könnten nicht-unabhängige Ereignisse besonders spannend sein?
- Welche Standardwerkzeuge haben wir, um mit voneinander abhängigien Ereignissen zu arbeiten?
- Gibt es einen Zusammenhang zwischen Kausalität und Bayes-Formel?

Zwei Ereignisse A und B heißen *unabhängig*, gdw. $P(A \cap B) = P(A)P(B)$. In der Realität wird man meistens argumentieren, die Ereignisse seien unabhängig und man dürfe die Formel deshalb so anwenden. Das Eintreten des einen Ereignisses beeinflusst also die Wahrscheinlichkeit des Eintretens des anderen nicht.

Was ist, wenn A und B nicht unabhängig sind?

$P(A \mid B)$ heißt *bedingte Wahrscheinlichkeit* von A unter B, die Wahrscheinlichkeit, dass A eintritt, nachdem B schon eingetreten ist, und ist definiert als

bedingte Wahrscheinlichkeit

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

Diese Definition ist vernünftig. Wir wollen nur Ereignisse aus A betrachten, für die B schon eingetreten ist. Es können also nur noch die Stichproben aus A auftreten, die schon in B sind (deshalb $A \cap B$). Außerdem hätte man gern, dass $P(B \mid B) = 1$ – wenn B schon eingetreten ist, ist B sicher, deshalb die Division durch $P(B)$. Für unabhängige Ereignisse ist

$$P(A \mid B) = \frac{P(A)P(B)}{P(B)} = P(A).$$

Natürlich ist diese Definition nicht anwendbar, wenn $P(B) = 0$.

Beispiel: Würfel. Es werde zwei Mal gewürfelt, A sei das Ereignis, dass beim zweiten Mal eine 1 fällt, B, dass beim ersten Mal eine 1 fällt. Der Stichprobenraum besteht also aus Tupeln (z_1, z_2) mit $z_1, z_2 \in \{1, \ldots, 6\}$.

Offenbar ist $P(A) = P(B) = 1/6$. Was ist $P(A \mid B)$? Es ist

$$P(A \cap B) = \frac{|A \cap B|}{|\Omega|} = \frac{|\{(1,1)\}|}{36}.$$

Damit ist

$$P(A \mid B) = \frac{1/36}{1/6} = \frac{1}{6} = P(A).$$

Wie erwartet, sind die Würfe unabhängig.

Werde nun ein Mal gewürfelt, B sei das Ereignis, dass eine gerade Zahl fällt, A, dass die Vier fällt. Es gilt:

$$P(A \mid B) = \frac{|\{4\}|/6}{|\{2,4,6\}|/6} = \frac{1}{3} \neq \frac{1}{6} = P(A).$$

Natürlich ist die Wahrscheinlichkeit, eine Vier zu finden, größer, wenn ich vorher schon weiß, dass eine gerade Zahl herausgekommen ist.

Ein anderes Beispiel: Sei Ω die Menge aller Texte am Netz (ein Text soll dabei einfach durch die Menge der in ihm vorkommenden Wörter repräsentiert sein), $P(\omega) = 1/|\Omega|$ für alle $\omega \in \Omega$. Wir interessieren uns für die Ereignisse $A = \{\omega \mid \text{Bundeskanzler} \in \omega\}$ und $B = \{\omega \mid \text{Schröder} \in \omega\}$ und nehmen an, dass google Ω ausschöpft, also für die Berechnung von $|A|$, $|B|$ und $|\Omega|$ taugt. Dann ist

$$P(A) = 189\,000/2\,073\,418\,204 = 9.12 \times 10^{-5}$$
$$P(B) = 516\,000/2\,073\,418\,204 = 2.49 \times 10^{-4}$$
$$P(A \cap B) = 79\,100/2\,073\,418\,204 = 3.81 \times 10^{-5}$$

Die Wahrscheinlichkeit, dass ein Text das Wort Bundeskanzler enthält, wenn es das Wort Schröder enthält, ist also

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{3.81 \times 10^{-5}}{2.49 \times 10^{-4}} = 0.15,$$

während die Wahrscheinlichkeit, dass eine Webseite, die Bundeskanzler enhält, auch von Schröder redet

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} = \frac{3.81 \times 10^{-5}}{9.1153824942495783 \times 10^{-5}} = 0.41$$

ist. Interpretiert man Wahrscheinlichkeiten als relative Häufigkeiten, so folgt, dass etwa 15% aller Webseiten, die von Schröders reden, auch über Bundeskanzler reden, während umgekehrt fast die Hälfte aller Seiten, die von Bundeskanzlern erzählen, auch den Namen Schröder enthalten.

Umformung der Definition bedingter Wahrscheinlichkeit:

$$P(A \cap B) = P(A)P(B \mid A)$$

Weil Vereinigung kommutiert, ist das auch gleich $P(B)P(A \mid B)$. Zusammen Bayes'sche Umkehrformel:

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}.$$

Beispiel: Sei A das Ereignis, dass eine bestimmte Person die Krankheit Burizystose hat, B das Ereignis, dass der der Plutopharma-Test der Sachse AG anschlägt. Es sei $P(A) = 0.001$, $P(B \mid \underline{A}) = 0.9$ und die Wahrscheinlichkeit, dass der Test bei Nichtinfizierten positiv ausfällt, $P(B \mid \bar{A}) = 0.01$. Die Wahrscheinlichkeit, dass jemand infiziert ist, wenn der Test positiv ausfällt, ist

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}.$$

$P(B)$, die Wahrscheinlichkeit, dass ein Test ohne Kenntnis des Gesundheitszustands des Probanden positiv ausfällt, kennen wir nicht. Wir können sie aber ausrechnen:

$$\begin{aligned} P(B) =& P\big((B \cap A) \cup (B \cap \bar{A})\big) \\ =& P(B \cap A) + P(B \cap \bar{A}) \\ =& P(A)P(B \mid A) + P(\bar{A})P(B \mid \bar{A}) = 0.011 \end{aligned}$$

Jetzt $P(A \mid B) = 0.001 \cdot 0.9/0.011 = 0.08$, die Wahrscheinlichkeit, dass jemand gesund ist, wenn der Test positiv ausfällt:

$$P(\bar{A} \mid B) = \frac{P(\bar{A})P(B \mid \bar{A})}{P(B)} = 0.91.$$

Aufgaben

(7.1)* Macht euch klar, dass und warum das

$$\begin{aligned} P\big((B \cap A) \cup (B \cap \bar{A})\big) \\ =& P(B \cap A) + P(B \cap \bar{A}), \end{aligned}$$

das wir oben verwendet haben, in Ordnung ist.

(7.2)* Überlegt euch beim Würfelbeispiel, wie die Ereignisse „gerade Zahl" und „kleiner als zwei" sowie „kleiner als drei" aussehen. Berechnet die bedingten Wahrscheinlichkeiten, die sich aus der gegenseitigen Bedingung auf diese drei Ereignisse ergeben. Denkt euch weitere Aufgaben dieser Art aus und löst sie.

(7.3)* Überlegt euch Wortpaare, bei denen ihr die Ereignisse „A bzw. B kommt in einem zufällig gezogenen Dokument im Netz vor" für abhängig oder unabhängig halten würdet. Benutzt Google (oder einen Suchmaschine eurer Wahl), um die relativen Häufigkeiten von A, B und ihrem gemeinsamen Vorkommen zu bestimmen. Unter der Annahme, dass Google wirklich den kompletten Stichprobenraum ausschöpft, könnt ihr $P(A)$, $P(B)$ und $P(A \cap B)$ ausrechnen. Wie sieht es hier mit der Unabhängigkeit aus? Bringt eure Beispiele ins Tutorium mit.

(7.4)* Nehmen wir an, in 1% der guten und 40% der Spam-Mails komme das Wort „click" vor Außerdem seien 10% der Mails gut und 90% Spam. Berechnet mit der Bayes-Formel, wie groß die Wahrscheinlichkeit ist, dass eine Mail, in der „click" steht, Spam ist. **(L)**

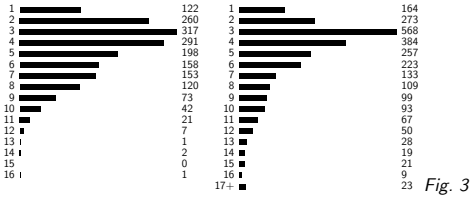

Fig. 3

8. Zufallsvariable

Leitfragen zu Abschnitt 8

- Warum definiert man Zufallsvariable? Wie ist ihr Zusammenhang mit Ergebnissen und Ereignissen?
- Wie wird der Begriff der Unabhängigkeit auf Zufallsvariable übertragen?
- Wo ist der „Margin“ in der Marginalverteilung?

Zufallsvariable

random variable

Verteilung

probability mass function

frequency function

Histogramm

Eine *Zufallsvariable* (*random variable*) ist eine Funktion $X\colon \Omega \to \mathbb{R}^n$.

Beim Würfeln kann die Zufallsvariable die Augenzahl sein, bei zwei Würfen etwa die Summe der Augenzahlen (so dass die Stichprobe $(1,3)$ auf 4 abgebildet wird) oder 0 für einen Pasch und 1 für einen Nicht-Pasch. Bei zufällig ausgewählten Personen könnten Zufallsvariablen das Alter, die Größe oder das Bruttoeinkommen sein, bei Wörtern ihre Länge, bei Zeichen ihr ASCII-Code, beim Münzwurf 0 für Blatt und 1 für Zahl.

Wir werden später das mit dem $\mathbb{R}^n$ nicht mehr so eng sehen und z.B. auch Wörter als Zufallsvariablen akzeptieren. Dann werden aber Begriffe wie Erwartungswert (nächste Folie) sinnlos, weswegen wir das vorerst mal nicht tun.

Schreibweise:

$$P(X = x) := P(\{\omega \in \Omega \mid X(\omega) = x\}).$$

Darstellung der *Verteilung* (*probability mass function*, pmf, *frequency function*) $P_X(x) = P(X = x)$ von Zufallsvariablen (die dadurch definierte Wahrscheinlichkeit dafür, dass bei einem Experiment die Zufallsvariable X den Wert x erhält, ist also einfach die Summe der Wahrscheinlichkeiten aller Stichproben ω, die von X auf x abgebildet werden) in *Histogramm*en, hier für die Wortlänge in einem englischen (links) und deutschen (rechts) Text:

(cf. Fig. 3)

Natürlich ist das nicht wirklich das Histogramm einer Zufallsvariablen. Diese muss nämlich aus einem Modell kommen und nicht aus einer Beobachtung von Häufigkeiten. Allerdings werden wir in Kürze endlich eine Verbindung zwischen den beiden Größen herstellen. Genau genommen sollten in das Histogramm auch die Werte von $P(X = x)$ eingetragen werden und nicht die rohen Zählungen. Dieser Defekt wäre allerdings durch schlichte Division durch die Gesamtzahl der Beobachtungen leicht heilbar.

Sind mehrere Zufallsvariablen $X_1, X_2, \ldots$ auf einem Wahrscheinlichkeitsraum definiert – etwa die Gesamt-Augenzahl und die Anzahl der Würfe mit Ergebnis sechs beim n-fachen Würfeln –, so kann man sie zu einer gemeinsamen Zufallsvariable

$$X(\omega) = \big(X_1(\omega), X_2(\omega), \ldots\big)$$

zusammenfassen (das Ergebnis ist also etwas wie ein Vektor).

Beispiel: Zweimaliges Würfeln. Zufallsvariablen: Augensumme, X_1, Augen im ersten Wurf, X_2; X_1 läuft waagrecht, X_2 läuft senkrecht.

	2	3	4	5	6	7	8	9	10	11	12	
1	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0	0	0	$\frac{1}{6}$
2	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0	0	$\frac{1}{6}$
3	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	0	$\frac{1}{6}$
4	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	0	$\frac{1}{6}$
5	0	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	0	$\frac{1}{6}$
6	0	0	0	0	0	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{6}$
	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$	

Marginalverteilung

Randverteilung

marginal distribution

unabhängig

In der Mitte der Tabelle ist die gemeinsame Verteilung von $X = (X_1, X_2)$, an den Rändern stehen die Marginalverteilungen $P(X_1)$ und $P(X_2)$. Dabei ist $P(X_1) = \sum_{X_2 \in \mathcal{X}_2} P(X_1, X_2)$ und analog für $P(X_2)$.

Die Verteilungen X_n heißen 1-dimensionale *Marginalverteilung* (*Randverteilung*, *marginal distribution*) – sie entstehen durch Addition über alle nicht betrachteten Zufallsvariablen. Entsprechend lassen sich mehrdimensionale Marginalverteilungen definieren, z.B. (X_2, X_3) über der Gesamtvariablen (X_1, X_2, X_3) durch Summation über X_1.

Unabhängigkeit für Zufallsvariablen

Zwei Zufallsvariablen sind *unabhängig*, wenn für alle x_1, x_2

$$P(X_1 = x_1, X_2 = x_2) = P(X_1 = x_1)P(X_2 = x_2)$$

gilt (und analog für mehrfache Produkte).

Eine mögliche Verteilung der Zufallsvariablen X_l „Wortlänge in einem deutschen Text“

X_l	1	2	3	4	5	6	7	8	9	10
$P(X_l)$	0.065	0.108	0.224	0.152	0.102	0.088	0.053	0.043	0.039	0.037
X_l	11	12	13	14	15	16	17	18	19	20
$P(X_l)$	0.026	0.020	0.011	0.008	0.008	0.004	0.002	0.004	0.002	0.007

Aufgaben

(8.1)* Überlegt euch Beispiele für Zufallsvariablen, die in der Sprachverarbeitung vorkomemn könnten: Wortlänge, Summe der ASCII-Codes, Zahl der Wörter in einem Dokument, die mit Suchanfrage übereinstimmen wären ein paar Vorschläge von mir. Gebt dazu jeweils den Stichprobenraum und soweit sinnvoll die Abbildung $\Omega \to \mathbb{R}$ an.

(8.2)* Erfindet eine Spielsprache mit vielleicht 10 Wörtern und fantasiert dazu eine Wahrscheinlichkeitsverteilung. Berechnet dann die Verteilung für die Zufallsvariablen Wortlänge und Zeichenindex (a=1, b=2...) des ersten Zeichens. Berechnet deren gemeinsame Verteilung. Sind die R.V.s unabhängig? Sollten sie es sein? Ratet, wie das wohl im Deutschen ist. Wenn ihr wollt, könnt ihr auch gerne ein Programm schreiben und aus einem größeren Korpus die beiden Verteilungen schätzen – allerdings sei warnend angemerkt, dass wir bisher nicht mal ordentlich schätzen können, geschweige denn wirklich testen, ob die beiden Verteilungen unabhängig sind.

9. Erwartungswert und Varianz

Erwartungswert

expectation

Funktional

Leitfragen zu Abschnitt 9

- Wie lassen sich Verteilungen charakterisieren?
- Was ist Linearität und warum macht sie das Leben bei Rechnungen viel leichter?
- Wie charakterisieren Erwartungswert, Varianz und Kovarianz eine oder mehrere Zufallsvariable?

Der *Erwartungswert* (*expectation*) einer Zufallsvariablen X ist definiert als

$$E(X) = \sum_{\omega\in\Omega} X(\omega)P(\omega).$$

Wenn $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ den Wertebereich von X bezeichnet, ist das gleichbedeutend mit

$$E(X) = \sum_{x\in\mathcal{X}} xP(X = x).$$

In der zweiten Form der Definition habe ich die Schreibweise

$$P(X = x) := \sum_{\omega\,|\,X(\omega)=x} P(\omega).$$

Damit ist auch klar, warum die beiden Formen äquivalent sind:

$$\begin{aligned}&\sum_{\omega\in\Omega} X(\omega)P(\omega)\\ =&x_1 \sum_{\omega\,|\,X(\omega)=x_1} P(\omega) + \dots + x_n \sum_{\omega\,|\,X(\omega)=x_n} P(\omega)\\ =&\sum_{x\in\mathcal{X}} xP(X = x).\end{aligned}$$

In Wirklichkeit haben wir etwas betrogen, weil wir uns auf endliche Wertebereiche beschränkt haben, aber das funktioniert auch sonst.

Anmerkungen:

1. Das entspricht einer Art mit den Wahrscheinlichkeiten gewichteten Mittelwert aller Werte, die die Zufallsvariable annehmen kann.
2. So etwas würde mit den Elementen von Ω zunächst nicht gehen, weil auf ihnen, selbst wenn wir sie 1, 2, 3 *nennen*, keine Arithmetik definiert ist. Deshalb kommt die Definition erst hier, obwohl „Mittelwert“ ein eigentlich sehr natürlicher Begriff sind.
3. Die Schreibweise $E(X)$ deutet an, dass E eine Funktion ist. Mit einem ausreichend allgemeinen Funktionsbegriff ist das auch so, im Allgemeinen aber bezeichnet man Funktionen, die Funktionen (hier das X) auf Zahlen abbilden, als *Funktionale*. Deshalb wird in der Literatur auch häufig EX oder $E[X]$ für $E(X)$ geschrieben. Uns braucht das nicht zu kümmern, solange wir im Kopf behalten, dass X eben keine Zahl ist.

Der Erwartungswert ist linear:

$$E(\lambda X) = \lambda E(X) \qquad E(X + Y) = E(X) + E(Y)$$

für ein $\lambda \in \mathbb{R}$.

Wenn X und Y unabhängig sind, ist $E(XY) = E(X)E(Y)$.

Varianz

Standardabweichung

Moment

zentrales Moment

Die zweite Behauptung folgt aus

$$\begin{aligned}E(XY) &= \sum_{\omega\in\Omega} X(\omega)Y(\omega)P(\omega)\\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} xyP(X = x, Y = y)\\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} xyP(X = x)P(Y = y)\\ &= E(X)\sum_{y\in\mathcal{Y}} yP(Y = y)\\ &= E(X)E(Y).\end{aligned}$$

Dabei haben wir im Schluss von der zweiten auf die dritte Zeile die Unabhängigkeit ausgenutzt und im Schluss von der dritten auf die vierte Zeile einfach ausgeklammert ($\sum_i xi = x\sum_i i$). Das geht hier, weil x_i und y_i unabhängig voneinander variieren und keine Abhängigkeit der einzelnen Faktoren besteht.

Trotzdem haben hier etwas betrogen. $\mathcal{X}$ ist nämlich auch bei uns häufig nicht endlich, und wenn die Summen über abzählbar unendliche Mengen ausgeführt werden, sind die Operationen, die wir hier machen, nicht unkritisch – in der Tat funktioniert das aber auch streng.

Schließlich: Die Schreibweise λX soll einfach die Zufallsvariable bezeichnen, in der jeder Wert mit λ multipliziert wird. Ist etwa der Wertebereich von X, „Augenzahl beim sechsseitigen Würfel“ $\{1, 2, \dots, 6\}$, ist der Wertebereich von $3X$ einfach $\{3, 6, 9, \dots, 18\}$.

Beispiel: Sechsseitiger Würfel, X ist „Augenzahl“:

$$E(X) = \sum_{i=1}^{6} \tfrac{1}{6} i = 1/6 + 2/6 + \dots = 21/6 = 3.5.$$

Diese Aussage werden wir bald als „Wenn wir wirklich oft würfeln, wird jeder Wurf im Mittel dreieinhalb Augen beitragen“ interpretieren können.

Nochmal Wortlängen X_l. Nach der Tabelle berechnet sich

$$E(X_l) = 1 \cdot 0.065 + 2 \cdot 0.108 + 3 \cdot 0.224 + \dots = 5.26.$$

Für einen englischen Text könnte $E(X_l) = 4.62$ sein.

Varianz

Die *Varianz* einer Zufallsvariable X ist definiert als

$$\begin{aligned}\mathsf{Var}(X) &= E\Big(\big(X - E(X)\big)^2\Big)\\ &= E(X^2) - E^2(X)\\ &= \sum_{x\in\mathcal{X}} (x - E(X))^2 P(X = x)\end{aligned}$$

Die Varianz ist nicht linear: $\mathsf{Var}(\lambda X) = \lambda^2\mathsf{Var}(X)$. Unter anderem deshalb arbeitet man gern mit ihrer Wurzel, der *Standardabweichung* $\sigma_X = \sqrt{\mathsf{Var}(X)}$.

Momente

Allgemein heißt $E(X^n)$ n-tes *Moment*, $E\Big(\big(X - E(X)\big)^n\Big)$ n-tes *zentrales Moment*. Der Erwartungswert ist also das erste Moment, die Varianz das zweite zentrale Moment.

Mehrere Variable, Kovarianz

Kovarianz

Korrelationskoeffizient

Wenn X und Y unabhängig sind, gilt $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

Das lässt sich einsehen, wenn man das Argument des Erwartungswerts ausrechnet, der für $\mathrm{Var}(X+Y)$ zu berechnen ist (dazu muss offenbar drei Mal die binomische Formel angewendet und die Linearität des Erwartungswerts genutzt werden):

$$\begin{aligned}&\Big((X+Y) - E(X+Y)\Big)^2\\ &= \Big(X^2 + 2XY + Y^2\Big) - 2\Big(XE(X) + Y(E(X) + XE(Y) + Y(E(Y)\Big)\\ &\qquad + \Big(E(X)^2 + 2E(X)E(Y) + E(Y)^2\Big)\\ &= \big(X - E(X)\big)^2 + \big(Y - E(Y)\big)^2 + 2XY - 2XE(Y) - 2YE(X) + 2E(X)E(Y).\end{aligned}$$

Der erste Teil der letzten Zeile ist im Erwartungswert schon das Ergebnis, der zweite Teil im Erwartungswert Null:

$$E\Big(2XY - 2XE(Y) - 2YE(X) + 2E(X)E(Y)\Big) = 2E(XY) - 4E(X)E(Y) + 2E(X)E(Y);$$

der erste Summand ist aber (und hier kommt die Unabhängigkeit) $2E(X)E(Y)$, der gesamte Ausdruck also Null.

Sind X, Y Zufallsvariablen über Ω, so heißt

$$\begin{aligned}\mathrm{Cov}(X,Y) &= E\Big(\big(X - E(X)\big)\big(Y - E(Y)\big)\Big)\\ &= \sum_{\omega\in\Omega}\Big(X(\omega) - E(X)\Big)\Big(Y(\omega) - E(Y)\Big)P(\omega)\end{aligned}$$

Kovarianz von X und Y. Sie ist ein Maß für die Ähnlichkeit der Verteilung. Als *Korrelationskoeffizient* wird die Größe

$$\rho_{XY} = \mathrm{Cov}(X,Y)/(\sigma_X\sigma_Y)$$

bezeichnet.

Die Kovarianz von unabhängigen Zufallsvariablen ist Null, denn mit X und Y sind auch $X - E(X)$ und $Y - E(Y)$ unabhängig, und daher ist

$$E\Big(\big((X - E(X)\big)\big((Y - E(Y)\big)\Big) = E\big((X - E(X)\big)E\big((Y - E(Y)\big).$$

Wegen der Linearität des Erwartungswertes ist weiter $E(X - E(X)) = E(X) - E(E(X)) = 0$. Die Kovarianz ist also ein Maß dafür, wie groß die Abhängigkeit von X und Y ist.

Aufgaben

(9.1)* Rechnet den Erwartungswert der Zufallsvariable „Zwei Würfelwürfe, Summe davon“ aus (Vorsicht: P_X ist keine Gleichverteilung).

(9.2) Weist nach, dass der Erwartungswert linear ist (d.h., dass die Definitionsgleichungen für Linearität für die spezielle Form des Erwartungswerts erfüllt sind. Macht euch klar, dass Linearität eine sehr spezielle Eigenschaft ist (z.B. indem ihr mal seht, ob x^2 oder $1/x$ linear sind.

(9.3)* Wenn X die Ordnungszahl des ersten Zeichens eines Wortes ist, was ist dann die Interpretation für $E(X)$? Was könnte man aus $E(X) < 14$ schließen?

(9.4)* Rechnet die Varianz der Zufallsvariablen „Augenzahl eines sechsseitigen Würfels“ aus und macht euch klar, dass dieser Wert nicht linear ist (z.B. indem ihr annehmt, dass jemand die Augen beim Würfel einfach verdoppelt hat und jetzt 2, 4,... herauskommt).

(9.5) Rechnet die Kovarianz der beiden Zufallsvariablen im Babysprachenbeispiel aus.

Leitfragen zu Abschnitt 10

Binomialverteilung

i.i.d.

- Was ist die Binomialverteilung und warum taucht sie häufig auf?
- Was ist die Normalverteilung und warum taucht sie häufig auf?

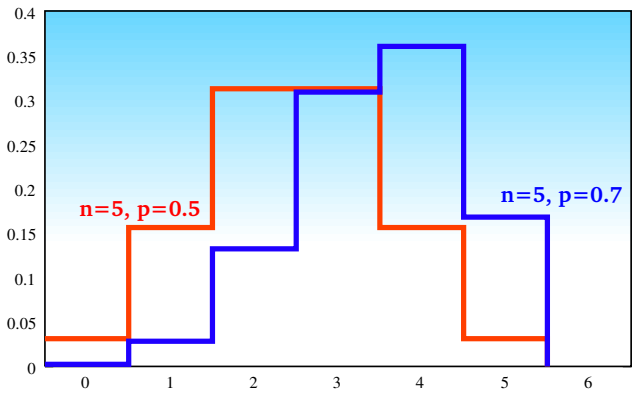

Fig. 4

10. Häufig auftretende Verteilungen

Binomialverteilung

Sei $\Omega = \{u, d\}$ der Stichprobenraum eines Bernoulli-Experiments, sei $P(u) = p$ und $P(d) = q = 1 - p$. Wir betrachten den Stichprobenraum $\tilde\Omega = \bigotimes_{i=1}^{n}\Omega$, also etwa n Münzwürfe. $\bigotimes$ soll dabei für ein mehrfaches kartesisches Produkt stehen, ganz, wie $\sum$ für eine mehrfache Summe steht. Wir haben also einfach ein Produktexperiment von Bernoulliexperimenten. Auf diesem Raum betrachten wir X, „Zahl der u in $\tilde\omega \in \tilde\Omega$“. Die Verteilung von X heißt *Binomialverteilung* (oder Bernoulli-Verteilung):

$$P(X = k) = b(k; n, p) := \binom{n}{k}p^k q^{n-k}.$$

Das lässt sich so einsehen: Jedes Element des Ereignisses $A =$„genau k-mal u in $\tilde\omega$“ hat die Wahrscheinlichkeit $p^k q^{n-k}$. Es bleibt die Frage nach der Kardinalität dieses Ereignisses. Effektiv ziehen wir k Positionen der u ohne Zurücklegen aus n Kugeln, wobei die Reihenfolge keine Rolle spielt, was unserem Modell Ω_{III} entspricht. Damit ist $|A| = \binom{n}{k}$

Es gilt $E(X) = np$ und $\mathrm{Var}(X) = npq$. $E(X)$ für $n = 1$ ist natürlich p, wegen der Linearität des Erwartungswerts summiert sich das einfach n Mal. $\mathrm{Var}(X)$ für $n = 1$ ist

$$E(X^2) - E(X)^2 = p\cdot 1^2 + q\cdot 0^2 - p^2 = p(1-p) = pq;$$

hier addieren sich die individuellen Varianzen, weil die X alle unabhängig sind. Verteilungen dieser Art treten immer auf, wenn unabhängige, identisch verteilte Experimente (independent identically distributed, *i.i.d.*) hintereinander ausgeführt werden.

(cf. Fig. 4)

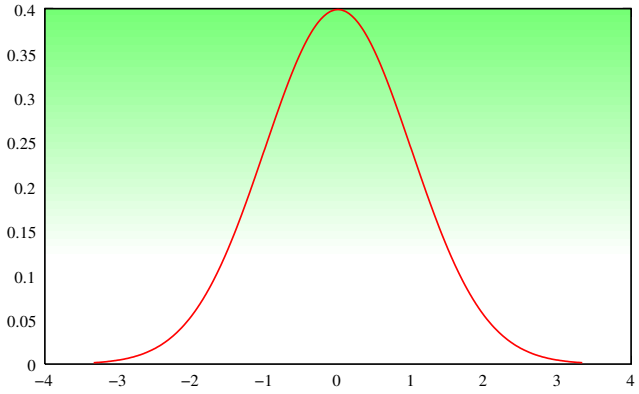

Fig. 5

Standard-Normal-verteilung

Zentrale Grenzwert-satz

Verteilungsfunktion

Dichte

Normalverteilung

Die *Standard-Normalverteilung* ist definiert als

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2).$$

Es handelt sich hier um eine Dichte, x ist reell. Mit solchen Dichten können wir eigentlich nicht umgehen, weil unsere Wahrscheinlichkeitsräume und damit auch Zufallsvariablen immer diskret waren und das auch bleiben sollen. Wir ignorieren diese Probleme so gut wie möglich.

Diese „Gaußkurve“ sieht so aus:

(cf. Fig. 5)

Es gilt der *Zentrale Grenzwertsatz*: Seien X_i, $i = 1, \ldots, n$ i.i.d. Zufallsvariable mit $E(X_i) = 0$ und $\mathrm{Var}(X_i) = 1$. Sei $S_n = \left(\sum_{i=0}^{n} X_i\right)/\sqrt{n}$. Dann gilt

$$\sup_{x \in \mathcal{X}} \left| P(S_n < x) - \int_{-\infty}^{x} \varphi(t)\, dt \right| \to 0 \quad \text{für } n \to \infty.$$

Der Beweis ist relativ schwierig. Die Einschränkung für die Zufallsvariablen ist nicht wichtig, man sie kann (fast) immer durch Einfühung einer neuen Zufallsvariablen erzwingen, etwa durch Subtraktion des Erwartungswerts und Division durch Standardabweichung. Damit besagt dieser Satz im Wesentlichen, dass alle Experimente, wenn ich sie nur oft genug hintereinander ausführe, schließlich auf normalverteilte Zufallsvariablen führen werden.

Vorsicht: Der Satz macht keine Aussage darüber, wie schnell das geht. Solange n „klein“ ist (und „klein“ kann durchaus noch drei Dezillionen sein), kann die tatsächliche Verteilung von der Normalverteilung beliebig abweichen.

Die Funktion $\int_{-\infty}^{x} \varphi(t)\, dt$, die angibt, wie viel „Wahrscheinlichkeit“ unterhalb von x liegt, heißt *Verteilungsfunktion*. Verteilungsfunktionen sind vor allem wichtig, wenn man mit kontinuierlichen Verteilungen arbeitet; die Verteilung (hier auch *Dichte*) ergibt sich als Ableitung der Verteilungsfunktion.

Standard-Normalverteilung heißt die oben vorgestellte Verteilung, weil sie Erwartungswert Null und Varianz 1 hat. Normalverteilungen zu beliebigen Erwartungswerten μ und Varianzen σ^2 werden durch die Funktionen

$$\varphi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\,\sigma^2}\right)$$

dargestellt.

tschebytschewsche Ungleichung

Aufgaben

(10.1)* Seht nach, ob ihr aus einer Folge unabhängiger Bernoulli-Experimente wirklich eine Binomialverteilung erhaltet. Das Bernoulli-Experiment soll das Werfen einer Münze sein. Werft sie vier Mal und zählt, wie oft „Zahl“ kommt. Wiederholt das fünf Mal. Kombiniert eure Ergebnisse mit denen eurer Mitstudis. Wie gut passt euer Gesamtergebnis auf die für dieses Experiment zu erwartende Binomialverteilung? Wie gut auf eine Gaußverteilung, die diese Binomialverteilung annähert?

(10.2)* Holt euch von der Webseite zur Vorlesung das Programm `gaussBinomi.py`, das die Binomialverteilung für verschiedene Parameter mit der ihr entsprechenden Gaußverteilung vergleicht.

Lasst das Programm laufen und verändert systematisch die Parameter der Bernoulli-Verteilung. Beobachtet, wie sich die entsprechende Gauss-Verteilung anpasst und wie gut die Übereinstimmung ist. Das Programm wird mit folgenden Tasten bedient:

- 8/2 - N erhöhen/reduzieren.
- 6/4 - p erhöhen/reduzieren
- z/Z: Zoom erhöhen/reduzieren (i.e. Bereich um 0 genauer ansehen)

(10.3)* Macht euch klar, warum die Gaussverteilung, die wir hier verwenden, die „zu erwartende“ für die entsprechende Bernoulliverteilung ist. Die Antwort dazu steht in `GaussVsBernComparer.setParameters`

(10.4) Beobachtet genau, was für kleine p und nicht zu große n um die Null rum passiert. Im Allgemeinen wird das Kriterium, ab wann Gauss eine gute Näherung für Bernoulli ist, in Abhängigkeit des Produkts np gegeben. Wo würdet ihr die Grenze setzen?

Anmerkung: Das rücksichtslose (und noch dazu schreibende) Reinpfuschen in die Attribute von Klassen, wie ich das hier bei den Funktionen mache, ist in der Realität natürlich verboten und hier nur durch die Faulheit des Lehrenden entschuldigbar.

(10.5) Sei X die Zufallsvariable „Ordnungszahl des ersten Zeichens eines Wortes“. Wir ziehen jetzt die Wörter eines Textes in ihrer Reihenfolge und rechnen $P(X = 1)$ (also „Das erste Zeichen ist ein a“) aus. Wird das eine Bernoulliverteilung sein? Warum (nicht)?

Datei(en) im PDF-Anhang: gaussBinomi.py

11. Das schwache Gesetz der großen Zahlen

Leitfragen zu Abschnitt 11

- Wie bekommt man eine Verbindung zwischen der Verteilung einer Zufallsvariablen und den Ergebnissen von Zufallsexperimenten hin?
- Was ist der Unterschied zwischen einem Mittelwert und einem Erwartungswert?
- Warum sind große Samples so wichtig?

Die *tschebytschewsche Ungleichung* für eine Zufallsvariable X lautet

$$P\left(|X - E(X)| \geq \epsilon\right) \leq \mathrm{Var}(X)/\epsilon^2.$$

Beweis: Sei $Z = X - E(X)$, also einfach X, nur auf Erwartungswert Null getrimmt. Definiere neue Zufallsvariable Y mit

$$Y(\omega) = \begin{cases} 0 & |Z(\omega)| < \epsilon \\ \epsilon^2 & |Z(\omega)| > \epsilon \end{cases}$$

Das sieht zunächst komisch aus, hat aber den Vorteil, dass wir einerseits sicher wissen, dass $Y \leq |Z|^2$ (und das werden wir in der Abschätzung brauchen), wir aber den Erwartungswert von Y gut ausrechenen können (wir haben nämlich de facto ein Bernoulliexperiment). Konkret haben wir mit diesem Trick erreicht:

$$\mathrm{Var}(X) = E(|Z|^2) \geq E(Y) = \epsilon^2 P\left(|X - E(X)| \geq \epsilon\right)$$

Große Abweichungen vom Erwartungswert werden um so unwahrscheinlicher, je kleiner die Varianz ist.

Wichtige Folge: Das „schwache Gesetz der großen Zahlen“.

Seien $X_1, \ldots, X_n$ i.i.d. Zufallsvariable mit beschränkter Varianz. Sei $H_n = 1/n \sum_{i=1}^{n} X_i$. Es gilt mit $\epsilon > 0$ und einer Konstanten M:

$$P\left(|H_n - E(X_1)| \geq \epsilon\right) \leq \frac{M}{\epsilon^2 n} \to 0 \quad \text{für } n \to \infty$$

.

Beweis: Wegen der Linearität des Erwartungswerts und unserer i.i.d.-Annahme ist $E(H_n) = E(X_1)$. Außerdem ist

$$\begin{aligned}\mathsf{Var}(H_n) &= n^{-2}\mathsf{Var}\left(\sum X_i\right) \\ &= n^{-2}\left(\sum \mathsf{Var}(X_i)\right) \\ &= n\mathsf{Var}(X_1)/n^2.\end{aligned}$$

Mit Tschebytschew ist nun

$$\begin{aligned}P(|H_n - E(H_n)| \geq \epsilon) &= P(|H_n - E(X_1)| \geq \epsilon) \\ &\leq \mathsf{Var}(H_n)/\epsilon^2 \\ &= \frac{\mathsf{Var}(X_1)}{n\epsilon^2.}\end{aligned}$$

Das ist mit $M = \mathsf{Var}(X_1)$ der eine Teil der Behauptung. Der andere folgt, weil M und ϵ konstant sind und $1/n$ für große n gegen Null geht.

Die Aussage ist also, dass bei vielen identischen Experimenten die Wahrscheinlichkeit, dass die Summe der Ergebnisse geteilt durch die Zahl der Ergebnisse (der *Mittelwert* einer Meßreihe) weit vom Erwartungswert abliegt, beliebig klein wird.

Tatsächlich ist die Abschätzung aus dem Gesetz der großen Zahlen bzw. aus der tschebytschewschen Ungleichung in aller Regel zu pessimistisch, gibt also relativ große Grenzen für die Wahrscheinlichkeit an, die normalerweise bei weitem nicht erreicht werden. Der Grund dafür ist die große Allgemeinheit von Tschebytschew, der sozusagen noch für die übelsten Verteilungen (etwa mit zwei ganz spitzen Maxima) gelten muss, während die meisten Verteilungen, denen man im Alltag so begegnet, viel netter sind. Wie man „bessere“ (man sagt auch gern „schärfere“) Abschätzungen bekommt, werden wir noch sehen.

Diese Formel liefert den Zusammenhang zwischen experimentell ermittelbaren relativen Häufigkeiten und einem Parameter der Verteilung.

Beispiel: Ω seien die Wörter eines deutschen Textes, die Zufallsvariable ist die Länge eines „zufällig“ gezogenen Worts. Es kann dann sein:

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
11	4	13	4	7	8	9	6	6	6

Sei von irgendwo bekannt, dass $\mathsf{Var}(X) = 12$ (also die Wortlänge in irgendeinem Sinn typisch um $\sqrt{12} \approx 3.5$ vom Erwartungswert abweicht).

Es ist $H_4 = 8$, die Wahrscheinlichkeit, dass die tatsächliche mittlere Wortlänge des Deutschen um zwei oder mehr (also $\epsilon = 2$) von 8 abweicht, also höchstens $12/2^2/4 = 0.75$ (die Abschätzung ist nicht sehr gut…). Die Wahrscheinlichkeit, dass sie um 5 oder mehr von 8 abweicht, ist höchstens $12/5^2/4 = 0.12$

Es ist $H_{10} = 7.4$. Abweichung um 5 oder mehr mit höchstens 0.048, Abweichung um 1 oder mehr mit höchstens 1.2.

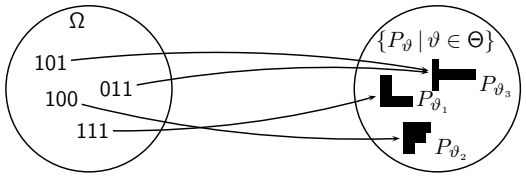

Fig. 6

Aufgaben

(11.1)* Im Anhang dieser Seite befindet sich ein Programm, dem ihr Erwartungswert und Varianz einer Zufallsvariable sowie eine Zahl n der Experimente vorgeben könnt und das dann einige Male diese Zahl gerade n Werte der Zufallszahlen zieht. Das Programm rechnet dann H_n und seine Abweichung vom Erwartungswert aus und gibt am Schluss die kumulierten relativen Häufigkeiten bestimmter Abweichungen aus. Ein Beispiellauf:

```
examples> python tschegame.py 1 20 400

Relative Häufigkeit |H_n-E(X)|>=3: 0.005000
Relative Häufigkeit |H_n-E(X)|>=2: 0.055000
Relative Häufigkeit |H_n-E(X)|>=1: 0.330000
Relative Häufigkeit |H_n-E(X)|>=0: 1.000000
```

Seht euch das Programm an und macht euch klar, dass es letztlich versucht, das schwache Gesetz der großen Zahlen (sGgZ) experimentell zu prüfen.

Führt diese Prüfung durch, d.h. vergleicht die Vorhersagen aus dem sGgZ mit dem, was das Programm findet. Tut das für verschiedene Werte des Erwartungswerts (im ersten Kommandozeilenargument), der Varianz (im zweiten) und dem n aus dem H_n (im dritten) – für diesen Vergleich müsst ihr schon annehmen, dass die relativen Häufigkeiten, die das Programm ausgibt, tatsächlich gute Schätzungen für die Wahrscheinlichkeiten aus dem sGgZ sind – wenn ihr wollt, könnt ihr mit dem sGgZ selbst schon nachsehen, wie groß die Wahrscheinlichkeit ist, dass das Programm sich arg irrt…

Was stellt ihr fest? Warum ist das kein Widerspruch zum sGgZ? Könnt ihr euch vorstellen, wie bei so einer Abschätzung von Wahrscheinlichkeiten anch oben der Begriff „scharf“ zu verstehen wäre?

(11.2)* Wenn wir glauben, die Varianz der Zufallsvariable X „Länge eines Wortes“ sei im Deutschen tatsächlich 12: Wie viele Wörter muss ich ansehen, damit mir schon das Gesetz der großen Zahl garantiert, dass meine $E(X)$-Schätzung nur mit, sagen wir, 10%, 5%, 1% um 1, 0.5, 0.1 (oder natürlich jeweils mehr) danebenliegt?

(11.3) Über welchem Wahrscheinlichkeitsraum sind unsere H_n aus dem Beweis des schwachen Gesetzes der großen Zahl eine Zufallsvariable?

Datei(en) im PDF-Anhang: chegame.py

12. Schätztheorie I

Leitfragen zu Abschnitt 12

- Was ist schätzen und warum macht man sich darüber Sorgen?
- Warum taugt der normale Stichprobenraum nicht als Basis der Schätztheorie und wodurch wird er ersetzt?
- Was ist die Idee hinter Maximum Likelihood? Was ist davon zu halten?

Mit dem schwachen Gesetz der großen Zahlen kann man offenbar Erwartungswerte aus Beobachtungen „kontrolliert schätzen“. Geht das auch für Varianz oder andere Parameter?

Dafür *Schätztheorie* mit

- einem Stichprobenraum Ω
- einer Familie von Verteilungen $\{P_\vartheta \mid \vartheta \in \Theta\}$
- einem zu schätzenden Parameter ϑ

(cf. Fig. 6)

In der Schätztheorie ist Ω im Allgemeinen ein Produkt aus n unabhängigen Einzelexperimenten, während ϑ ein Parameter der Verteilung der Einzelexperimente ist.

Beispiel: Schätze die Wahrscheinlichkeit, dass ein zufällig gezogenes Wort länger als 5 Zeichen ist. Der Stichprobenraum ist $\{0,1\}^n$ (1 für „Wort länger als 5“). Wir haben ein Bernoulli-Experiment, unsere Familie von W-Maßen ist also

$$\{P_\vartheta\} = \left\{ \binom{n}{x} p^x (1-p)^{n-x} \mid \vartheta \in [0,1] \right\}.$$

Wir haben dabei ganz flott eine Zufallsvariable X eingeführt, die Zahl der Einsen in ω; x ist ihr Wert beim vorliegenden Experiment.

Wir wollen p schätzen, also ist $\vartheta = p$.

Das Ganze ließe sich noch etwas verallgemeinern durch Einführung einer Funktion $g\colon \Theta \to \mathcal{Y}$, so dass man direkt nicht nur den Parameter ϑ, sondern gleich funktional davon abhängige Größen schätzen kann. Das allerdings verkompliziert die Notation und bringt wenig tiefere Einsicht, weshalb wir hier darauf verzichten.

Der *Maximum-Likelihood-Schätzer* für ϑ ist eine Abbildung $\hat{\vartheta}\colon \Omega \to \Theta$, so dass die *Likelihood-Funktion* $L_\omega(\vartheta) = P_\vartheta(\omega)$ für $\hat{\vartheta}(\omega)$ maximal wird.

Generell heißt jede Abbildung $\hat{\vartheta} : \Omega \to \Theta$ *Schätzer* (*estimator*) von ϑ – darin sind natürlich auch Schätzer, die völligen Quatsch schätzen.

Noch einmal: ϑ ist einfach ein Parameter, $\hat{\vartheta}\colon \Omega \to \Theta$ eine Funktion, $\hat{\vartheta}(\omega)$ der konkrete *Schätzwert*, der aus der Beobachtung der Stichprobe ω folgt, also das ϑ, das wir als in irgendeinem Sinne „besten“ Parameter zur Erkärung der Beobachtung ansehen.

Der Maximum-Likelihood-Schätzer ist so gebaut, dass er den Parameter $\vartheta = \vartheta_0$ wählt, für den *bei der konkreten Beobachtung* ω die Ungleichung $P_{\vartheta_0}(\omega) \geq P_\vartheta(\omega)$ für alle $\vartheta \in \Theta$ gilt – man maximiert also die Wahrscheinlichkeit für das Auftreten von ω durch Wahl derjenigen Verteilung, in der ω gerade die höchste Wahrscheinlichkeit hat.

Das ist absolut nicht zwingend – es wäre etwa auch denkbar, dass wir ein paar „benachbarte“ $P_\vartheta(\omega)$ zusammenzählen und diese Größe maximieren. Oder wir misstrauen unserer Beobachtung in irgendeiner systematischen Weise und bauen dieses Misstrauen in unseren Schätzer ein.

Im Beispiel ist die Likelihood-Funktion

$$L_x(p) = \binom{n}{x} p^x (1-p)^{n-x}.$$

Das Maximum dieser Funktion bestimmt man am Besten nach Logarithmierung (dann werden die hässlich abzuleitenden Produkte zu einfach abzuleitenden Summen, Maxima bleiben aber Maxima). Wir müssen sehen, wo die Ableitung nach p Null wird. Also:

$$\frac{x}{p} - \frac{n-x}{1-p} = 0.$$

Eine Lösung davon ist $p = x/n$. In der Tat ist dort ein Maximum, also ist der ML-Schätzer für unser Problem $\hat{\vartheta}(\omega) = x/n$ mit x der Zahl der Wörter länger als fünf und n der Zahl der Wörter insgesamt.

Das hätte einem natürlich schon der gesunde Menschenverstand gesagt, in der Realität sind die Probleme aber nicht so einfach, und häufig versagt der gesunde Menschenverstand.

Für eine Zufallsvariable $T\colon \Omega \to \mathbb{R}$ ist $E_\vartheta(T) = \sum_{\omega\in\Omega} T(\omega) P_\vartheta(\omega)$ der Erwartungswert von T bezüglich P_ϑ. Ein Schätzer $\hat{\vartheta}$ heißt *erwartungstreu* (*unbiased*), wenn

$$E_\vartheta(\hat{\vartheta}) = \vartheta$$

für alle ϑ. Der *Bias* von $\hat{\vartheta}$ ist

$$b(\vartheta, \hat{\vartheta}) = E_\vartheta(\hat{\vartheta}) - \vartheta.$$

Maximum-Likelihood-Schätzer
Likelihood-Funktion
Schätzer
estimator
Schätzwert
erwartungstreu
unbiased
Bias

Was bedeutet Erwartungstreue? So, wie wir unsere Schätztheorie aufgezogen haben, behaupten wir, es gebe ein „wahres“ ϑ, das von einem konkreten System realisiert wird. Die Zufallsvariable, mit der wir schätzen, `ist` dann eben gemäß ϑ verteilt. In unserem Beispiel oben wird ein Schätzer nun zu jedem möglichen $x \in \{0, 1, \ldots n\}$ ein $\hat{\vartheta}(x) = x/n$ liefern, also mitnichten jedes Mal das wahre ϑ – wenns so einfach wäre, bräuchten wir keine Schätztheorie.

Ein erwartungstreuer Schätzer $\hat{\vartheta}$ liefert aber *im Mittel* das richtige ϑ. Warum? Wenn wir das Experiment ganz oft (sagen wir, N Mal) mit unabhängigen Samples, die alle gemäß ϑ verteilt sind, wiederholen, sollte nach dem Gesetz der großen Zahlen mit $\hat{\vartheta}$ als Zufallsvariable

$$P(|H_N - E_\vartheta(\hat{\vartheta})| \leq \epsilon) \leq M/(\epsilon^2 N)$$

gelten, $H_N = 1/N \sum_{i=1}^N \hat{\vartheta}(x_i)$ also schließlich fast immer sehr nah an $E_\vartheta(\hat{\vartheta})$ liegen. H_N ist aber gerade die Schätzung, die wir aus einer langen Versuchsreihe ableiten, während $E(\hat{\vartheta})$ bei einem erwartungstreuen Schätzer gerade ϑ ist – anders gesagt: wenn wir so lange experimentieren, dass wir jedes x aus dem Wertebereich von X tatsächlich mit $NP_\vartheta(X = x)$ Mal beobachten, liefert ein erwartungstreuer Schätzer wirklich den wahren Wert.

Ist unser ML-Schätzer erwartungstreu? Rechnen wir nach:

$$\begin{aligned} E_\vartheta(\hat{\vartheta}) &= \sum_{x=0}^{n} \frac{x}{n} \binom{n}{x} \vartheta^x (1-\vartheta)^{n-x} \\ &= \frac{1}{n} n\vartheta, \end{aligned}$$

weil der zu berechnende Term (in der Summe steht einfach der Schätzer x/n und die Verteilung P_ϑ) gerade der Erwartungswert der Bernoulli-verteilten Zufallsvariablen $X = x$ ist, den wir oben zu np berechnet hatten. Demnach ist unser Schätzer erwartungstreu.

Haben wir einen nicht-erwartungstreuen Schätzer, ziehen wir offenbar etwas aus den Daten, das zunächst nicht drinsteht – es weicht eben gerade um den Bias davon ab. Hat sowas überhaupt einen Sinn? Ja, denn erwartungstreue Schätzer sind zwar dann gut, wenn man Ω oder analog den Wertebereich von X im erwähnten Sinn „ausschöpfen“ kann. Wenn wir aber wissen, dass das garantiert nicht passiert, ist auch klar, dass ein erwartungstreuer Schätzer garantiert ein falsches Ergebnis liefert.

Wir könnten beispielsweise die Wahrscheinlichkeiten schätzen, mit denen bestimmte Wörter auf auf das Wort „im“ folgen: im Haus, im Hörsaal, im Neckar usw. Wir *wissen*, dass es auch mal „im Rhein“ sein kann, selbst wenn wir das nicht sehen – nicht-erwartungstreue (und nicht-ML) Schätzer erlauben uns, für sowas Platz zu reservieren. In diesem Fall haben wir ein (begründetes) „Vorurteil“ (eben ein bias) über die Qualität unserer Daten, und das sollten wir besser in unseren Schätzer einbauen, wenn wir brauchbare Ergebnisse erhalten wollen.

Der bias hängt übrigens offenbar von ϑ ab. Das ist auch gut so, denn es mag durchaus sein, dass man z.B. für große ϑ einen kleinen Bias haben möchte und für kleine einen größeren. Wenn wir im obigen Beispiel bei 1000 Bigrammen mit „im“ sagen wir 30 Mal „im Haus“ finden und nur 1 Mal „im Neckar“, würden wir 30/1000 wohl als eine ganz brauchbare Schätzung für die Wahrscheinlichkeit von „im Haus“ ansehen, während wir nicht sicher sein können, ob „im Neckar“ nicht in Wirklichkeit sehr selten sein sollte, wir aber zufällig einen Text erwischt haben, der über Heidelberg redet (was auch schon andeutet, dass der Stichprobenraum bei Schätzproblemen eminent wichtig ist – P(„im Neckar“) hat, wenn der Stichprobenraum „alle möglichen Texte über Heidelberg“ heißt, einen ganz anderen Wert als über dem Stichprobenraum „Texte, die sich mit Wüsten beschäftigen“). Entsprechend würden wir das 1/1000 von „im Neckar“ wohl deutlich stärker bestrafen als das 30/1000 von „im Haus“.

Ein populärer nicht-erwartungstreuer Schätzer für Häufigkeiten dieser Art ist Expected Likelihood Expectation (ELE, auch Jeffreys-Perks-Gesetz, hier für Bigramme formuliert):

$$P(w_1 w_2) = \frac{|w_1 w_2| + 1/2}{N + B/2}$$

Darin ist $|w_1w_2|$ die absolute Häufigkeit des Bigramms w_1w_2, N die Zahl der beobachteten Bigramme (quasi der word tokens) und B die Zahl der *verschiedenen* möglichen Bigramme (also quasi der word types) – es ist eingestandenermaßen nicht immer offensichtlich, wie viele das wohl sein werden.

So, wie das gemacht ist, wird also im Zähler ein bisschen was addiert, im Nenner ein bisschen mehr, so dass die Wahrscheinlichkeiten kleiner werden, und zwar um so mehr, je kleiner die relative Frequenz von w_1w_2 ist. Nur haben eben auch ungesehene word types eine finite Wahrscheinlichkeit (nämlich $1/(2N + B)$). Diese Formel lässt sich übrigens sogar begründen als in irgendeinem Sinn optimale Mischung zwischen einer ML-Schätzung und einer Bayesianischen Schätzung mit gleichverteiltem Prior (dazu später mehr).

Der *mittlere quadratische Fehler* eines Schätzers ist

$$\begin{aligned} R(\vartheta, \hat{\vartheta}) &= E_\vartheta\Big((\hat{\vartheta} - \vartheta)^2\Big) \\ &= \mathsf{Var}_\vartheta(\hat{\vartheta}) + b(\vartheta, \hat{\vartheta})^2. \end{aligned}$$

Den zweiten Teil der Gleichung sieht man so ein:

$$\hat{\vartheta} - \vartheta = \Big(\hat{\vartheta} - E_\vartheta(\hat{\vartheta})\Big) - \Big(\vartheta - E_\vartheta(\hat{\vartheta})\Big).$$

Das kann man quadrieren und ausmultiplizieren. Danach steht unter Ausnutzung der Linearität des Erwartungswerts das Ergebnis fast schon da – problematisch bleibt nur das gemischte Glied, das nach Ausmulitplizieren

$$E_\vartheta(\vartheta\hat{\vartheta}) - E_\vartheta\big(\hat{\vartheta}E_\vartheta(\hat{\vartheta})\big) - E_\vartheta\big(\vartheta E_\vartheta(\hat{\vartheta})\big) + E_\vartheta(\hat{\vartheta}^2)$$

lautet.

Nun sind die Erwartungswerte in den Klammern Konstanten und ziehen sich vor die Klammer, also heben sich die Terme 2 und 4 weg. Außerdem ist ϑ für die Berechnung von E_ϑ ebenfalls konstant und kann aus dem Erwartungswert herausgezogen werden, also heben sich auch die Terme 1 und 3 weg.

Ein Streben nach möglichst kleinen mittleren quadratischen Fehlern ist übrigens ein weiterer Grund, von der Erwartungstreue Abstand zu nehmen. Am Ende des Tages interessiert man sich bei einem Schätzer eben doch eher dafür, wie sehr er sich bei realen Experimenten verschätzt als dafür, ob er, wenn man Ω koplett abgrasen würde und alle Ergebnisse mit den nach ihren Wahrscheinlichkeiten zu erwartenden Frequenzen erhalten hat, das wahre und endgültige ϑ liefern würde – eben weil dies meist unmöglich ist. Auf der letzten Folie zum Good-Turing-Schätzer werden beispielsweise wir einen Graphen sehen, der zeigt, dass für ein recht typisches Problem der nicht-erwartungstreue Good-Turing-Schätzer durchweg erheblich geringere mittlere quadratische Fehler aufweist.

Ein erwartungstreuer Schätzer für den Erwartungswert einer Zufallsvariablen X, die in n unabhängingen Experimenten mit Ausgängen X_k beobachtet wird, ist der *Mittelwert*

$$\bar{X} = \frac{1}{n}\sum_{k=1}^{n} X_k,$$

für das Quadrat ihrer Standardabweichung s^2

$$s^2 = \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \bar{X})^2.$$

Warum sind diese Schätzer erwartungstreu? Es ist

$$E_\vartheta(\bar{X}) = \frac{1}{n}\sum_{k=1}^{n} E_\vartheta(X_k) = \frac{1}{n} n E_\vartheta(X_1).$$

Wir brauchen dabei weder ϑ noch die Familien $P_{\vartheta\in\Theta}$ festzulegen – die Formel gilt unabhängig von der konkreten Modellierung, interessant ist nur der Erwartungswert. Dies ist einer der Fälle,

mittlere quadratische Fehler
Mittelwert

in denen man dringend *nicht* die Parameter der Verteilungen (weil wir gar keine Verteilungen annehmen), sondern stattdessen direkt eine Funktion des Parameters (nämlich den Erwartungswert) schätzen möchte. Dieses formale Detail soll uns jetzt nicht stören.

Das $n - 1$ im Zähler des Schätzers für s^2 wird häufig als mystisch deklariert. In der Tat führt diese Wahl aber zu einem erwartungstreuen Schätzer:

$$E_\vartheta(s^2) = \frac{1}{n-1}\sum_{k=0}^{n} E_\vartheta\big((X_k - \bar{X})^2\big).$$

Wir rechnen die Summanden aus und bezeichnen $\mu = E_\vartheta(X_i)$:

$$\begin{aligned} E_\vartheta\big((X_k - \bar{X})^2\big) &= E_\vartheta\Big(\big((X_i - \mu) - (\bar{X} - \mu)\big)^2\Big) \\ &= E_\vartheta\big((X_i - \mu)^2\big) + E_\vartheta\big((\bar{X} - \mu)^2\big) - 2E_\vartheta\big((X_i - \mu)(\bar{X} - \mu)\big). \end{aligned}$$

Der erste Summand in diesem Ausdruck ist per definitionem s^2. Der zweite Summand lässt sich folgendermaßen berechnen:

$$E_\vartheta\big((\bar{X} - \mu)^2\big) = \mathsf{Var}_\vartheta(\bar{X}) = \mathsf{Var}_\vartheta\left(\frac{1}{n}\sum_{k=1}^{n} X_k\right) = \frac{1}{n^2}\sum_{k=1}^{n}\mathsf{Var}_\vartheta(X_k) = \frac{s^2}{n}.$$

Bleibt der letzte Summand. Es ist

$$\begin{aligned} E_\vartheta\big((X_i - \mu)(\bar{X} - \mu)\big) &= E_\vartheta\left((X_i - \mu)\left(\frac{1}{n}\sum_{k=0}^{n}(\bar{X} - \mu)\right)\right) \\ &= \frac{1}{n}\sum_{k=0}^{n} E_\vartheta\big((X_i - \mu)(X_k - \mu)\big). \end{aligned}$$

Der Erwartungswert in der letzten Zeile ist gerade die Kovarianz. Da die verschiedenen Experimente unkorreliert sind, ist diese Null, wenn $i \neq k$. Also schnurrt die Summe zusammen auf den Term $E_\vartheta\big((X_i - \mu)^2\big)$ und damit auf s^2.

Zusammen ist

$$E_\vartheta(s^2) = \frac{1}{n-1}\sum_{k=0}^{n}\left(s^2 + \frac{s^2}{n} - \frac{2s^2}{n}\right) = \frac{1}{n-1}\sum_{k=0}^{n}\frac{s^2(n-1)}{n} = s^2$$

Aufgaben

(12.1)* Besorgt euch von der Webseite das Programm empBias.py. Ihr findet darin zwei Schätzer, estELE und estML. Überzeugt euch, dass diese in der Tat die die ELE- und ML-Schätzer implementieren (das B-Argument in estML dient nur dazu, dass estML und estELE auf die gleiche Weise aufgerufen werden können). Werft einen kurzen Blick über die Hilfsfunktionen. Überzeugt euch vor allem, dass die Funktion `getBinomialDist` eine Binomialverteilung für `n` Experimente mit Erfolgswahrscheinlichkeit `p` zurückgibt.

Wir wollen hier Bernoulli-Experimente durchführen. Die ersten paar Aufgaben tun allerdings noch gar nichts Zufälliges, sondern nutzen den Umstand, dass wir hier die Verteilung ausrechnen können, um das Verhalten der Schätzer zu untersuchen.

(12.2)* Seht euch die Funktion compareEstimators an. Was tut sie, warum tut sie es so? Ruft sie auf, variiert das Argument. Was beobachet ihr? **(L)**

(12.3)* Seht euch die Funktion `compareBiases` an. Was tut sie? Wird hier tatsächlich der Bias (zum Parameter p) berechnet? (Dies ist eine gute Zeit, nochmal kurz über der Definitionsgleichung für den Bias zu meditieren).

Ihr müsst compareBiases im estimator-Argument entweder estML oder estELE übergeben. Probiert es zunächst mit estML. Bestätigt sich unsere Behauptung von oben, nach der der ML-Schätzer erwartungstreu sei?

(12.4)* Wie verhält sich der ELE-Schätzer in Sachen biases? Entspricht das eurer Erwartung aus dem Vorversuch mit compareEstimators?

(12.5) Schreibt auf dieser Basis eine Funktion `meanSquareError`, die den mittleren quadratischen Fehler der Schätzer zu Werten von p ausrechnet und vergleicht die Fehler von estML und estELE.

Datei(en) im PDF-Anhang: empBias.py

Konfidenzbereich

Konfidenzniveau

Konfidenzintervall

13. Schätztheorie II

Leitfragen zu Abschnitt 13

- Wie lässt sich die Zuverlässigkeit eines Schätzers definieren und bestimmen?
- Was genau ist es eigentlich, das Schätzer bestimmen?

Wie genau ist jetzt so ein Schätzer? Das weiß man nie, weil er durch seine Stochastizität immer beliebig daneben liegen kann. Man kann aber eine Wahrscheinlichkeit dafür angeben, dass ein Intervall das „wahre“ ϑ enthält. Um das zu untersuchen, wollen wir als Versuchsergebnis eine Zufallsvariable X mit Werten $x \in \mathcal{X}$ annehmen. Natürlich funktioniert das auch ohne Zufallsvariablen und direkt auf Ω, aber wir werden in der Regel nicht direkt Versuchsergebnisse, sondern von ihnen abhängige Zufallsvariablen untersuchen.

Wir geben eine Familie von Teilmengen $\{C(x) \subset \Theta \,|\, x \in \mathcal{X}\}$ an, so dass für alle $\vartheta \in \Theta$

$$P_\vartheta\big(\{x \in \mathcal{X} \,|\, \vartheta \in C(x)\}\big) \geq 1 - \alpha$$

ist. $C(x)$ heißt dabei *Konfidenzbereich* zum *Konfidenzniveau* $1 - \alpha$. In der Regel wird $C(x)$ ein *Konfidenzintervall* sein, also einfach alle Zahlen von $a \in \mathbb{R}$ bis $b \in \mathbb{R}$.

Seht euch diese Gleichung genau an – sie ist letztlich der Schlüssel zum Verständnis frequentistischer Schätztheorie. Das Argument von P_ϑ ist eine Menge, und der senkrechte Strich markiert keine bedingte Wahrscheinlichkeit, sondern ein „mit der Eigenschaft“ in der Menge. Intuitiv würde man von einem Schätzer gerne haben, dass er etwas wie $P(\vartheta \in C(x)) \geq 1 - \alpha$ sagt, dass also die Wahrscheinlichkeit, dass ϑ im Konfidenzintervall ist, größer als eine Schranke ist – damit könnten wir den Messfehler einer bestimmten Messung x kontrollieren.

Die Art, wie wir die Schätztheorie hier aufgezogen haben, lässt das aber nicht zu, weil ϑ keine Zufallsvariable ist (Nebenbei: $\hat{\vartheta}$ wäre eine Z.V., über der man Verteilungen haben kann – aber hier liefert der Schätzer ja stattdessen die $C(x)$ als Z.V.). Deshalb ist $\vartheta \in C(x)$ bei gegebenem x ist kein Ereignis, sondern einfach wahr oder falsch.

Unser Umweg über die Menge aller x, die für ein bestimmtes ϑ zusammen eine bestimmte Wahrscheinlichkeitsmasse sammeln, erlaubt uns das Rumdrücken um dieses konstruktionsbedingte Problem, allerdings um den Preis einer weniger intuitiven Interpretation.

Was wir kontrollieren, ist die Irrtumswahrscheinlichkeit bei einem bestimmten Modell. Ist nämlich das System, das wir beobachten, eine Realisierung des durch $\tilde{\vartheta}$ beschriebenen Modells, und beobachten wir das oft genug, werden nur 5% (für $\alpha = 0.05$) der Messwerte außerhalb des Konfidenzbereichs liegen. Leider ist damit keine Aussage möglich, wie wahrscheinlich Fehler für ein gegebenes x sind. Eine Möglichkeit, diesem Phänomen etwas nachzugehen, ist die Beobachtung der Wahrscheinlichkeitsmasse in den unten definierten $A(\vartheta)$ – da ist einfach alles möglich.

Eine – im statistischen Mainstream allerdings unpopuläre – Alternative, die eine intuitive Interpretation liefert, werden wir später als Belief Intervals kennenlernen.

Woher kommen die $C(x)$? Entweder aus Tschebytschew, der aber meistens zu große Intervalle liefert, weil er die Verteilung nicht kennt. Ansonsten existieren viele „vorberechnete“ Schätzer, für die Konfidenzintervalle bekannt sind. Im Allgemeinen ist das Verfahren aufwändig.

Idee: Gebe

$$\tilde{C} := \{(x, \vartheta) \,|\, \vartheta \in C(x)\}.$$

Zwei Mengen dieser Art zu verschiedenen Konfidenzniveaus sind die grauen Gebilde in der folgenden Abbildung. Die hellgraue Menge gehört zu einem Schätzer geringerer Signifikanz. Bei unseren

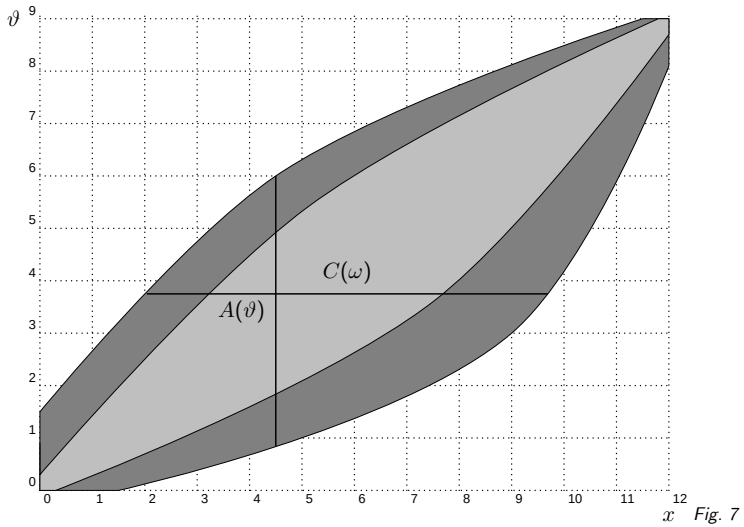

Fig. 7

diskreten Mengen $\mathcal{X}$ (unten) und Θ (links) wären diese Mengen natürlich nicht „Flächen“ sondern Mengen von Punkten. Die Bestimmung dieser Flächen, äquivalent duch $A(\vartheta)$ oder $C(x)$, ist unser Ziel.

(cf. Fig. 7)

Die Figur oben zeigt zwei mögliche $\tilde{C}$, wobei der hellere Part zwar kleinere Signifikanzintervalle liefert, diese aber weniger signifikant sind (nämlich weniger Wahrscheinlichkeit auf sich vereinen.)

Beachtet, dass ϑ hier irgendwelche Werte zwischen 0 und 9 annehmen kann. Das ist komplett in Ordnung, wir haben keine Aussagen darüber gemacht, was Θ eigentlich sei (auch wenn eine der Standardanwendungen die Schätzung des p von Binomialverteilungen und damit $\Theta = [0, 1]$ ist). Was der Parameter hier bedeutet, verrate ich natürlich nicht.

Konkret lassen sich die $A(\vartheta)$ wie folgt definieren:

$$A(\vartheta) = \{x \in \mathcal{X} \,|\, (x, \vartheta) \in \tilde{C}\}.$$

Die $A(\vartheta)$ enthalten also die x, die für ein gegebenes Signifikationniveau auf ϑ führen können.

Wir konstruieren die $A(\vartheta)$ so, dass

$$P_\vartheta(A(\vartheta)) \geq 1 - \alpha.$$

Daraus ergeben sich umgekehrt die $C(x)$.

Ein diskretes Beispiel. Wir wollen schätzen, wie groß die Wahrscheinlichkeit ist, dass auf das Wort „Louis“ das Wort „Philippe“ folgt und werden versuchen, das aus dem Text „Eighteenth Brumaire of Louis Bonaparte“ (Project Gutenberg, mar1810.txt) zu schätzen.

Wenn wir zunächst den Stichprobenraum $\tilde{\Omega}$, „Bigramme im Englischen“ betrachten und mit w_1, w_2 das erste bzw. zweite Wort eines Bigramms bezeichnen, können wir die Ereignisse $A =$

$\{w_2 = \text{Phillipe}\}$ und $B = \{w_1 = \text{Louis}\}$ definieren. Wir interessieren wir uns für die Größe $p := P(A \mid B)$. Der ML-Schätzer für diese Größe ist einfach:

$$\hat{p} = \frac{|A \cap B|}{|B|},$$

also die Zahl von „Louis Philippes“ durch die Zahl der „Louis“. In unserem Text sind das 19 bzw. 43; die Erwartung wäre also, dass etwas wie 0.44 geschätzt werden sollte.

Um unser Schätzproblem zu formulieren, nehmen wir jetzt alle möglichen (englischen) Texte als Ω. Wir betrachten die Zufallsvariable $X = |A \cap B|$. Als Modelle für die Verteilungen dieses X nehmen wir Bernoulli-Verteilungen an:

$$P_\vartheta(X = x) = \binom{|B|}{x} \vartheta^x (1-\vartheta)^{|B|-x}.$$

Zum Zweck der Darstellung legen wir uns auf $|B| = 43$ fest – das ist nicht so arg schlimm, wir könnten etwa, wenn wir auf die Ganzzahligkeit von x verzichten, einfach alle x mit $43/|B|$ multiplizieren. In der Realität würde man allerdings wahrscheinlich gleich $X = |A \cap B|/|B|$ setzen. Dadurch würde hier natürlich das bequeme Zählen verloren gehen.

Als Θ würde man normalerweise $[0, 1]$ verwenden; wir wollen es aber diskret haben und setzen $\Theta = \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

Durch simples Ausrechnen von $P_\vartheta(x)$ für die verschiedenen Elemente von Θ und x errechnet sich folgendes Tableau:

x	0-4	5-9	10-14	15-19	20-24	25-29	30-34	35-40	40-44
ϑ									
0.1	♡0.547	♡0.443	0.010	0.000	0.000	0.000	0.000	0.000	0.000
0.3	0.001	♡0.108	♡0.563	♡0.306	0.022	0.000	0.000	0.000	0.000
0.5	0.000	0.000	0.011	♡0.214	♡0.549	♡0.214	0.011	0.000	0.000
0.7	0.000	0.000	0.000	0.000	0.022	♡0.306	♡0.563	♡0.108	0.001
0.9	0.000	0.000	0.000	0.000	0.000	0.000	0.010	♡0.443	♡0.547
$C(x)$	{.1}	{.1,.3}	{.3}	{.3,.5}	{.5}	{.5,.7}	{.7}	{.7,.9}	{.9}

Dabei wurden, beginnend bei den größten, immer alle Werte in einer Zeile mit Herzlein markiert, bis die Summe der geherzten Wahrscheinlichkeiten über 0.95 lag – die in einer Zeile geherzten x bilden dann zusammen die $A(\vartheta)$ zum Signifikanzniveau 0.05. Dann lassen sich in den Spalten die $C(x)$ ablesen. Damit weiß ich beispielsweise, dass bei 19 beobachteten Phillipes die Menge $\{0.3, 0.5\}$ nur mit einer Wahrscheinlichkeit von 0.05 das tatsächliche ϑ nicht enthält.

Ein anderes Problem ist natürlich, dass der Text, aus dem wir diese Schätzung abgeleitet haben, eher nicht repräsentativ für Texte ist – ein schneller Blick auf google lehrt, dass am Netz etwa 11 Millionen Bigramme mit Louis existieren, aber nur etwa 67 000 auch mit Philippe weitergehen. Solche Fehler werden wir bald als „Fehler dritter Art“ kennen lernen.

Natürlich macht in Wirklichkeit niemand solche Rechnungen (oder jedenfalls nur im äußersten Notfall). Wenn man in der Praxis Schätzer verwendet, ignoriert man entweder einfach den Umstand, dass die Schätzungen fehlerbehaftet sind (das machen leider viel zu viele Leute, auch wenn es durchaus genug Anwendungen gibt, bei denen das legitim ist), oder man bekommt mit der Formel für den Schätzer eine Formel für die Größe der Konfidenzbereiche mit.

Um eine ganz grobe Schätzung der Größe der Konfidenzbereiche zu bekommen, reicht es häufig auch, aus den rohen Zählungen N schlicht $N - \sqrt{N}$ und $N + \sqrt{N}$ zu berechnen und diese durch die jeweilige Maschinerie zu ziehen. Das Ergebnis könnte mit Glück der so genannte 1-σ-Konfidenzbereich sein, der einem Schätzer zum Niveau 0.33 entspricht. Dass diese Magie häufig funktioniert, liegt am oben erwähnten zentralen Grenzwertsatz. Da wir uns mit diesem nicht näher auseinandergesetzt haben, wollen wir auch dieser Faustregel nicht näher unter die Haube gucken.

Aufgaben

(13.1)* Besorgt euch von der Vorlesungsseite das Programm confInter.py. Um es laufen lassen zu können, benötigt ihr die Tkinter-Erweiterung von python, die aber in fast allen üblichen Python-Paketen enthalten ist. Sorgt dafür, dass oben im Programm etwas wie

```
totalDrawn = 10
thetaDiscPoints = 10
useGaussian = 0
```

steht. Das bedeutet: unser Experiment ist ein Produktexperiment aus 10 Bernoulli-Experimenten (totalDrawn); wir werden uns für die Zahl der Erfolge interessieren, also haben wir $\Omega = \{0, 1, \ldots, 10\}$ (wir brauchen hier nicht den Umweg über eine Zufallsvariable). Weil wir diskret schätzen wollen, müssen wir die möglichen Werte von ϑ (nämlich der Erfolgswahrscheinlichkeit in einem Bernoulli-Experiment) irgendwie diskretisieren – hier wollen es 10 verschiedene Werte sein (thetaDiscPoints). Und schließlich wollen wir vorerst wirklich im einer Binomialverteilung arbeiten (useGaussian). Ihr könnt diese Werte nach einem ersten Test beliebig ändern – aber Vorsicht, totalDrawn und thetaDiscPoints haben einen dramatischen Einfluss auf die Geschwindigkeit des Programms.

Startet das Programm und macht euch klar, wie das, was ihr seht, mit der besprochenen Konstruktion von Konfidenzintervallen zusammenhängt.

Anmerkung: Das hier ist zwangsläufig ein ziemlicher Haufen Code, und ich war zu faul, ihn ordentlich zu dokumentieren. Ich hoffe, es ist nicht allzu schwer, die (wenigen) „interessanten“ Stellen zu finden. **(L)**

(13.2)* Wenn ihr auf ein Feld (links-) klickt, wird es grün, und es wird zu A(theta) hinzugefügt (die Summe ganz rechts ändert sich). Fangt in jeder Zeile bei der größten Wahrscheinlichkeit mit dem Markieren an und markiert die nächstkleineren, bis ihr die Summe 1-gewünschtes Niveau erreicht habt. Wenn ihr das in jeder Zeile gemacht habt, könnt ihr die Konfidenzintervalle für alle möglichen x in den grünen Feldern jeder Spalte ablesen. Probiert das auf jeden Fall mal mit einem Niveau von 0.2 und verschärft den Schätzer dann auf ein Niveau von 0.05. Was beobachtet ihr?

Wenn wir wollt, ändert thetaDiscPoints und totalDrawn und seht, wie das Spiel dann aussieht.

(13.3) Neben der Binomialverteilung, die wir ja bisher für P_ϑ verwendet haben, habe ich auch noch eine diskretisierte Gaussverteilung vorbereitet. Sie kann per `useGaussian = 1` gewählt werden und wird unten im „main“ erzeugt. Experimentiert auch mit ihr. Der entscheidende Vorteil davon ist, dass wir hier einfach an der Varianz der Verteilung spielen können und nachsehen, wie sich die Konfidenzintervalle dadurch verändern. Die Varianz ist der Ausdruck `(totalDrawn+1)*p*(1-p)` in der Instanzierung der zuständigen `Discrete2DFunc`. Hängt da zunächst mal ein `/4.` dran, um nur ein Viertel der Varianz der genäherten Binomialerteilung zu erhalten. Wie verändern sich die Konfidenzintervalle?

(13.4) Seht euch die Methode `ConfInspector._selectForRowSum` an und überzeugt euch, dass sie genau das, was ihr eben per Hand gemacht habt, automatisch tut. Dabei steht 1-Niveau zur Zeit als default parameter im Funktionskopf. Die Methode ist an die rechte Maustaste jedes Feldes gebunden, wenn ihr also rechts in eine Zeile klickt, markiert die Methode euer $A(\vartheta)$ automatisch. Beobachtet für ein paar Niveaus (und größere totalDrawn und thetaDiscPoints – die Grenze ist da wohl die Größe eures Bildschirms), wie sich die $A(\vartheta)$ und damit auch die $C(\omega)$ verhalten. Tut das sowohl für die Binomialverteilung als auch für ein paar Varianzen unserer diskretisierten Gauss-Verteilung.

Datei(en) im PDF-Anhang: confInter.py

14. Der Good-Turing-Schätzer I

adding one

Leitfragen zu Abschnitt 14

- Was sind die speziellen Anforderungen an Schätzer für Wortfrequenzen oder ähnliche linguistische Daten?
- Welche populären Ansätze zu deren Erfüllung gibt es? Was ist von ihnen zu halten?
- Wann greift der Good-Turing-Schätzer? Was ist seine Grundidee, was muss erfüllt sein, damit sie gut ist?

ML-Schätzer sind notorisch schlecht für Sprachdaten, vor allem, weil nicht beobachtete Phänomene automatisch Wahrscheinlichkeit Null bekommen und beobachtete Phänomene überschätzt werden.

Wir diskutieren die Alternativen am Beispiel des Wörterzählens. Natürlich würde das ganz analog mit n-grammen oder was immer gehen.

Ein einfacher Trick ist *adding one*. Dabei berechnet man

$$P_{\mathsf{ao}}(w_i) = \frac{|w_i|+1}{N+B} \quad \text{statt} \quad P_{\mathsf{ML}}(w_i) = \frac{|w_i|}{N}.$$

Dabei ist N die Zahl der Wörter (word tokens) im Korpus, $|w_i|$ die Häufigkeit des betrachteten Wortes und B die Anzahl der word types (die ich hier mal „Spezies“ nennen möchte, weil der Schätzer natürlich für alles mögliche funktioniert), die wir erwarten. Im Beispiel ist das schon etwas schwierig, denn die Anzahl verschiedener Wörter in einer Sprache ist keine zugängliche Größe (auch wenn ich argumentieren würde, dass sie finit ist). Einfacher ist das z.B. mit n-grammen über einer bekannten Anzahl grammatischer Kategorien o.ä.

Der Effekt ist, dass Wahrscheinlichkeit für die unbeobachteten Spezies reserviert wird. Beobachten wir etwa nur $n < B$ verschiedene Wörter, so ist die Summe der Wahrscheinlichkeiten

$$\sum_{i=1}^{n} \frac{|w_i|+1}{N+B} = \frac{1}{N+B}\left(\sum_{i=1}^{n}|w_i| + \sum_{i=1}^{n} 1\right) = \frac{N+n}{N+B},$$

da natürlich die Summe über alle $|w_i|$ wieder die Gesamtzahl der Wörter ist. Damit haben wir $1 - \frac{N+n}{N+B}$ als reservierte Wahrscheinlichkeit, die wir z.B. gleichmäßig über alle unbeobachteten Spezies verteilen können.

Obwohl diese Variation des oben erwähnten ELE-Schätzers nicht so beliebig ist, wie es vielleicht scheinen mag (es ist das Ergebnis, wenn man nach Bayes schätzt und annimmt, dass alle Spezies a priori gleich wahrscheinlich sind), sind die Ergebnisse meist schlecht, zumal für beobachtete Spezies oft schlechter als ML-Schätzungen.

Besser ist der Good-Turing-Schätzer, der für typische Anwendungen der Computerlinguistik einen guten Kompromiss aus bias und mittlerem quadratischen Fehler bietet.

Die Situation: Wir haben s Spezies, die unabhängig voneinander mit Wahrscheinlichkeiten p_i, $i = 1, \ldots, s$ beobachtet werden. Die Stichprobe besteht aus tatsächlich beobachteten Frequenzen $r_1, \ldots, r_s$ mit $N = \sum_{i=1}^{s} r_i$.

Wir disktuieren hier „Spezies“, weil, wie gesagt, bei weitem nicht nur Word Types in Frage kommen.

Grundidee

Fit

Wir fragen zunächst nach der Anzahl der gerade r Mal auftretenden Spezies, n_r. Für diese lässt sich zeigen:

$$r^* = (r+1)\frac{E(n_{r+1})}{E(n_r)} \quad P_0 = \frac{E(n_1)}{N}$$

Dabei ist r^* die nach den Wahrscheinlichkeiten zu beobachtetende Zahl der Individuen der mit beobachteter Häufigkeit r, P_0 die Wahrscheinlichkeit für alle nicht beobachteten Spezies (die „Missing Mass“) zusammen.

Der Beweis dieses Theorems ist zu aufwändig, als dass ich ihn hier breit diskutieren wollte – er braucht aber eigentlich keine schweren Geschütze. Ausgangspunkt ist dabei, dass die beobachteten Frequenzen mit p_i Bernoulli-verteilt sind. Damit lässt sich $E(n_r)$ hinschreiben. Mit einigen Umformungen auch des Binomialkoeffizienten lässt sich der Beweis führen.

Übrigens verwende ich hier Sampsons Notation. Wir haben Schätzer bisher immer mit einem Zirkumflex geschrieben, die Größe oben wäre dann also etwas wie $\hat{r}$, also ein Schätzer für die „wahre“ Häufigkeit.

Für $E(n_r)$ könnten wir unsere Beobachtungen einsetzen. Für kleine r ist das auch gut (weil n_r groß ist); für große r klappt das nicht mehr gut, weil n_r klein oder gar Null wird.

Es ist die Lösung dieses Problems, das die Anwendung von Good-Turing im allgemeinen Rahmen etwas schwierig macht. Für praktisch alle interessanten Anwendungen in der Computerlinguistik gibt es aber eine recht einfache Lösung: Wir fitten Zipf's Law auf die beobachtete Verteilung und schätzen $E(n_r)$ für kleine n_r aus dem Fit. Das resultierende Verfahren wird meistens unter einem Titel wie Simplified Good-Turing (SGT) vorgestellt. Mehr darüber ist in dem sehr empfehlenswerten Buch „Empirical Linguistics“ von Geoffrey Sampson zu erfahren.

Was ist ein *Fit*? Wenn man Beobachtungsdaten hat und vermutet, dass sie eigentlich durch eine Funktion einer bestimmten Form beschrieben werden sollten, versucht man, die Parameter der Funktion so zu bestimmen, dass sie die Beobachtung „möglichst gut“ beschreiben, dass also ein irgendwie definierter Fehler möglichst klein wird. Wie wir das hier machen, sehen wir unten.

Aufgaben

(14.1)* Zipf's Law besagt in der Originalfassung, dass die Frequenz eines Wortes umgekehrt proportional zum Rang der Wortfrequenz ist (dabei hat das häufigste Wort Rang 1, das zweithäufigste Rang 2 usf). Wenn wir diesen Rang R nennen, heißt das, dass $r = C/R$ gelten muss, wobei r die Häufigkeit des R-t häufigsten Wortes und C eine irrelevante Konstante ist.

In der Tat will man solche Gesetzmäßigkeiten noch etwas allgemeiner haben und zulassen, dass am R noch eine Potenz steht, so dass die allgemeine Form des Gesetzes $n = Cm^{-s}$ ist – die Zipf-Beziehung oben ist dann der Spezialfall $s = 1$.

Zeigt, dass aus einem Gesetz wie $n = Cm^{-s}$ durch Ziehen des Logarithmus auf beiden Seiten der Gleichung ein Gesetz wie $x = -sy + z$ wird. Was sind x, y und z? **(L)**

(14.2)* Holt euch von der Webseite das Programm `checkZipf.py` und, sofern ihr keine eigenen Texte ausreichender Länge habt, noch `mar1810.txt`. Werft einen kurzen Blick über das Programm. Unter der Zuweisung an `fofDist` („Frequencies of frequency distribution“) stehen ein ganzer Haufen Funktionsaufrufe. Kommentiert alle bis auf den ersten aus.

Seht euch nochmal die Funktion `getFoFDistribution` an, bis ihr überzeugt seid, das sie tatsächlich ein Dictionary zurückgibt, in dem die Frequenzen die Schlüssel und die Frequenzen dieser Frequenzen als Werte stehen. Lasst sie euch dann per `prettyPrintDist` ausgeben.

Überprüft an ein paar Stichproben, ob rR wirklich in etwa konstant ist (wie Zipf's Gesetz verlangt). Vorsicht: Die Verteilung, die ausgeben wird, ist r gegen n_r, den Rang müsst ihr also auszählen (was aber nicht irre viel Arbeit ist, wenn man einmal gezählt hat, wie lang eine Spalte ist).

Auf Dauer ist das trotzdem mühsam, und drum gibt es die Funktion checkRankLaw. Seht sie euch an und lasst sie laufen. Versucht die Abweichungen von Zipf's Law, die ihr (wahrscheinlich) seht, zu beschreiben. **(L)**

(14.3)* Eine alternative Formulierung von Zipf's Law (die so auch gern als Benford's Law bezeichnet wird), ist $n_r = C/r^s$ mit $s \approx 2$. Da wir in unserer Behandlung des Good-Turing-Schätzers mit Sampson argumentieren und also mit Frequenzen und nicht mit Rängen arbeiten wollen, wird das die Formulierung sein, die wir verwenden. Die Funktion `checkFreqLaw` gibt $n_r r^2$ nach Rang aus. Versucht, die offensichtlichen Abweichungen von der Erwartung

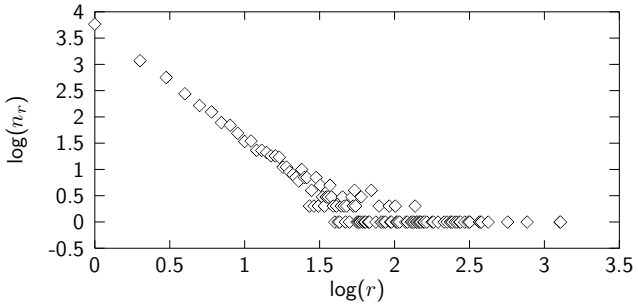

Fig. 8

wieder zu kommentieren. Vielleicht wollt ihr auch etwas am Exponenten von `freq` in checkFreqLaw rumspielen, um der Erwartung eines flachen Verlaufs näher zu kommen (richtig flach werdet ihr es wohl nicht hinkriegen, eine „Badewanne“ bleibt meistens). **(L)**

(14.4)* Nach der Vorübung müsste $\log n_r$ über $\log r$ aufgetragen eine Gerade sein, wenn die Behauptung aus der letzten Aufgabe stimmt(e), etwa mit Steigung -2. Um das zu sehen, wollen wir Plots machen. `checkZipf` ist auf eine Zusammenarbeit mit gnuplot vorbereitet – ich empfehle euch, euch das Programm, das auf allen auch nur irgendwie diskutablen Maschinen kostenlos verfügbar ist, zu installieren, wenn ihr es nicht ohnehin schon habt.

Die Funktion printGnuplotFile erzeugt Eingaben für Gnuplot aus einer Liste von Python-Werten (die Details sind nicht so interessant). Ihr seht zwei Aufrufe von printGnuplotFile am Ende des Skripts. Der erste erzeugt einen Plot von n_r gegen r in linearem Maßstab. Seht ihn an und beschließt, dass ihr nichts erkennt.

Probiert danach den zweiten aus. Ihr solltet über weite Bereiche eine Gerade erkennen können. Hat sie in etwa Steigung zwei? Woher kommt das Auseinanderstreben der Punkte, wenn wir uns $\log n_r = 0$ nähern? Wie ist der Zusammenhang mit den Problemen, die wir oben festgestellt haben?

Technische Anmerkung: Am Bequemsten könnt ihr mit checkZipf und gnuplot ohne weitere Tricks arbeiten, indem ihr zwei Fenster offen haltet. In einem ruft ihr etwas wie `checkZipf.py mar1810.txt > topl.gnu` auf – damit ist in topl.gnu ein Skript, das gnuplot verarbeiten kann (vorausgesetzt, ihr habt alles außer dem einen printGnuplotFile-Aufruf wieder auskommentiert).

Im anderen Fenster habt ihr dann Gnuplot offen und könnt darin `load "topl.gnu"` sagen und solltet den Graphen sehen.

(14.5) Erweitert das Programm, so dass es mehr als eine Datei verarbeiten kann und sammelt so viel Text wie ihr könnt. Seht euch die Kurven für möglichst große Datensammlungen an.

Datei(en) im PDF-Anhang: checkZipf.py mar1810.txt

15. Der Good-Turing-Schätzer II

Leitfragen zu Abschnitt 15

- Warum taugen die rohen Verteilungen von z.B. Wortfrequenzen nicht als Eingabe für einen Good-Turing-Schätzer?
- Was ist Glättung, wann klappt sie gut, wann geht sie in die Hose?
- Wozu dient hier die lineare Regression?

Zipf's law spricht von Wortfrequenzen r und Frequenzen von Wortfrequenzen n_r. Uns interessiert also z.B. die Information, dass es $n_{20} = 4$ Spezies gibt, die $r = 20$ Mal im Korpus auftreten. Außerdem behauptet Zipf's Law, dass eine lineare Beziehung zwischen den *Logarithmen* von r und n_r gibt.

Plot von $\log n_r$ gegen $\log r$ für einen kleinen Korpus:

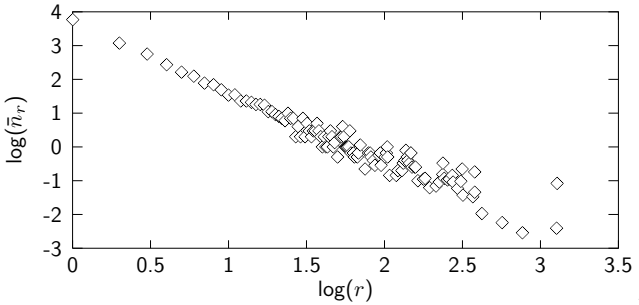

Fig. 9

(cf. Fig. 8)

Es ist wichtig, diesen Plot zu verstehen, zumal er anders orientiert ist als die, die üblicherweise als Illustration für Zipf's Law verwendet werden: Ganz links stehen die Spezies mit $r = 1$, also Wörter, die nur ein Mal vorkommen. Davon gibt es hier etwa $10^{3.7}$. Dann gehen wir weiter zu Spezies mit zwei Vertretern usf. Rechts stehen demgegenüber die Spezies mit vielen Vertretern. Dabei gibt es zu jeder Frequenz r nur eine Spezies, $\log(n_r) = 0$.

Während man bei den niedrigfrequenten Wörtern Zipf's Law recht gut realisiert sieht, ist das bei den hochfrequenten Wörtern nicht mehr so. Das liegt daran, dass es mehr oder minder Zufall ist, ob „ist“ gerade 2104 Mal vorkommt oder doch 2105 Mal. Wir müssten eigentlich den Beitrag von „ist“ zu n_{2104} für etliche der benachbarten n_r „verschmieren“, die Kurve also glätten.

1. Schritt: Glätten der beobachteten Verteilung

Wir behaupten, dass für große r die zugehörigen n_r-Werte reduziert werden müssen. Eine gute Methode dazu ist, das n_r gleichmäßig über die „leeren“ r in der Nachbarschaft zu verteilen.

Konkret: Wir beobachten F Frequenzen r_i, $i = 1, \ldots, F$. Für große r_i werden r_{i-1} und r_{i+1} in der Regel ziemlich weit von r_i abliegen (z.B. $r = 586$, daneben $r = 422$ und $r = 768$, alle mit $n_r = 1$ – die betreffenden Spezies sind hier „in“, „das“, und „und“). Wenn wir das $n_{586} = 1$ über die ganze Nachbarschaft jeweils bis zur Hälfte des benachbarten r verschmieren. Die Größe dieser Nachbarschaft ist $l = (768 - 422)/2$, statt n_{586} nehmen wir also n_{586}/l.

Allgemein glätten wir, indem wir $\bar{n}_r = 2n_r/(r_+ - r_-)$ setzen, wobei r_+ der nächsthöhere beobachtete r-Wert ist und r_- der nächstniedrigere. Bei $r = 1$ setzen wir $r_- = 0$, beim höchsten Wert für r setzen wir r_+ auf $r + (r - r_-)$ (wir erweitern die Nachbarschaft symmetrisch).

Das Ergebnis ist, dass sich n_r-Werte gar nicht ändern, wenn wir sowohl für $r - 1$ als auch für $r+1$ Spezies haben, dass sie aber um so stärker reduziert werden, je weiter der Abstand zwischen zwei „besetzten“ r wird.

Das Ergebnis sieht dann so aus:

(cf. Fig. 9)

Das ist offenbar eine erhebliche Verbesserung gegenüber den rohen Daten. Vor allem die oberste Frequenz reißt aber etwas aus. Das ist ein Artefakt des Umstandes, dass die beiden häufigsten Wörter „die“ und „der“ mit $r = 1269$ und $r = 1281$ eigentlich viel zu nahe beieinanderliegen – das nächsthäufige Wort bei uns ist „und“ mit $r = 768$. Solche Dinge passieren, und Schätzer müssen damit fertig werden („robust sein“).

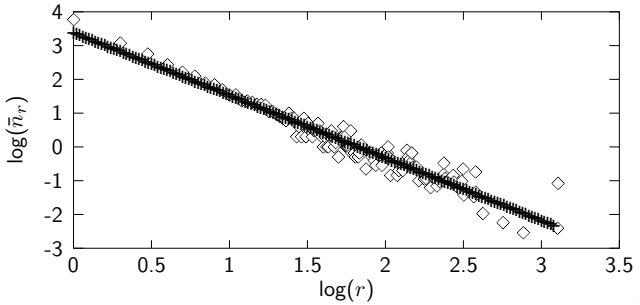

Fig. 10

2. Schritt: Lineare Regression

Zipf's Law behauptet, dass wir hier eine Gerade durchlegen können. Dazu dient die *lineare Regression*, ein mathematisches Verfahren, das die „beste" Gerade $ax+b$ durch die beobachteten Daten $(\log r, \log \bar{n}_r)$ findet.

Dazu minimiert man den quadratischen Fehler der geschätzten Gerade gegenüber den gegebenen Punkten, $\sum_r (\log n_r - a \log r - b)^2$, d.h. sorgt dafür, dass die Ableitung dieser Größe Null wird. Das Ergebnis ist ein 2×2-Gleichungssystem für die Steigung a und den Achsenabschnitt b und kann gelöst werden. Näheres in Press et al: Numerical Recipies.

Übrigens: Dies ist eine (bei normalverteilten Fehlern in $\bar{n}_r$) eine ML-Schätzung, die hier aber ausreichend robust ist (was nicht heißt, dass das nicht besser ginge).

Ergebnis der linearen Regression:

(cf. Fig. 10)

16. Der Good-Turing-Schätzer III

Leitfragen zu Abschnitt 16

- Wie baut man die Schritte der letzten beiden Folien zusammen, um endlich zu Schätzwerten zu kommen?
- Wie können wir sehen, ob sich die Arbeit gelohnt hat?

Schritt 3: Berechnung der Schätzwerte

Wir können jetzt die Good-Turing-Formeln von oben anwenden. Zunächst ist $P_0 = n_1/N$. Was mit dieser Größe anzufangen ist, hängt von der Anwendung ab.

Wir rechnen jetzt r^* als $(r+1)n_{r+1}/n_r$ aus. Mit fallenden n_r wird das irgendwann schlechter als als die Schätzungen aus der linearen Regression. Spätestens wenn ein n_{r+1} undefiniert ist, schalten wir um auf

$$r^* = (r+1)\frac{10^{a \log(r+1)+b}}{10^{a \log r+b}}$$

mit der Steigung a und Achsenabschnitt b der Regressionsgeraden.

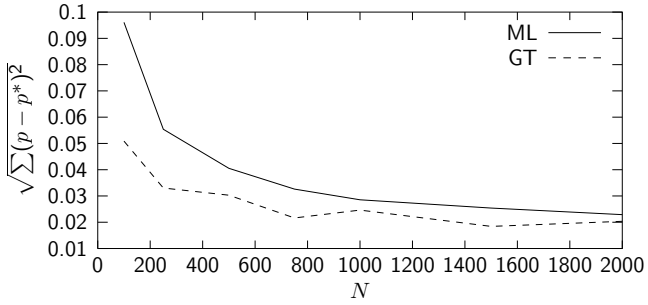

Fig. 11

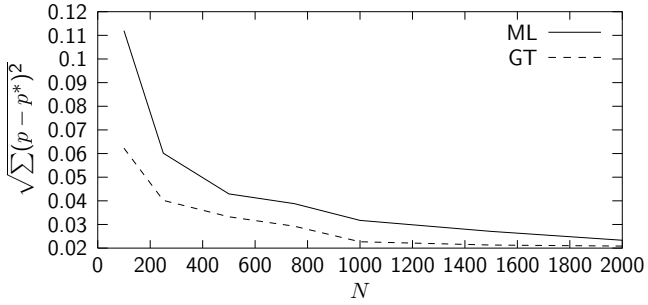

Fig. 12

Dieser Ausdruck lässt sich natürlich zu $r^* = (r+1)10^{a \log((r+1)/r)}$ vereinfachen. Tieferes Nachdenken liefert, dass man schon Umschalten sollte, wenn die Differenz aus rohen und geglätteten r^* größer wird als

$$1.96\sqrt{(r+1)^2\frac{n_{r+1}}{n_r^2}\left(1+\frac{n_{r+1}}{n_r}\right)}.$$

Die Wurzel in diesem magischen Wert ist gerade eine Schätzung für die Varianz von r (das ja eine Zufallsvariable ist), die 1.96 kommt daraus, dass die Entscheidung wieder mal zu einem Niveau von 0.05 gefällt werden soll. Es ist offensichtlich, dass man sich an diese Vorschriften nicht allzu sklavisch halten muss.

Evaluation

Test mit bekannter Ground Truth: Synthese eines Textes auf der Basis eine angenommenen Verteilung von 8743 Wörtern. Plot des quadratischen Fehlers, zunächst nur für gesehene Wörter:

(cf. Fig. 11)

Mit $P(w) = P_0/|U|$ für ungesehene Wörter bei Good-Turing (U ist die Missing Mass):

(cf. Fig. 12)

Aufgaben

(16.1)* Besorgt euch die Dateien bigrams.dist, goodturing.py und bigramModels.py von der Webseite zur Vorlesung. Werft einen kurzen Blick auf goodturing.py, um eine Vorstellung zu bekommen, dass hier tatsächlich ein Good-Turing-Schätzer implementiert ist.

Nehmt zunächst die Datei bigrams.dist. In ihr stehen in jeder Zeile zunächst zwei Zeichen und dann die Häufigkeit dieser Zeichenkombination. Prüft zunächst, ob die (hier aus irgendeinem deutschen Korpus gewonnenen) Bigramme von Buchstaben auch annähernd nach Zipf verteilt sind. Das geht mit der Unix-Shell und gnuplot recht einfach: Mit

```
cut -b 4- bigrams.dist | sort -n | uniq -c > rnr.dist
```

bekommt ihr in rnr.dist eine Verteilung von r über n_r (Vorsicht: der Plot wird dann anders orientiert sein als gewohnt, wir haben bisher immer n_r über r aufgetragen). Lest in den manpages zu den Programmen nach, was hier passiert, wenn es euch nicht eh schon klar ist. Jetzt ruft ihr gnuplot auf und sagt

```
set logscale xy
plot "rnr.dist" with points
```

und solltet sehen, dass wir nicht ewig weit von Zipf's Law weg sind.

(16.2)* Seht euch jetzt bigramModels.py an. Die drei Klassen für die Modelle sind hier interessant, wobei der eigentliche Kicker die estimate-Methoden der ML- und GT-Bigramm-Modelle sind, die aus den `gefeedeten` Daten die Verteilungen der Bigramme schätzen. Versucht, zu verstehen, was darin passiert.

Die Funktion runTest ist demgegenüber langweilig – sie fragt nach einer Eingabe und lässt dann die beiden Schätzer beurteilen, wie wahrscheinlich die Eingabe unter den Schätzern ist.

Lasst das Programm laufen und seht nach, wie gut die beiden Modelle geeignet sind Tippfehler zu finden (wenn wir das nach Bigrammodell wahrscheinlichere Wort als das richtige akzeptieren). Ganz aufschlussreiche Eingaben sind z.B.

der dre

(hier stimmen die Modelle überein, „der“ ist besser. Na ja.)

Xylophon Xlyophon

(hier versagt ML mit dem verwendeten Korpus, beide Wahrscheinlichkeiten sind Null, während Good-Turing das richtige Wort auswählen würde)

Methyämoglobinämie Methyamoglobinämie

(hier liegen beide daneben, auch wenn GT immerhin der ersten, richtigen, Fassung nicht Wahrscheinlichkeit Null zuordnet. Aber natürlich folgen diese Wörter nicht wirklich einem deutschen Sprachmodell)

Wort Wrot

(hier liegen wieder beide richtig, wenn auch ML dem Wrot Wahrscheinlichkeit Null zuweist).

Ansonsten lasst eurer Fantasie freien Lauf; wenn ihr gute Beispiele habt, lasst es mich wissen – hier sind ein paar Vorschläge, die mich so erreicht haben:

- Sehenswürdigkeit – Sehenswrüdigkeit

Datei(en) im PDF-Anhang: goodturing.py bigramModels.py bigrams.dist makeCharBigramModel.py

17. Entropie

Leitfragen zu Abschnitt 17

- Was ist Information? Warum ist unsere mathematische Definition nicht weit von einer intuitiven entfernt?
- Warum brauchen wir Verteilungen, um Information definieren zu können?
- Warum geht die Bestimmung von Entropie immer nur im Rahmen von Modellen?

Der Begriff der Entropie kommt ursprünglich aus der Thermodynamik und bedeutete dort etwas wie den Grad der Unordnung – eine Mischung aus Helium und Wasserstoff hat höhere Entropie als ein System, in dem die beiden Gase getrennt vorkommen.

Umgekehrt brauche ich bei hoher Ordnung weniger Information, um das Gesamtsystem zu beschreiben. Auf Datenströme (wie Sprachäußerungen) bezogen wird die Entropie dann gleich ein Maß für den Informationsgehalt.

Wie viel Information steckt in einer Nachricht? Modell für Nachricht: Folge von Zufallsvariablen $X = X_1, \ldots X_n$. Annahme zunächst: Alle X_i unabhängig und mit $P(X)$ identisch über $\mathcal{X}$ verteilt.

Wir werden nicht den Informationsgehalt einer Nachricht als solcher analysieren, sondern den der ihr zugrundeliegenden Verteilung. Hauptgrund dafür ist, dass wir Information statistisch definieren möchten und wir so nicht vom Ergebnis eines Zufallsexperiments ausgehen können – die Unterstellung ist: „Wir können beliebig viele Nachrichten generieren“, und genau das geht natürlich nicht, wenn wir nur eine Nachricht haben.

Anschaulich gesprochen ist der Informationsgehalt einer Nachricht auch tatsächlich nicht zu bestimmen, ohne zu wissen, was noch alles (mit welcher Wahrscheinlichkeit) übermittelt werden kann. Im Extremfall enthält die Nachricht „Ich habe eine Sechs gewürfelt“ gar keine Information, wenn nämlich der Würfel immer nur eine Sechs zeigt (und wir nicht am Zeitpunkt des Würfelwurfs interessiert sind).

Aber natürlich hindert uns niemand daran, aus einer konkreten Nachricht die zugrundeliegende Verteilung ihrer Bestandteile – so sie denn welche hat – zu schätzen und daraus eine Entropie der Nachricht zu bestimmen.

In diesem Sinne ist die Modellierung einer Nachricht bereits ein Spezialfall. Tatsächlich könnten wir, wenn wir etwa an der Entropie von Sprachäußerungen interessiert sind, einfach alle Äußerungen nehmen und eine Verteilung über sie ermitteln. Wenn wir allerdings diese Verteilung irgendwie sinnvoll bestimmen können, können wir auch den Lehrstuhl zumachen, denn dann hätten wir wohl alle Probleme der maschinellen Sprachverarbeitung gelöst…

Für ein vernünftiges Maß der Information H wollen wir:

1. $H(X)$ ist in allen Werten von $P(X)$ stetig (Wir wollen nicht, dass es plötzlich große Sprünge gibt, wenn wir nur ein ganz klein wenig an einer Wahrscheinlichkeit drehen)
2. $H(X)$ wird für Gleichverteilung maximal (eine Abweichung von der Gleichverteilung teilt uns bereits einiges über X mit, das wir nicht mehr aus der Nachricht entnehmen müssen, d.h. der Informationsgehalt der Nachricht sinkt)
3. Information soll in einem gewissen Sinn additiv sein (für unabhängige Zufallsvariable soll $H(X,Y) = H(X) + H(Y)$ gelten – um eindeutig auf die Entropie zu kommen, müsste diese Forderung noch etwas verschärft werden)

Ein Maß, das diese Forderungen erfüllt, ist die *Entropie*, konventionell definiert als

$$H(X) = -\sum_{x\in\mathcal{X}} P(x) \operatorname{ld} P(x).$$

Dabei bezeichnet ld den *Logarithmus Dualis*, den Logarithmus zur Basis 2. Nebenbei: Es gilt $\operatorname{ld} x = \ln x / \ln 2$. Dass gerade die Basis 2 gewählt wurde (und darin, nicht in der funktionalen

Form, liegt die Konvention), liegt daran, dass wir unsere Entropie in *bits* ausdrücken wollen, also in Informationseinheiten, die gerade zwei Zustände annehmen können. Wollten wir unsere Informationen in Dezimalzahlen ausdrücken und demnach bestimmen, wie viele Dezimalstellen zur Repräsentation der Daten nötig wären, könnten wir den dekadischen Logarithmus nehmen. Nur würde uns dann jedeR missverstehen.

bits

Hinweis: $P(x)$ ist nur eine Abkürzung für $P(X = x)$. Manning und Schütze mögen diese Abkürzung in ihrem Abschnitt über Entropie noch nicht (wiewohl sie sie später durchaus verwenden), weshalb dort eine Funktion $p : \mathcal{X} \to \mathbb{R}$ definiert wird durch $p(x) = P(X = x)$.

Diese Definition sieht zunächst ein wenig beliebig aus. Das ist sie aber nicht. Betrachten wir dazu Zahlenraten, der Einfachheit halber von Zahlen $\leq 2^n$. Bei jedem Rateschritt gewinnen wir ein Bit Information (gesuchte Zahl ist kleiner/größer als geratene Zahl). Wie viele Rateschritte brauchen wir im schlimmsten Fall unbedingt (d.h. wenn wir maximal geschickt fragen)? Erste Frage: Größer als 2^{n-1}? Zweite Frage: Größer als $A_1 2^{n-1} + 2^{n-2}$? k-te Frage: Größer als $\sum_{i=1}^{k-1} A_i 2^{n-i} + 2^{n-k}$; dabei ist $A_i = 1$, wenn die i-te Frage bejaht wurde, 0 sonst. Wenn 2^{n-k} eins ist, wissen wir die Zahl (da wir nur ganze Zahlen raten und sich ganze Zahlen gerade um eins unterscheiden). Damit brauchen wir n binäre Fragen, um Zahlen bis 2^n zu erraten.

Zusammenhang mit der Entropie: Sei die zu ratende Zahl x ein Wert einer Zufallsvariablen X, die auf $\{1, \ldots, 2^n\}$ gleichverteilt ist. Dann ist die Entropie von X gerade

$$H(X) = -\frac{1}{2^n} \sum_{x=1}^{2^n} \mathrm{ld}\left(\frac{1}{2^n}\right) = -\frac{2^n(-n \,\mathrm{ld}\, 2)}{2^n} = n$$

– gerade die Anzahl der Fragen, die wir schlimmstenfalls brauchen. Ohne Gleichverteilung sinkt die Entropie, da wir „geschickter“ fragen können, wenn wir wissen, dass bestimmte Zahlen sehr unwahrscheinlich sind (genauer beschäftigt sich damit die Codierungstheorie). Im Extremfall – es wird immer dieselbe Zahl x_0 genommen – ist $P(X = x_0) = 1$ und 0 sonst; die Entropie ist dann

$$H(X) = -P(X = x_0) \,\mathrm{ld}\, P(X = x_0) = -1 \cdot 0.$$

Wir brauchen gar keine Information mehr, um das Ergebnis zu kennen.

Man könnte verbieten, dass Null-Wahrscheinlichkeiten in den Verteilungen auftreten, es ist aber äquivalent und bequemer, $0 \log 0 = 0$ zu *definieren*.

„Die Entropie ist der negative Erwartungswert der Logarithmen der Wahrscheinlichkeiten einer Verteilung“

Entropie von Sprache I

Sprachmodell I: X hat Werte aus ASCII, Buchstabe für Buchstabe. Wir schätzen $P(X)$ aus gesammelten Artikeln der „Zeit“ mit 4.5×10^7 Zeichen und finden $H(X) = 4.499$.

Sprachmodell II: X entspricht Wörtern. Hier folgt $H(X) = 12.03$ aus 310791 verschiedenen Wörtern, was bei einer mittleren Wortlänge von 5.34 Buchstaben einer Entropie pro Buchstaben von 2.25 entspricht. Wie wir auf der nächsten Folie sehen werden, haben wir hier die reale Entropie vermutlich erheblich unterschätzt. Die Verwendung des Good-Turing-Schätzers für die Verteilung empfiehlt sich.

gzip-Test: Packer entfernen *Redundanz* (die definierbar ist als Differenz zwischen realer und maximaler Entropie; für byteorientierte Maschinen ist die maximale Entropie offenbar 8 bit pro Zeichen) aus Datenströmen. gzip -9 komprimiert den Korpus um einen Faktor 2.6, die ursprüngliche Entropie für gzip war also $8/2.6 = 3.1$.

Redundanz

Warum verschiedene Werte für $H(X)$ für den gleichen Datenstrom? Wir bestimmen nicht wirklich die Entropie des Datenstroms, sondern die einer Verteilung, die wir aus dem Datenstrom schätzen. Dieser Verteilung liegt ein Modell zugrunde, das mehr oder weniger falsch ist. Sowohl

für Wörter als auch für Buchstaben ist die Annahme gegenseitiger Unabhängigkeit aufeinanderfolgender X_i falsch, für Buchstaben offenbar falscher als für Wörter. Das Kompressionsprogramm gzip nutzt unter anderem diese Nicht-Unabhängigkeit, um Redundanz zu entfernen.

Entropie pro Zeichen

18. Entropie II

Leitfragen zu Abschnitt 18

- Warum ist die „Berechnung“ von Entropien ein riskantes Geschäft?
- Wie baut man Teilwissen in die Entropie ein?

Entropie von Sprache II

Einfachste Modellierung von Abhängigkeiten der Zeichen in einem Text: Folge $X_1, \ldots, X_n$ i.i.d. Zufallsvariablen mit Werten in den *n-grammen*. Es wird in der Regel auf die *Entropie pro Zeichen* normiert, die resultierende Entropie also durch n geteilt.

Warnung: Die Entropie von n-gram-Modellen ist schon für mäßige n praktisch nicht naiv zu berechnen, da die Verteilungen nur aus enormen Datenmengen zuverlässig zu schätzen sind.

Die maximale Entropie einer Verteilung über N elementaren Ausgängen ist die der Gleichverteilung, nämlich

$$-\sum_i = 1^N \frac{1}{N}(-\,\mathrm{ld}\, N) = \mathrm{ld}\, N.$$

Um auf eine Entropie von 12 zu kommen, bräuchte man also schon im „günstigsten“ Fall $2^{12} = 4096$ elementare Ausgänge, bei der Schätzung der Entropie pro Wort also verschiedene Wörter (Word Types). Wollte man eine Entropie von 12 bei einer realistischeren Nichtgleichverteilung erreichen, bräuchte man natürlich noch weit mehr verschiedene Zeichen (deren Wahrscheinlichkeiten dann aber auch noch mit einiger Zuverlässigkeit geschätzt werden müssen, was die Zahl der nötigen Word Tokens im Beispiel weiter in die Höhe treibt).

Wenn man einfach nur Wörter aus einem Text sammelt und daraus naiv die Verteilung schätzt, braucht man sehr lange Texte – unser Brumaire-Text etwa hat nur rund 6000 verschiedene Wörter, und selbst wenn deren Häufigkeiten „richtig“ schätzbar wären, ist, weil wir eben nicht Gleichverteilung haben, nicht mit einer zuverlässigen Schätzung der wortbasierten Entropie des Englischen zu rechnen.

Übrigens hat Shannon als einer der Väter der Informationstheorie lange bevor es Rechner gab, die solche Schätzungen halbwegs zuverlässig hätten durchführen können, die wortweise Entropie des Englischen recht gut durch das „Shannon game“ abgeschätzt: Man liest einen Teil eines Textes und lässt dann Menschen raten, was das nächste Wort wohl sei. Aus dem Verhältnis von richtigen zu falschen Antworten lässt sich die Entropie (über die Perplexität, s.u.) bestimmen.

(cf. Fig. 13)

Die Grafik oben zeigt den Verlauf der geschätzten Entropie für zufällig generierte Strings mit 16 verschiedenen Symbolen, die alle gleiche Wahrscheinlichkeiten haben – die erwartete Entropie ist also immer $\mathrm{ld}\, 16 = 4$. Die verschiedenen Linien entsprechen Strings verschiedener Länge, also sozusagen der Korpusgröße.

Man sieht, dass alle Strings die 1-gram-Entropie ganz gut schätzen lassen. Schon für Bigramme liegt der 100er-Schätzer grob daneben. Es ist schon prinzipiell nicht möglich, dass er richtig schätzt, denn wir haben darin nur rund 100 Bigramme, also 100 elementare Ausgänge des Zufallsexperiments. Selbst wenn kein Bigramm doppelt vorkäme (was möglich ist, denn es gibt insgesamt $16^2 = 256$ Bigramme über einem Alphabet von zwei Zeichen), wäre die Entropie

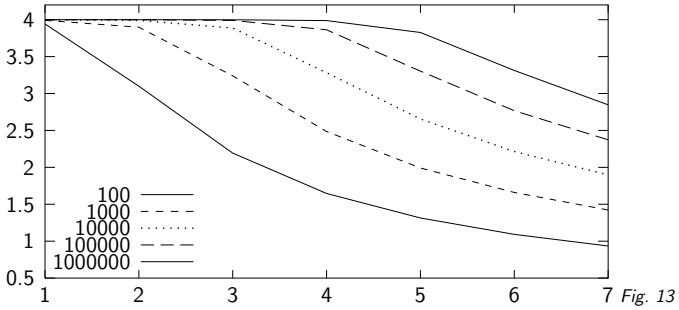

Fig. 13

bedingte Entropie

der resultierenden Verteilung nur $\mathsf{ld}\,100 = 6.64$. Die „wahre“ Entropie der Verteilung ist demgegenüber $4 \cdot 2 = 8$. Trigramme werden noch vom 10000er-Schätzer halbwegs ordentlich behandelt. Das ist in diesem Fall auch zu erwarten, denn die maximale Entropie für 10000 gleichverteilte Ergebnisse, $\mathsf{ld}\,10000 = 13.29$, liegt noch gut über der erwarteten Entropie $4 \cdot 3 = 12$. Bei 5-Grammen reichen dann auch 10^6 Zeichen nicht mehr aus (selbst im unwahrscheinlichen günstigsten Fall nicht doppelt vorkommender 5-Gramme wäre das so, denn $\mathsf{ld}\,10^6 = 19.93$ ist immer noch kleiner als $4 \cdot 5 = 20$).

All diese Rechnungen gelten für die besonders freundliche Gleichverteilung mit nur 16 Zeichen. Reale Texte haben selbst nach großzügiger Vereinfachung deutlich mehr als 16 Symbole, und die Zahl der möglichen n-gramme geht exponentiell mit der Zahl der Symbole. Die Sprache hilft zwar ein wenig dadurch, dass die meisten n-gramme gar nicht auftreten (jedenfalls, solange man sich innerhalb einer Sprache bewegt) und daher die Gesamtzahl zu schätzender Wahrscheinlichkeiten nicht ganz 27^n (oder was immer) ist, die aus obiger Figur sprechende Warnung gilt jedoch unvermittelt weiter – in der Tat habe ich das Experiment mit $n = 16$ gemacht, damit es überhaupt ein Referenzmodell gibt, für das die Entropie mit so naiven Methoden bestimmt werden kann.

Empirische Ergebnisse für einen Deutschen Text:

Unigramm	Bigramm	Trigramm
4.7	4.2	3.7

Modellierung von Abhängigkeiten

Will man Abhängigkeiten komplexer als durch n-gramme modellieren, brauchen wir Entropie für beliebig abhängige Z.V.s.

Ausgangspunkt: Sei $P(x, y)$ die gemeinsame Verteilung von X und Y. Die gemeinsame Entropie von X und Y ist dann

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y)\, \mathsf{ld}\, P(x, y).$$

Daraus ableitbar: Die *bedingte Entropie*

$$H(Y \mid X) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y)\, \mathsf{ld}\, P(y \mid x)$$

Wie kommt man darauf? Wir können schon die Entropie einer Verteilung einer Zufallsvariablen ausrechnen, also etwa

$$H(Y \mid x) = -\sum_{y \in \mathcal{Y}} P(y \mid x)\, \mathsf{ld}\, P(y \mid x).$$

Die Additivität der Entropie liefert dann $H(Y \mid X) = \sum P(x) H(Y \mid x)$, was nach Einsetzen der Definition der bedingten Wahrscheinlichkeit – beachte $P(x, y) = P(X = x \cap Y = y)$ – und Ausklammern das obige Ergebnis liefert.

Kettenregel der Entropie

Es gilt die *Kettenregel der Entropie*:

$$H(X, Y) = H(X) + H(Y \mid X).$$

Das ist mit der Definition der bedingten Wahrscheinlichkeit $P(x, y) = P(x) P(y \mid x)$ zu vergleichen – der Logarithmus macht auch hier eine Summe aus deinem Produkt.

Aufgaben

(18.1)* Holt euch das Skript entropy.py von der Webseite zur Vorlesung. Seht euch darin die Funktion computeEntropy an und macht euch klar, dass sie, wenn dist eine als Dictionary modellierte Verteilung ist, wirklich deren Entropie zurückgibt.

Anmerkung: Das log aus dem math-Modul ist der natürliche Logarithmus, der zur Basis $e = 2.71\ldots$ gerechnet wird. Die Division durch log(2) sorgt dafür, dass das Ergebnis das ist, das wir bekommen hätten, wenn wir von vorneherein den Zweierlogarithmus genommen hätten.

(18.2)* Seht euch die Funktion probB an und lasst sie laufen, wenn ihr verstanden habt, was hier passiert. Verändert die Verteilung darin und seht euch an, wie die Entropie der Verteilung sich mit euren Manipulationen verändert. Ihr solltet euch überzeugen, dass die Gleichverteilung die höchste Entropie hat, und die Entropie fällt, je weiter ihr von der Gleichverteilung weggeht. Überzeugt euch, dass die Entropie einer „Verteilung“, die überhaupt nur für einen Wert eine von Null verschiedene Wahrscheinlichkeit hat, Null ist.

(18.3) Studiert die Funktion getRandomString. Überzeugt euch anhand der Dokumentation zum random-Modul von Python – http://docs.cl.uni-heidelberg.de/python/lib/module-random.html – dass sie in der Tat einen „Text“ zurückgibt, dessen Symbole asymptotisch so verteilt sind wie die im „template“, das einfach nur eine Sequenz von Symbolen ist. Ihr könnt, wenn ihr im Template ein Symbol mehrmals verwendet, auch von der Gleichverteilung der Symbole abgehen. Lasst euch ein paar Zufallstexte erzeugen, verwendet dabei auch n-gram-Templates wie etwa `["der", "die", "das"]`.

(18.4)* Wir wollen jetzt die Entropie der von getRandomString erzeugten Texte bestimmen und sehen, wie gut wir die Entropie der erzeugenden Verteilung aus der Beobachtung des generierten Strings bestimmen können. probD bereitet das vor. Probiert auch andere Templates als die vorgegebenen, erzeugt auch größere oder kleinere Samples. Wann ist die Übereinstimmung zwischen wahrer Entropie (aus der Verteilung des Templates) und geschätzer Entropie (aus dem Sample) gut, wann schlecht? Warum funktioniert das so schlecht, wenn man n-gram-Templates verwendet?

(18.5) Eine „unabhängige“ Schätzung der Entropie mit einem wesentlich verschiedenen Modell (und etlichen weiteren Hindernissen, so dass das hier nicht zu ernst genommen werden sollte) bekommt ihr von gzip. probD schreibt das aktuelle Sample immer in eine Datei sample.txt. Komprimiert diese mit

```
gzip -9 sample.txt
```

und vergleicht die Größe der Datei vor und nach der Kompression. Eine Abschätzung der Entropie ergibt sich daraus, dass im gzip-Resultat die Entropie pro Zeichen 8 sein sollte – idealerweise würde gzip eine Datei erzeugen, in der alle 256 Bytes gleich wahrscheinlich sind – probiert das aus, im Interpreter:

```
from entropy import *
computeEntropy(getDistributionFromSeq(open("sample.txt.gz").read()))
```

Demnach ist die Entropie der Ausgangsdatei $8\text{bytes}_{\text{komprimiert}}/\text{bytes}_{\text{unkomprimiert}}$ Die gzip-Geschichte funktioniert für große Samples besser. In der Tat sollte für n-gram-Verteilungen die gzip-Entropie besser zur template-Entropie passen als die Unigramm-Entropie aus dem sample, vor allem, wenn n groß ist.

(18.6) Baut das Experiment mit den n-grammen über 16 gleichverteilten Zeichen nach. Schätzt dann aber die Verteilungen nicht mit Maximum Likelihood, sondern mit Good-Turing. Was stellt ihr bezüglich der Qualität der Entropieschätzung fest?

Datei(en) im PDF-Anhang: entropy.py

Leitfragen zu Abschnitt 19

- Wie kann man die Entropie nutzen, um etwas über Abhängigkeiten verschiedener Zufallsvariablen herauszubekommen.
- Wie kann man die Entropie nutzen, um herauszufinden, wie gut geschätzte Verteilungen sind?

19. Entropie III

relative Entropie
Transinformation
Mutual Information
noisy channel model
Kullback-Leibler Divergence
Kreuzentropie

Mutual Information

Wegen der Kettenregel ist $H(X) - H(X|Y) = H(Y) - H(Y|X)$. Diese Größe heißt *relative Entropie*, *Transinformation* oder meistens *Mutual Information* $I(X;Y)$. Sie ist offenbar 0 für unabhängige X, Y und wächst mit dem Grad der Abhängigkeit wie auch der Entropie der einzelnen Zufallsvariablen. Schließlich ist $I(X;X) = H(X)$.

Zur konkreten Berechnung der Mutual Information kann

$$I(X;Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P(x,y) \operatorname{ld} \frac{P(x,y)}{P(x)P(y)}$$

dienen.

Eine Anwendung der Mutual Information ist das *noisy channel model*. Dabei überträgt man Information (modelliert durch eine Zufallsvariable X) über einen Kanal, der dem Signal „Rauschen“ hinzufügt und bekommt am anderen Ende eine Zufallsvaribale Y heraus. Beispiele für solche Kanäle können Funkstrecken sein (etwa zu Mobiltelefonen), ein Zyklus von Drucken und Scannen, aber auch Übersetzung oder der Weg von gesprochener Sprache zur Ausgabe des Spracherkenners.

Ziel eines Übertragungssystems ist nun, dass Y so viel wie möglich mit X „gemeinsam“ hat, also die Mutual Information zu maximieren. Es ist klar, dass keine Informationsübertragung mehr möglich ist, wenn X und Y unabhängig sind – das ist gerade der Fall $I(X;Y) = 0$. Ist die Übertragung perfekt, so ist $X = Y$, damit also $I(X;Y) = H(X)$, was bedeutet, dass alle Information, die in X steckte, auch in Y steckt.

Kullback-Leibler Divergence

Ebenfalls gern als relative Entropie bezeichnet wird die *Kullback-Leibler Divergence*

$$D(P\|Q) = \sum_{x\in\mathcal{X}} P(x) \operatorname{ld} \frac{P(x)}{Q(x)},$$

die die Verschiedenheit zweier *Verteilungen* über der gleichen Zufallsvariablen beschreibt. Sie wächst mit wachsender Verschiedenheit von P und Q, kann also die Qualität einer approximierten Verteilungsfunktion messen. Vorsicht: Die KLD ist nicht symmetrisch!

Bei der Bildung einer Modellverteilung Q für eine Zufallsvariable X mit „wahrer“ Verteilung P möchten wir, dass die KLD von Q und P möglichst gering wird. Wir kennen aber P nicht.

Umweg: Die *Kreuzentropie* eines mit P verteilten X und einer Verteilung Q ist

$$\begin{aligned} H(X,Q) &= H(X) + D(P\|Q) \\ &= -\sum_{x\in\mathcal{X}} P(x) \operatorname{ld} Q(x). \end{aligned}$$

Man kann zeigen, dass die Kreuzentropie wächst, je weniger das Q – die Schätzung – mit P übereinstimmt.

Perplexität
punktweise Entropie
ergodisch

Für Nachrichten $L(n) = x_1, \ldots, x_n$, die nach P gezogen wurden, ist die Kreuzentropie pro „Zeichen“

$$H(L,Q) = -\lim_{n\to\infty} \frac{1}{n} \sum_{L(n)} P(L(n)) \operatorname{ld} Q(L(n)).$$

Vorsicht: P und Q sind hier Verteilungen auf kompletten Nachrichten. In der Regel wird man natürlich nur z.B. Bi- oder Trigrammodelle haben, aber es ist klar, dass diese eben auch kompletten Texten Wahrscheinlichkeiten zuweisen. Leider sind wir unser unbekanntes P immer noch nicht losgeworden.

Für ein stationäres, ergodisches Sprachmodell „mittelt“ sich das P aber heraus, und es gilt

$$H(L,Q) = -\lim_{n\to\infty} \frac{1}{n} \operatorname{ld} Q(L(n))$$

Diese Größe ist empirisch berechenbar (wir können Q schätzen), und ihre Minimierung führt zu einer Minimierung der KLD von Q und P. Hinter diesem Schritt steckt einige tiefe Weisheit, mit der wir uns auch nicht auseinandersetzen wollen.

Die *Perplexität* einer Schätzung ist $2^{H(L,Q)}$.

Es gibt eine andere Interpretation dieser Größe: Nehmen wir an, wir hätten ein Sprachmodell $P(w \mid h)$, das zu jeder Vorgeschichte h die Verteilung des nächste Worts/Zeichens w ergibt. Für festes w und h nennen wir $H(w \mid h) = -\operatorname{ld} P(w \mid h)$ die *punktweise Entropie*.

Die punktweise Entropie bemisst, wie schlecht eine Beobachtung zu einem Modell passt, also sozusagen die „Überraschung“ unter dem Modell. Ist z.B. in einem Bigramm-Modell sicher, dass nach einem q ein u kommt, so ist $P(\mathtt{u} \mid \mathtt{q}) = 1$; der Logarithmus ist 0, entsprechend sind wir nicht überrascht, wenn wir das Wort qualitätsmanagement sehen (was insgesamt eher beunruhigend ist). Wenn nun aber die Marketing-Abteilung der Sirius Cybernetics Corporation beschließt, dass Überraschung den Verkauf fördert und qwest für einen tollen Namen hält, wird unser System versuchen, den Logarithmus aus $P(\mathtt{w} \mid \mathtt{q}) = 0$ zu ziehen und seine „unendliche“ Überraschung durch einen Laufzeitfehler kundtun.

Summiert man die punktweisen Entropien über eine ganze Nachricht, bekommt man ein Maß für die Wahrscheinlichkeit dieser Äußerung unter unserem Modell – wir werden so etwas ähnliches später beim Forward-Algorithmus von Markow-Ketten kennenlernen. Und eben dies ist unsere empirische Kreuzentropie.

Nochmal: Es wäre natürlich besser, die KLD angeben zu können, da wir ja eigentlich nur sagen wollen, wie falsch wir mit unserer Schätzung liegen. In realen Situationen kennen wir aber weder $P(X)$ noch $H(X)$ (also die Entropie des Phänomens, das wir modellieren wollen), und daher ist die Kreuzentropie oder die Perplexität die beste Größe, um die Qualität des Modells zu bemessen. Darin wird natürlich immer die Entropie des zugrundeliegenden Modells stecken, aber wir wissen immerhin, wann unser Modell besser geworden ist.

Dem oben verwendeten Begriff „stationär“ werden wir bei Markow-Ketten nochmal begegnen: Die Wahrscheinlichkeiten $P(w \mid h)$ ändern sich nicht mit der „Zeit“ (also der Position im Datenstrom). Mit *ergodisch* bezeichnen wir demgegenüber Prozesse, bei denen „Zeitmittel gleich Scharmittel“ ist. Das meint, dass in einer langen Nachricht (Zeit) die Elemente tatsächlich so verteilt sind, wie das die Verteilungsfunktion (Schar) erwarten lässt. Prozesse können nichtergodisch sein, wenn sie „Senken“ (wenn also z.B. das Modell nur noch b erzeugt, wenn es einmal b erzeugt hat) oder Zyklen (nach einem b kann nur noch ababab. . . kommen) enthalten.

Exkurs: Algorithmische Information

Ein alternativer Informationsbegriff ist der der algorithmischen Entropie oder Kolmogorov-Komplexität. Dabei definiert man als Informationsgehalt einer Nachricht die Zahl der Bits, die man mindestens braucht, um eine universelle Turingmaschine so zu programmieren, dass sie die Nachricht reproduziert.

Auch dabei wird die Entropie nicht plötzlich eine der Nachricht innewohnende Eigenschaft – man muss sich zunächst darauf einigen, wie man die universellen Turingmaschine programmiert. Allerdings kann man hier zeigen, dass sich verschiedene Maße algorithmischer Komplexität nur um eine additive Konstante unterscheiden – die zwar groß sein kann, aber doch nur additiv. In diesem Sinn geht die algorithmische Informationstheorie viel mehr auf die Nachricht selbst ein als unsere Shannon-Theorie.

Eine Möglichkeit, Turingmaschinen zu beschreiben, haben wir in den formalen Grundlagen der Linguistik kennengelernt. Einigt man sich auf diese Beschreibung, die mit Nullen und Einsen auskommt, reduziert sich das Problem der Bestimmung der algorithmischen Information darauf, die kürzeste Beschreibung (also sozusagen das kürzeste Programm) zu finden, das die Nachricht produziert und dann zu zählen, wie viele Nullen und Einsen darin vorkommen. Leider ist es in der Regel unmöglich, tatsächlich zu zeigen, dass eine Beschreibung die kürzestmögliche ist.

Allgemein gilt, dass die algorithmische Information einer Nachricht nicht berechenbar ist (das lässt sich ganz analog zur Nichtentscheibarkeit des Halteproblems beweisen). Trotzdem lassen sich mithilfe der algorithmischen Information viele wichtige Theoreme beweisen – die aber leider nicht hierher gehören.

Der Zusammenhang zwischen Shannon'scher Entropie und Kolmogorov-Komplexität ist subtil, auch wenn er natürlich existiert. So kommen beispielsweise beide Begriffe zum Ergebnis, dass ich im Mittel $\mathrm{ld}\, n$ bits brauche, um gleichverteilte Zahlen zwischen 1 und n zu übertragen – auch Turing-Maschinen können nicht hexen.

20. Testtheorie I

Leitfragen zu Abschnitt 20

- Inwieweit laufen Test- und Schätztheorie parallel, wo unterscheiden sie sich?
- Was will man beim Testen rauskriegen, was kann man dabei falsch machen?
- Was ist die Gütefunktion und warum ist sie zentral zum Verständnis der Grundlagen der Testtheorie?

Bei einem *Test* geht es darum, eine Hypothese zu prüfen, etwa, dass „Bundeskanzler“ und „Schröder“ eine Kollokation (in unserem Sinn) bilden.

Formal: Wir haben die Wertemenge $\mathcal{X}$ einer Zufallsvariablen X und eine Menge $\{P_\vartheta \,|\, \vartheta \in \Theta\}$ von möglichen Verteilungen von X. Die *Hypothese* $H \subset \Theta$ ist eine echte Teilmenge von Θ, ihr Komplement $K = \Theta \setminus H$ heißt *Alternative*. Der Test ist eine Vorschrift, ob $\vartheta \in H$ oder nicht. Im ersten Fall spricht man von der *Annahme* der Hypothese, im zweiten Fall vom *Verwerfen*.

Wie schon in der Schätztheorie wird jedem $x \in \mathcal{X}$ ein $\vartheta \in \Theta$ zugeordnet. In „fertigen“ Tests will man in der Regel das ϑ nicht mehr sehen und gibt deshalb lieber eine Menge $R \subset \mathcal{X}$ an, für die die Hypothese verworfen wird (den *kritischen Bereich* oder *Verwerfungsbereich*).

Wenn $\mathcal{X} \subset \mathbb{R}$ (das wird praktisch immer so sein), so wird R häufig durch eine Funktion T beschrieben, so dass $R = \{x \in \mathbb{R} \,|\, T(x) \geq t\}$. Die Funktion T heißt *Teststatistik*, die Zahl t ist eine vom Test abhängige Funktion des gewünschten Niveaus, die man sich entweder vom Computer ausrechnen lässt oder aus einer Tabelle abliest.

Test
Hypothese
Alternative
Annahme
Verwerfen
kritischen Bereich
Verwerfungsbereich
Teststatistik

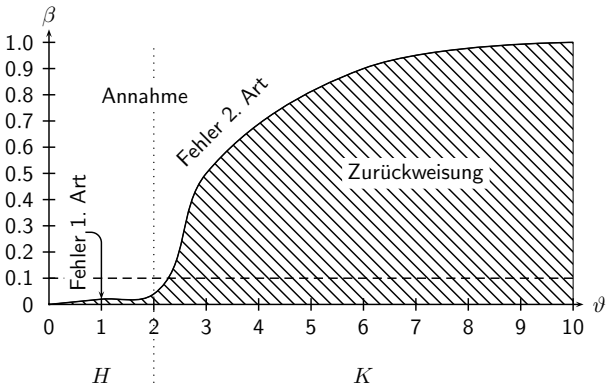

Fig. 14

Fehler erster Art
Fehler zweiter Art
Fehler dritter Art
Gütefunktion
Niveau
Macht
Nullhypothese

Ein *Fehler erster Art* liegt vor, wenn die Hypothese verworfen wird, obwohl sie richtig ist, ein *Fehler zweiter Art*, wenn die Hypothese angenommen wird, obwohl sie falsch ist, ein *Fehler dritter Art*, wenn das Modell falsch war.

Die *Gütefunktion*

$$\beta(\vartheta) = P_\vartheta(X \in R)$$

gibt zu jedem ϑ die Wahrscheinlichkeit, dass H verworfen wird. Wenn $\beta(\vartheta) \leq \alpha$ für alle $\vartheta \in H$, hat der Test das *Niveau* α. Wenn $\vartheta \in K$, heißt $\beta(\vartheta)$ *Macht*. Die Wahrscheinlichkeit für einen Fehler erster Art ist höchstens das Niveau, die für einen Fehler zweiter Art höchstens $1-$Macht.

Warum ist das so? Nun, wenn $\vartheta \in H$, dann sagt $X \in R$, dass die Hypothese verworfen wird, obwohl sie wahr ist – das ist der Fehler erster Art. Ist aber $\vartheta \in K$, ist $X \in R$ gerade das, was idealerweise herauskommen sollte; falsch liegen wir, wenn wir das Gegenteil entscheiden, die Fehlerwahrscheinlichkeit (für einen Fehler zweiter Art) ist dann also $1 - P(X \in R)$.

Grundsätzlich ist diese Formel der Schlüssel zum Verständnis der Aussage von Tests. Zur Verdeutlichung mag die folgende Grafik dienen:

(cf. Fig. 14)

Hier ist ein (erfundenes) $\beta(\vartheta)$ geplottet. Im schraffierten Bereich wird zurückgewiesen (er gibt nach Definition von β für jedes ϑ die Wahrscheinlichkeit der Zurückweisung der Hypothese), im weißen Bereich angenommen. Links von der vertikalen gepunkteten Linie ist die Hypothese, rechts die Alternative (es ist durchaus denkbar, eine kompliziertere Topologie zu haben, etwa eine Hypothese, die auf beiden Seiten von der Alternative eingeschlossen ist – man würde dann von einem zweiseitigen Test sprechen). Demnach entspricht der schraffierte Bereich über H dem Fehler erster Art, der weiße Bereich über K dem Fehler zweiter Art.

Man ahnt auch, dass man in der Regel (und jedenfalls nicht bei kontinuierlichem ϑ und stetigem β) nicht gleichzeitig global Macht und Niveau kontrollieren kann – will man mehr Niveau, sinkt die Macht und umgekehrt. In der Praxis kontrolliert man eben den Fehler erster Art. Im Beispiel bleibt β über H immer unter 0.1 (die gestrichelte waagrechte Linie), und so haben wir hier einen Test zum Niveau 0.1 (tatsächlich sogar zum Niveau 0.075, aber auch zu 0.4 usf.

Weil wir das Niveau kontrollieren und nicht die Macht, müssen wir unsere Tests so konstruieren, dass uns Fehler 2. Art nicht sehr jucken: Formulieren einer *Nullhypothese* H_0, die das „Unüberraschende“ behauptet: Gleichverteilung, Unabhängigkeit etc. Danach muss man nur noch die Teststatistik berechnen, Grenze zu Niveau aus Tabelle oder Programm ablesen.

Eine typische Wahl des Niveaus ist $\alpha = 0.05$ (dahinter steckt nicht viel Weisheit – es ist einfach ein Kompromiss, auf den man sich in vielen Disziplinen geeinigt hat). Das bedeutet insbesondere:

Wir müssen damit rechnen, dass eine von zwanzig Zurückweisungen der Nullhypothese fehlerhaft ist.

Dass die Macht eines Tests nicht global kontrollierbar ist, heißt nicht, dass nicht auch sie eine nützliche Größe wäre. Häufig ist es nämlich so, dass man gerne Effekte *einer bestimmten Größe* ausschließen möchte. Um das zu illustrieren, möchte ich kurz *den* Klassiker der Testtheorie seit den fundamentalen Werken von Fisher, Pearson und Neyman in der ersten Hälfte des 20. Jahrhunderts einführen:

Die Tea Testing Lady

Eine etwas schrullige Landadelige aus den Südwesten Englands behauptet, sie könne genau schmecken, ob bei ihrem Tee zunächst die Milch (wir sind in England – Tee ohne Milch zu trinken, kommt natürlich nicht in Frage) oder zuerst der Tee eingegossen wurde. Das klappt selbstverständlich nicht immer, aber doch meistens.

Um jetzt zu prüfen, ob diese Behauptung besser in eine Kategorie mit Schlossgespenstern und freier Marktwirtschaft zu stecken oder ob vielleicht doch etwas dran sei, kann man die Lady einfach etliche Tassen klassifizieren lassen und sehen, wie viel Erfolg sie hat. Wenn es egal ist, was zuerst in die Tasse kommt, sollten ihre Erfolge zufällig kommen, und zwar mit einer Erfolgswahrscheinlichkeit von 0.5. Nimmt man an, dass die einzelnen Versuche unabhängig voneinander sind, so müsste die Zahl ihrer Erfolge binomialverteilt sein.

Die Nullhypothese ist das, was die Lady gern widerlegen würde: $p = 0.5$. Die Alternative ergibt sich dann als $p > 0.5$, dass nämlich die Lady öfter als durch reinen Zufall bestimmt richtig liegt. Den Fall $p < 0.5$ ignorieren wir – formal könnten wir $\Theta = [0.5, 1]$ setzen.

Wenn ihr nun die Aufgaben bearbeitet, findet ihr, dass ein Test mit 20 Versuchen und der Zahl der Erfolge als Teststatistik mit einem kritischen Wert von 15 noch ein Niveau von 0.05 hat (genauer sogar 0.02). Ihr seht allerdings auch, dass der Test ziemlich unfair gegenüber der Lady ist, weil beispielsweise $\beta(0.7)$ auch nur gut 0.4 ist, wir also nur in gut 40% der Fälle die Nullhypothese zurückweisen würden, wenn die Lady immerhin 70% der Tassen richtig klassifizieren würde – was sicher als deutlicher Hinweis auf Forschungsbedarf zu werten wäre. Hier ist also die Macht des Tests nicht ausreichend.

Man kann nun die Macht des Tests einfach durch Verschiebung des kritischen Werts erhöhen – mit $t = 12$ etwa würden wir die Nullhypothese bei einer $p = 0.7$-Lady zu fast 90% zurückweisen und mithin glauben, dass etwas an ihrer Behauptung dran ist. Leider geht dabei das Niveau vor die Hunde, und wir würden mit einer Wahrscheinlichkeit von fast einem Drittel Scharlataninnen für Feinschmeckerinnen halten.

Der einzige Weg, das Niveau zu halten und die Macht zu erhöhen ist, die Fallzahl hochzupumpen. Mit einer Fallzahl von 80 und einem kritischen Wert von 49 haben wir immer noch ein Niveau von 0.05, haben aber unsere Macht bei $p = 0.7$ auf über 0.95 gesteigert.

Bei realen Testdesigns fragt man darum häufig: Wie viele Fälle brauche ich, um einen bestimmte Effektgröße noch mit einer bestimmten Macht behandeln zu können. Fertig gepackte Tests kommen häufig mit Vorschriften zur Berechnung von Fallzahlen.

Datei(en) im PDF-Anhang: simpletest.py trytest.py

Aufgaben

(20.1)* Holt euch das Programm simpletest.py von der Webseite zur Vorlesung. Studiert den Quellcode ein wenig. Die GUI-Klassen sind nicht sehr spannend, wesentlich ist eigentlich nur die Klasse ExactBinomialTest. Untersucht die getPowerFunction-Methode und erklärt, wie da die Gütefunktion (power function) berechnet wird.

Startet das Programm, seht euch die Gütefunktion an, erklärt noch einmal, was dargestellt ist und spielt ein wenig mit den beiden Parametern, die in der Fußzeile stehen, herum (sie werden übernommen, wenn wir im Feld Return drückt, wenn ihr das Feld z.B. mit Tab verlasst oder wenn ihr auf „Set“ drückt).

Angewandt auf das Problem der Tea Testing Lady steht n dabei für die Zahl der Tassen, die die Lady im Test klassifizieren muss, t für den kritischen Wert der Teststatistik „erfolgreich klassifizierte Tassen“.

(20.2)* Setzt zunächst $n = 10$. Verändert den kritischen Wert t, bis ihr einen Test mit dem Niveau $\alpha = 0.05$ habt. Was ist t? (Tipp: rechts oben könnt ihr die aktuelle Cursorposition in logischen Koordinaten ablesen). **(L)**

(20.3)* Seht jetzt, wie ihr die Macht dieses Tests ablesen könnt. Macht euch am Bild nochmal klar, dass aus Stetigkeitsgründen $\min_\vartheta(\beta(\vartheta)) = \beta(t)$ gilt. Ihr könnt aber sehr wohl sagen, wie groß die Wahrscheinlichkeit ist, dass wir der Lady glauben, wenn sie in Wirklichkeit 60, 70, 80 oder 90% der Tassen richtig klassifiziert. Wie groß sind diese Wahrscheinlichkeiten mit dem Test aus der letzen Aufgabe?

(20.4)* Konstruiert jetzt einen Test mit $\alpha = 0.05$, dessen Macht reicht, um in 95% aller Versuche eine Lady, die sich nur bei einem Drittel aller Tassen täuscht, als „Könnerin“ klassifiziert.

(20.5)* Guckt, ob euer Test auch funktioniert. Besorgt euch dazu das Programm trytest.py von der Vorlesungs-Webseite. Seht euch dessen Quellcode an, versteht, was es tut und seht dann nach, ob euer Test das erwartete Verhalten zeigt (man muss zugeben, dass diese Prüfung letztlich zirkulär ist und mehr ein Test für den Zufallszahlengenerator von Python ist als für euer Testdesign).

21. Identifikation von Kollokationen

Leitfragen zu Abschnitt 21

- Warum ist die Identifikation von Kollokationen eine Anwendung der Testtheorie? Worauf testen wir?
- Was muss man beim t-Test rechnen? Was sind seine Annahmen, was seine Aussage?
- Was muss man beim χ^2-Test rechnen? Wann sollte man ihn besser nicht verwenden?

Wir definieren $P(w)$ als die Wahrscheinlichkeit, dass das Wort w auftaucht, $P(w_1w_2)$ als die Wahrscheinlichkeit, dass w_2 hinter w_1 steht. Eine Formalisierung der Nullhypothese „w_1 und w_2 stehen *nicht* besonders häufig nacheinander“ könnte sein $P(w_1w_2) = P(w_1)P(w_2)$.

Sehen wir, ob in unserem Brumaire-Text „Louis Bonaparte“ überhäufig ist (d.h., wir nehmen an, die ML-Schätzungen der Worthäufigkeiten aus den Zählungen in dem Text gäben ein brauchbares Wahrscheinlichkeitsmaß – was eigentlich Unsinn ist, aber für ein Beispiel taugt). Der Brumaire-Text enthält mit einem naiven Tokenisierer 42570 Wörter und aus Wortzählungen folgt

$$P(\text{Louis}) = \frac{44}{42570} = 0.00103$$
$$P(\text{Bonaparte}) = \frac{167}{42570} = 0.00392$$
$$P(\text{Louis Bonaparte}) = \frac{13}{42570} = 0.000305.$$

Damit haben wir $0.00103 \cdot 0.00392 = 4.04 \times 10^{-6}$ als Schätzung für die Wahrscheinlichkeit von „Louis Bonaparte“ unter Annahme der Unabhängigkeit der beiden Wörter, was mit der tatsächlich beobachteten Wahrscheinlichkeit von 3.09×10^{-4} zu vergleichen ist. Offenbar sind die Werte verschieden (was natürlich nicht überrascht). Reicht aber die Verschiedenheit, um auf eine Überhäufigkeit des Bigramms (und damit auf eine Kollokation in unserem Sinn) zu schließen?

Nehmen wir „party“ und „you“ als Beispiel für ein Wortpaar, das sicher über jedem Kollokationsverdacht steht:

$$P(\text{party}) = \frac{164}{42570} = 0.00385$$
$$P(\text{you}) = \frac{70}{42570} = 0.00164$$
$$P(\text{party you}) = \frac{1}{42570} = 2.35 \times 10^{-5}.$$

Es ist $P(\text{party})P(\text{you}) = 6.63 \times 10^{-6}$. Es scheint, als sei auch „party you“ deutlich überhäufig. Es ist aber jedenfalls mal „weniger überhäufig“.

Wenden wir also unsere Testtheorie an, im Wissen, dass unsere Stichprobe so klein ist, dass wir uns hart am Rand des zulässigen befinden und darüber hinaus natürlich auch unsere ML-Schätzungen wenig Wert haben.

Häufig wird zur Klärung solcher Fragen der *t-Test* angewandt. Dabei hat man N normalverteilte Zufallsvariablen X_i. Zu testen ist, ob ein gegebener Erwartungswert μ_0 der Stichprobe entspricht oder nicht. Die Teststatistik ist dann

$$t = \frac{\sqrt{N}(\bar{X} - \mu_0)}{\sqrt{s(X)^2}},$$

worin $\bar{X}$ und $s(X)^2$ unsere Schätzer für Erwartungswert und Standardabweichung sind.

Mit der Teststatistik geht man in eine Tabelle, die zu einem gegebenen N und α den kritischen Wert liefert.

Anwendung auf unser Beispiel: Die X_i sind definiert über einem Ω aller möglichen Bigramme, so dass $X_i = 1$, wenn das gesuchte Bigramm auftaucht und $X_i = 0$ sonst (man nennt ein X_i dieser Art auch gern *Indikatorvariable* für das Ereignis „gesuchtes Bigramm gefunden“). Dies ist effektiv ein Bernoulliexperiment, das, wenn Louis und Bonaparte unkorrelliert wären, gerade mit $p = 4.04 \times 10^{-6}$ verteilt sein müsste und entsprechend auch $\mu_0 = 4.04 \times 10^{-6}$ haben sollte – das jedenfalls ergibt die beste Annäherung durch eine Normalverteilung, die bei diesem p allerdings garantiert nicht atemberaubend gut sein wird. Weiter ist $\bar{X} = 0.000305$ (nämlich die beobachtete Häufigkeit von „Louis Bonaparte“), $N = 42570$ (genau genommen hätte das letzte Wort keinen Partner fürs Bigramm, es gibt also ein Bigramm weniger als es Wörter gibt), und unser Schätzer für s^2 ergibt

$$\frac{1}{42570}\left(42557 \cdot \left(0 - 4.04 \times 10^{-6}\right)^2 + 13 \cdot \left(1 - 4.04 \times 10^{-6}\right)^2\right) = 0.000305$$

(Übung: Macht euch klar, warum $\bar{X}$ und $s(X)^2$ hier so nahe beieinanderliegen).

Unsere Teststatistik berechnet sich damit zu $t = 3.56$; wenn wir zu einem Niveau von $\alpha = 0.05$ bestimmen wollen, dass Louis Bonaparte hier eine Kollokation darstellt, müssen wir laut Tabelle (vgl. fast beliebige Tabellenwerke, etwa auch am Ende von Manning-Schütze) über 1.645 kommen. Wir nehmen damit in Kauf, dass wir uns unter zwanzig Vermutungen einen Fehler erster Art machen. Das ist angesichts der sonstigen Unwägbarkeiten (die X_i sind sicher nicht unabhängig, und normalverteilt sind sie auch nicht – Fehler dritter Art) gar nicht schlecht. Wir können also die Nullhypothese der Unabhängigkeit von Louis und Bonaparte zum Niveau von 0.05 zurückweisen.

Wenn wir die gleiche Prozedur für „party you“ durchziehen (tut es!), kommen wir auf eine Teststatistik von $t = 0.71$, was deutlich unter der Grenze von 1.645 liegt und uns nicht erlaubt, die Nullhypothese zu verwerfen.

Der t-Test unterstellt, dass X normalverteilt ist. Für unsere Zufallsvariablen gilt das fast nie. Deshalb populärerer Test: χ^2-Test. Unter diesem Namen firmieren etliche verwandte Verfahren. Wir behandeln hier nur den Test auf Unabhängigkeit. Ihnen gemeinsam ist, dass sie schwächere Annahmen über die beteiligten Zufallsvariablen machen als der t-Test.

t-Test

Indikatorvariable

Idee: Vergleiche die Zahl der erwarteten Fälle mit der der beobachteten Fälle, so dass das die Teststatistik groß wird, wenn die Abweichungen von der Erwartung groß werden. Dazu Tabelle der Zahl der Bigramme w_1w_2 mit

	$w_1 = \text{L.}$	$w_1 \neq \text{L.}$	
$w_2 = \text{B.}$	13	154	167
$w_2 \neq \text{B.}$	31	42372	42403
	44	42426	42570

Die Teststatistik ist

$$V^2 = \sum_{i,j} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}},$$

wobei $O_{i,j}$ den beobachteten Wert in der i-ten Zeile und der j-ten Spalte und $E_{i,j}$ den erwarteten Wert an dieser Stelle bezeichnen. Die $E_{i,j}$ berechnen wir aus der Marginal„verteilung“.

Dies entspricht einem ML-Schätzer. So ist $E_{1,1}$ einfach das Produkt der ML-Wahrscheinlichkeiten ganz unten in der ersten Spalte und ganz rechts in der ersten Zeile, wieder zurückgewandelt in eine absolute Häufigkeit,

$$E_{1,1} = (44/42570 \cdot 167/42570) \cdot 42570 = 0.173$$

Die komplette Teststatistik ist, nach dieser Systematik berechnet, $V^2 = 958$. Was bedeutet das? Zunächst nicht viel, unter bestimmten Umständen (die hier übrigens eigentlich nicht erfüllt sind, siehe unten) kann man aber annehmen, dass V^2 einer χ^2_n-Verteilung folgt (die Details sind da irrelevant, n ist jedoch die Zahl der Freiheitsgrade, in einer $N \times M$-dimensionalen Tabelle ist das $(N-1)(M-1)$). Wenn das so ist, kann man in einer Tabelle nachsehen und finden, dass in unserem Fall der kritische Wert für V^2 zum Niveau 0.05 bei gerade mal 3.84 liegt, wir haben also wieder gefunden, dass die Nullhypothese, Louis und Bonaparte seien unkorreliert, nicht angenommen werden kann (bzw. zurückgewiesen wird).

Für „party“ und „you“ sieht unsere Tabelle so aus:

	$w_1 = \text{p.}$	$w_1 \neq \text{p.}$	
$w_2 = \text{y.}$	1	163	164
$w_2 \neq \text{y.}$	69	42337	42406
	70	42500	42570

Damit folgt ein $V^2 = 1.99$ – auch der χ^2-Test findet, dass die Nullhypothese der Unabhängigkeit für „party you“ nicht verworfen werden kann.

Allerdings haben wir bei allen Untersuchungen an dem kleinen Brumaire-Text einen Fehler dritter Art gemacht, wir haben nämlich an zu wenig Daten gearbeitet, als dass wir den Tests ernsthaft Vertrauen entgegenbringen könnten.

Im Fall des χ^2-Tests gibt es eine gute Faustregel, ab wann die Test wohl vertrauenswürdig sein werden: Wenn eine Erwartungszelle eine Zahl ≤ 2 enthält, ist der Test nicht mehr „gut“, und man sollte sich etwas anderes Ausdenken. Hat man viele Freiheitsgrade, etwa $n > 10$, mag auch eine Grenze von 1 für die Erwartungen ausreichen.

Wirklich überzubewerten ist das alles nicht – all die vorberechneten Tests machen Annahmen über Verteilungen, die den Beobachtungen zugrunde liegen, und diese Annahmen sind im Bereich der Sprachverarbeitung eigentlich immer verletzt. Eine genauere Untersuchung dieser Fragen findet sich etwa in Pedersen, Kayaalp und Bruce (1996)[3].

[3] http://citeseer.ist.psu.edu/pedersen96significant.html

Aufgaben

(21.1)* Besorgt euch von der Webseite zur Vorlesung das Skript `chisquare.py` (die beiden anderen Skripte braucht ihr nicht).

Vollzieht nach, wie `computeChisquare` und `_computeExpectations` gemeinsam χ^2 berechnen und wie `runOneTest` die Kontingenztabelle baut. Entspricht das dem Vorgehen aus der Vorlesung?

Die Funktionen `lchisqprob` und Freunde könnt ihr ignorieren – sie berechnen so in etwa die Grenzen, die normalerweise aus den Tabellen kommen. Vgl. die nächste Aufgabe.

(21.2) Wir sehen zur Beurteilung der Teststatistik hier nicht in einer Tabelle von kritischen Werten nach, sondern verwenden gleich so genannte p-values. Diese geben das kleinste Niveau, zu dem die Nullhypothese noch verworfen würde. Sie bedeuteten natürlich keine „Irrtumswahrscheinlichkeit“ oder so etwas, das einzige, was legal mit ihnen gemacht werden darf, ist, sie gegen ein vorher festgelegtes Niveau zu vergleichen (d.h., wenn der p-Value kleiner als 0.05 ist, kann die Nullhypothese zum Niveau von 0.05 verworfen werden).

Besorgt euch eine Tabelle von kritischen Werten des χ^2-Tests und ruft `lchisqprob` mit den dort gegebenen kritischen Werten auf (am besten in einer interaktiven Session der Art

```
import chisquare
chisquare.lchisqprob(3.84, 1)
```

Versteht ihr die Ausgabe?

Wer zu faul ist, eine Tabelle zu suchen, hier ein paar Daten aus Manning-Schütze:

Für einen Freiheitsgrad:

Niveau	0.1	0.05	0.01	0.001
Kritischer Wert	2.71	3.84	6.63	10.83

Für 4 Freiheitsgrade (`df=4`):

Niveau	0.1	0.05	0.001
Kritischer Wert	7.78	9.49	18.47

Wenn euch interessiert, was `lchisqprob` eigentlich tut, findet ihr bei Press: Numerical Recipies erschöpfende Auskunft.

(21.3)* Besorgt euch einen Korpus (auf der Webseite gibt es einen, aber der ist eigentlich etwas klein). Für diese Zwecke kann er eigentlich gar nicht zu groß sein (wohl aber für euren Rechner...) Das Programm kann auch ge-gzippte Texte lesen (wenn ihre Namen auf `.gz` enden). Kommentiert die Zeile mit `runCollocTest` ganz unten ein und lasst das Programm mit dem Namen eures Korpus als Argument laufen. Ihr könnt jetzt jeweils zwei Wörter eingeben und sehen, ob sie „Kollokationen“ sind.

In der Regel werden die Tests wohl an Spareseness leiden – seht selbst, ob ihr die Ergebnisse vernünftig findet. Ein paar Vorschläge, was ihr mit dem Korpus von der Webseite so versuchen könnt; erklärt jeweils, warum das Programm zu dem ausgebenen Schluss kommt: (ist kein), (wohl nicht), (rektor gut), (uni heidelberg), (heidelberg stadt), (stadt heidelberg), (neues schloss) – Anmerkung: wenn ihr Wörter mit Umlauten verarbeiten wollt, müsst ihr ein deutsches Locale haben; mit der GNU libc (ihr ihr vermutlich benutzt, wenn ihr etwas entfernt Unixähnliches habt) etwa könnt ihr die Umgebungsvariable `LC_ALL` dazu auf `de_DE` setzen.

Wenn ihr einen kleinen Rechner oder wirklich große Korpora habt, könnt ihr auch die alternative Implementation des Kollokationsprogramms in `chisquare-large.py` verwenden.

(21.4)* Kommentiert runCollocTest wieder aus und dafür showSortedCollocs ein. Pipet die Ausgabe des Skripts durch more (oder less) und seht euch an, welche Bigramme das Programm für Kollokationen hält und welche nicht. Erklärt die Entscheidungen des Programms.

(21.5) Im letzten Versuch dürften viele Bigramme wie „das ist“ als „Kollokationen“ bezeichnet worden sein. Damit haben wir uns natürlich von allen sinnvollen Kollokationsbegriffen weit entfernt und sehen vor allem syntaktische Phänomene (die natürlich auch nicht uninteressant sind – hier seht ihr den Grund für die in der Regel ganz ordentliche Leistung von Markov-Taggern).

Damit man etwas interessantere Ergebnisse bekommt, kann man die häufigsten Wörter ausschließen. Dafür hat `showSortedCollocs` das Argument `freqThreshold`. So, wie das hier gemacht ist, gibt diese Schwelle an, dass Wörter, die häufiger `N/freqThreshold`, nicht für Kollokationen berücksichtigt werden. Setzt freqThreshold mal auf 500 oder 1000 und seht, was dann passiert. Leider ist mit Korpora in den Größen, wie wir sie mit diesem Programm verarbeiten können, da kein Blumentopf mehr zu gewinnen...

Datei(en) im PDF-Anhang: chisquare.py chisquare-large.py fileiter.py unimut.all.gz

Leitfragen zu Abschnitt 22

- Was sind Likelihood-Quotienten-Tests?
- Ist es eine gute Idee, Tests und Scoring zu kombinieren?

stop words

Quantile

22. Testtheorie II

Eine andere Anwendung des χ^2-Tests ist der Vergleich von Texten: Sind zwei Texte aus dem gleichen Grundkorpus gezogen oder nicht?

Unsere Tabelle sieht jetzt so aus:

	$\lvert w \in C_1 \rvert$	$\lvert w \in C_2 \rvert$
w = studis	292	24
w = wurde	290	58
w = heidelberg	290	84
⋮	⋮	⋮

Die Teststatistik bleibt unverändert, ist jetzt aber ein Maß für die „Abweichung vom Dreisatz“ (d.h., idealerweise wäre für aus dem gleichen Korpus gezogene Texte das Verhältnis der Zahlen in beiden Spalten über alle Zeilen konstant).

Problematisch ist die Auswahl der Wörter, die in die Tabelle aufgenommen werden. Die häufigsten Wörter sind typischerweise nicht sehr korpusspezifisch (der, und, in...), seltene Wörter stören den Test. Denkbar: *stop words* (also manueller Ausschluss insignifikanter Wörter, verbunden mit Auschluss seltener Wörter), *Quantile* (etwa: Die 5% häufigsten und die 85% seltensten Wörter werden ausgeschlossen).

Ein Beispiel: Zwei Texte aus dem UNiMUT aktuell (Artikel vor und nach dem 10.9.2001), Ua und Un, ein Satz von Pressemitteilungen des Pressesprechers des Rektors der Uni Heidelberg, P, ein Artikel aus der Schülerzeitung MSZ, MS, und der schon erwähnte 18. Brumaire (englisch), B.

Wir wählen jeweils alle Wörter, deren Erwartung in den beiden zu vergleichenden Texten > 4 ist. Leider wird sich die Zahl dieser Wörter je nach Paar drastisch unterscheiden, was die Anwendung der Testtheorie erschwert. Wir wollen nur ein Gefühl für die Verschiedenheit der Texte bekommen und dividieren deshalb das V^2 durch die Zahl der verwendeten Wörter (also etwa die Zahl der Freiheitsgrade). Da die kritischen Werte im χ^2-Test nicht linear ansteigen, ist das *nicht* streng richtig.

Das Ergebnis, jeweils als Verschiedenheit von Ua:

Text	Un	MS	P	B
V^2	2.27	2.70	14.4	144

Es werden also sehr schön sowohl verschiedene Texttypen (SchülerInnen-, Studierendenpresse vs. Presseerklärungen von Rektoren) als auch verschiedene Sprachen (Deutsch vs. Englisch) unterschieden. Bei genauerer Ausführung des *Tests* allerdings wird man finden, dass ohne große Tricks (eher linguistischer Natur) alle verwendeten Texte zu fantastischen Signifikanzniveaus verschiedenen Korpora angehören. Das allerdings liegt zu guten Stücken daran, dass die Behauptung, wir hätten bei so einem Vergleich so viele Freiheitsgrade wie word types, grundfalsch ist. Die Frage, wie die Zahl der Freiheitsgrade wirklich bestimmt werden sollte, ist allerdings schon wieder viel komplizierter.

Likelihood-Quotienten-Tests

Hintergrund der meisten Tests ist der *Likelihood-Quotient*

$$q(x) = \frac{\sup\{L_x(\vartheta) \mid \vartheta \in K\}}{\sup\{L_x(\vartheta) \mid \vartheta \in H\}}.$$

Ein Test φ ist ein L.-Q.-Test, wenn mit einem bestimmten $c \in \mathbb{R}$ die Hypothese für $q(x) < c$ angenommen, für $q(x) > c$ verworfen wird.

Der L.-Q. ist sozusagen das Verhältnis zwischen der Wahrscheinlichkeit der besten Erklärung für die Beobachtung von x, wenn die Alternative zutrifft und der Wahrscheinlichkeit der besten Erklärung, wenn die Hypothese zutrifft. Wenn letztere groß ist, ist $q(x)$ klein, die Hypothese wird also angenommen.

Dass in der Formel das Supremum (sup) und nicht ein Maximum über die Menge der Werte der Likelihood-Funktion L_x für alle ϑ genommen wird, liegt daran, dass ϑ häufig reell sein wird, die Menge also überabzählbar sein kann. Solche Mengen müssen nicht unbedingt ein Maximum haben, selbst wenn sie beschränkt sind – die Menge $M = \{x^2/(x^2-1) \mid x \in \mathbb{R}\}$ etwa ist eine Teilmenge von $[0,1]$, und sicher sind alle Elemente < 1. Die Eins ist jedoch nicht in ihr enthalten. Umgekehrt gibt es zu jeder Zahl $y < 1$ garantiert ein $z \in M$, so dass $z > y$, so dass 1 die kleinste Zahl ist, die jedes Element von M majorisiert. Das ist gerade das Supremum. Wem solche Subtilitäten zuwider sind, darf sich da ein Maximum vorstellen.

Sowohl der χ^2- als auch t-Test sind L.-Q.-Tests, sie unterscheiden sich nach der Vorschrift, nach der c berechnet wird – wie das in einem relativ komplizierten Sinn optimal zu geschehen hat, hängt insbesondere von den zugrundeliegenden Verteilungen ab.

Der Likelihood-Quotient kann unter der Annahme, dass die Bigramme in der Tat binomialverteilt sind, für unser Kollokationsbeispiel recht einfach hingeschrieben werden – wir brauchen dann zunächst gar keine Annahmen mehr über irgendeine Sorte asymptotischen Verhaltens, wie sie für die Anwendbarkeit von t- und χ^2-Test nötig sind. Dieses Verfahren ist in unseren Kreisen als Dunning's Test bekannt.

Dazu formulieren zur Entscheidung, ob zwei Wörter x und y Kollokationen sind über unserem üblichen Bigrammodell folgende Hypothesen:

- H_1 – $P(w_2 = y \mid w_1 = x) = p = P(w_2 = y \mid w_1 \neq y)$ (Unabhängigkeit, keine Kollokation)
- H_2 – $P(w_2 = y \mid w_1 = x) = p_1 \neq p_2 = P(w_2 = y \mid w_1 \neq y)$ (Kollokation)

Jetzt schätzen wir die darin vorkommenden Wahrscheinlichkeiten per ML zu $p = |y|/N$, $p_1 = |xy|/|x|$ und $p_2 = (|y| - |xy|)/(N - |x|)$.

Die wesentliche Annahme ist nun, dass die Frequenzen aus einer Binomialverteilung generiert werden, was im Wesentlichen heißt, dass die Bigramme unabhängig voneinander gezogen werden (auch diese Annahme ist natürlich falsch, aber nachweisbar „weniger falsch“ als die, die hinter den anderen Tests stehen).

Man kann dann die Likelihoods der beiden Hypothesen berechnen; üblicherweise berechnet man gleich den Logarithmus des Verhältnisses, also

$$\log \lambda = \log \frac{L(H_1)}{L(H_2)} = \log \frac{b(|xy|; |x|, p) b(|y| - |xy|; N - |x|, p)}{b(|xy|; |x|, p_1) b(|y| - |xy|; N - |x|, p_2)}.$$

Man *könnte* jetzt auf die Idee kommen, sich für H_1 zu entscheiden, wenn dieses $\log \lambda$ größer als Null ist (weil dann das Verhältnis selbst größer als eins und mithin der Nenner größer als der Zähler ist).

Ohne weiteres wäre das fatal, weil die Hypothesen normalerweise nicht gleichberechtigt sind. In der Testtheorie hat man gerade deshalb Nullhypothese und Alternative eingeführt, und man möchte normalerweise schon „starke Evidenz“ haben, um die Nullhypothese zu verwerfen – „ein bisschen wahrscheinlicher“ reicht uns nicht. Dazu kommt, dass hier H_1 und H_2 nicht die gleiche Zahl freier Parameter haben – für H_1 kann man nur eine Zahl, p, wählen, für H_2 hat man zwei Zahlen. Es ist aber klar, dass ich eine gegebene Beobachtung immer werde besser beschreiben können, wenn ich mehr Parameter zur Anpassung meiner Theorie auf die Beobachtung habe. Man

würde nun vielleicht gerne die Likelihood von H_2 irgendwie „bestrafen“, um das auszugleichen – aber so etwas macht man besser in einem Bayesianischen Rahmen (vgl. unten). Dort heißt diese Sorte von Bestrafung MDL-Prior.

Im hier vorliegenden Fall arbeiten die beiden Effekte gegeneinander – die Nullhypothese der Unabhängigkeit wird zunächst immer wahrscheinlicher sein, weil sie nämlich zwei Parameter hat. Trotzdem hätte man gerne unsere mühsam erworbenen Instrumente (und vor allem das Niveau) auch hier. Tatsächlich ist das unter schwachen Annahmen möglich, denn die Größe $-2 \log \lambda$ ist wieder asymptotisch χ^2-verteilt, kann also als eine robustere Teststatistik in einem χ^2-Test dienen.

Dunnings Test ist generell besser für unsere Probleme als der normale, auf Kontingenztabellen basierende χ^2-Test. Trotzdem bleibt die Verwendung von $-2 \log \lambda$ ein asymptotisches Resultat, Wunder bei praktisch nicht vorhandenen Daten darf mensch also auch nicht erwarten. An der Einsicht, dass man nur aus Daten schätzen kann, die man auch wirklich hat, führt bei aller Testtheorie eben kein Weg vorbei.

23. Bayes'sche Statistik

Leitfragen zu Abschnitt 23

- Was ist ein Prior und warum ergibt er sich eigentlich sehr natürlich aus der Bayes'schen Formel?
- Was passiert beim Schätzen nach Bayes?

Bisher: Relative Häufigkeit im Zentrum der Überlegungen („Frequentismus“). Alternative: Bayesianismus.

Es gibt hitzige Debatten zwischen Frequentisten und Bayesianern. Trotzdem sollte man keinen allzu großen Gegensatz konstruieren, korrekt eingeführte Wahrscheinlichkeiten und überlegte Anwendung der Methoden führen mit dem einen wie mit dem anderen Ansatz zu „richtigen“ Ergebnissen. Allerdings hat der Bayesianismus gerade in der Computerlinguistik mit ihren häufig unbefriedigenden Daten und wenigstens teilweise durchaus vernünftig begründbaren Erwartungen einige Vorteile.

Grundidee ist, nicht zu glauben, es gebe eine richtige Hypothese und die Beobachtung auf diese Hypothese zu bedingen (Hintergrund von ML ist letztlich $\operatorname{argmax}_p P(o \mid \mu_p)$), sondern eher von der Beobachtung auszugehen und die Modelle auf diese Beobachtung zu bedingen, also etwa $P(\mu_p \mid o)$ zu betrachten.

Das ist zunächst grundvernünftig, denn immerhin wissen wir ja nur die Beobachtung mit Bestimmtheit – die Modelle kommen von uns, und auf sie zu bedingen heißt irgendwo, auf Phantasie zu bedingen. Aber das sollte nicht so ernst genommen werden, denn auch BayesianerInnen brauchen Modelle.

Viel dramatischer ist, dass Bayesianer eben auch Theorien oder Modellen „Wahrscheinlichkeiten“ zuschreiben, und das ist es, was die Frequentisten ärgert – letztere argumentieren nämlich, Wahrscheinlichkeiten könnten nur aus beliebig oft wiederholbaren Experimenten kommen, und in so ein Bild passen Modelle nicht rein. Tatsächlich ist schon eine Aussage wie „Die Regenwahrscheinlichkeit am nächsten Sonntag ist 30%“ für Frequentisten ein Graus, weil es eben nur einen nächsten Sonntag gibt (während die Regenwahrscheinlichkeit an Sonntag*en* in Ordnung wäre).

Als Nebeneffekt des mutigen Schritts, Wahrscheinlichkeiten so zu emanzipieren, können Bayesianer Theorien auch ohne Daten Wahrscheinlichkeiten zuordnen. Das klingt nach schwarzer Magie, ist aber häufig gut motivierbar. Occams Klinge ist beispielsweise ein uraltes und bewährtes Prinzip der Naturwissenschaften und sagt in etwa „Wenn du die Wahl zwischen einem einfachen und einem komplizierten Modell hast, nimm das einfache, wenns irgend geht“. Für uns könnte sowas

heißen: Bevorzuge ein Modell mit wenigen Parametern gegenüber einem mit vielen. Oder auch: Bevorzuge ein Modell ohne allzu wilde Annahmen gegenüber einem, das mehr Mut braucht.

Ein Modell ist wieder eine aus einer Familie von Verteilungen gewählte Verteilung. Für ein Bernoulli-Experiment mit n Versuchen wäre ein geeignetes Modell für die Zufallsvariable E „Zahl der Erfolge“

$$P(E = e \mid \mu_p) = \binom{n}{e} p^e (1-p)^{n-e}.$$

Ausgehend hiervon fragt der Frequentist nach dem Modell, das die Wahrscheinlichkeit der Beobachtung $P(e \mid \mu_p)$ maximiert. Wir das hatten in der Schätztheorie durch Betrachtung der Nullstellen der Ableitung (aus technischen Gründen der des Logarithmus) bestimmt:

$$\ln P(e \mid \mu_p) = \ln \binom{n}{e} + e \ln p + (n-e) \ln(1-p),$$

$$0 = \frac{d}{dp} P(e \mid \mu_p) = e/p - \frac{n-e}{1-p}.$$

Als Lösung der zweiten Gleichung hatten wir $p = e/n$ gefunden.

Letztlich sagt dies aber noch gar nichts – denn wir wissen, dass diese Antwort mit endlicher Wahrscheinlichkeit falsch ist. Um das handhaben zu können, hatten wir Konfidenzintervalle konstruiert, die so gemacht waren, dass wir die Wahrscheinlichkeit, unzutreffende Konfidenzintervalle zu erhalten, kontrollieren konnten, aber nicht in dem Sinn, dass die Fehlerwahrscheinlichkeit einer einzelnen Schätzung kontrollierbar war.

Wir möchten eigentlich lieber wissen, wie wahrscheinlich ein gegebenes Konfidenzintervall den konkreten Wert enthält:

$$P(\mu_p \mid E = e) = \frac{P(E = e \mid \mu_p) P(\mu_p)}{P(E = e)}.$$

Problem: Was ist $P(\mu_p)$, was ist $P(E = e)$?

Die erste Frage bietet FrequentistInnen reichlich Gelegenheit zu Breitseiten: Kann man überhaupt Wahrscheinlichkeiten von Modellen definieren? (Nebenbei: Hier geht das natürlich unproblematisch, denn wir können mühelos eine Verteilung auf z.B. der Menge der Parameter Θ definieren – in allgemeineren Fällen, z.B. dem Wetter am nächsten Wochenende ist das nicht so unproblematisch) Und vor allem: Was sind vernünftige Annahmen für $P(\mu_p)$?

Wenn wir $P(\mu_p)$ kennen, ist $P(E = e)$ einfach, nämlich die Marginalwahrscheinlichkeit über alle möglichen Modelle – es ist nicht ganz verkehrt, sich das als Erwartungswert vorzustellen:

$$\begin{aligned} P(E = e) &= \sum_p P(E = e, \mu_p) \\ &= \sum_p P(E = e \mid \mu_p) P(\mu_p). \end{aligned}$$

$P(\mu_p)$ muss sozusagen a priori angenommen werden, daher heißt $P(\mu_p)$ auch *Prior*. Darin drückt sich dann eine Erwartung an das Modell aus; denkbar ist Gleichverteilung, Gleichverteilung über einem Intervall (Haar prior), Polynome, Gauss-Verteilungen usf.; diese Verteilungen lassen sich häufig streng ableiten oder wenigstens motivieren, ebenso häufig aber nicht.

Besonders nett sind so genannte non-informational Priors, deren Ziel ist, nicht mehr in die Beobachtungen reinzustecken, als wirklich drin ist. Die Gleichverteilung ist so ein Prior, oft nützlicher und erkenntnistheoretisch motiviert hingegen ist eine mathematische Umsetzung von Occams Klinge namens MDL, ausgeschrieben *minimum description length*.

Ausgehend davon können wir den für eine gegebene Beobachtung wahrscheinlichsten Parameter bestimmen als $\operatorname{argmax}_p P(\mu_p)$.

Prior

minimum description length

Nehmen wir für $P(\mu_p)$ zunächst Gleichverteilung an. Dann gilt $\frac{d}{dp} P(\mu_p) = 0$, weiter ist hier ohnehin immer $\frac{d}{dp} P(E = e) = 0$ (in der Summe oben wird das p wegsummiert, bevor das $\frac{d}{dp}$ es sehen könne – ähnliches gilt, wenn wir, wie hier eigentlich, mit kontinuierlichen Dichten arbeiten und statt Summen Integralzeichen hätten). Damit ist

$$\frac{d}{dp} P(\mu_p \mid e) = \text{const} \frac{d}{dp} P(e \mid \mu_p).$$

Da wir zur Schätzung von p nur die Ableitung brauchen und die Nullstellen auf der rechten und linken Seite gleich sein werden, ist Bayesianismus mit einem gleichverteilten Prior ziemlich äquivalent zum Frequentismus.

Nehmen wir aber an, wir hätten einen guten Grund, bestimmte Werte von p zu favorisieren. Für Bigramme beispielsweise haben wir gesehen, dass der Schluss von der Abwesenheit eines Bigramms im Korpus auf dessen Unmöglichkeit grober Unfug ist – und Bayesianismus lässt uns genau die Überzeugung, dass viele Bigramme eben doch möglich sind, obwohl wir sie nicht beobachten, in eine Theorie packen.

Der Einfachheit halber wollen wir im Beispiel daran glauben, dass $p = 0.5$ ein richtig guter Wert sei – sagen wir, für einen Münzwurf; die Beobachtung ist dann nach wie vor die Zahl der Erfolge e. Alle anderen (zwischen Null und Eins) sind aber auch irgendwie gut. Ein analytisch einfacher Prior in dieser Situation ist

$$P(\mu_p) = \begin{cases} 6p(1-p) & 0 < p < 1 \\ 0 & \text{sonst} \end{cases}$$

– an den „Klebestellen“ dieser zusammengesetzten Funktion passieren keine schlimmen Dinge, wir brauchen für die Dichte keine Differenzierbarkeit. Das ist keine schlechte Dichte: Ihr Maximum ist wie gewünscht bei 0.5, und normiert ist sie auch. Sie drückt aber kein sehr starkes Vertrauen in unsere Erwartung $p = 0.5$ aus, weil sie relativ breit ist. Nun ist

$$\frac{d}{dp} P(\mu_p \mid E = e) = \frac{1}{P(E = e)} \frac{d}{dp} \Big(P(E = e \mid \mu_p) P(\mu_p) \Big).$$

Wieder ist es technisch angenehmer, die Ableitung des Logarithmus auszurechnen – die Lage des Maximums verschiebt sich dadurch nicht. Die zu lösende Gleichung lautet dann

$$\frac{1}{p} - \frac{1}{1-p} + \frac{e}{p} - \frac{n-e}{1-p} = 0$$

Eine Lösung davon ist

$$\hat{p} = \frac{1+e}{n+2}.$$

Interessant daran ist, dass dieser Schätzer mehr „Trägheit“ hat als unser ML-Schätzer. Wenn wir 10 Erfolge in 10 Versuchen haben, sagt ML $p = 1$, unser Schätzer aber $p = 11/12$. Das muss keine bessere Schätzung sein, erinnert aber daran, dass natürlich auch die fairste Münze mal nur Kopf liefern mag und eben nicht klar ist, dass Erfolg das sichere Ereignis ist. Ebenso gehen wir bei 0 Erfolgen nicht gleich auf $p = 0$ wie ML, sondern eben nur auf $p = 1/12$, bei acht Erfolgen auf $p = 0.75$ statt auf $p = 0.8$. Wenn wir einen stärker gepeakten Prior genommen hätten, wären wir auch weniger weit zum ML-Resultat marschiert, hätten wir einen schwächer gepeakten Prior genommen, wären wir näher zum ML-Resultat gekommen. Darin steckt eine gewisse Beliebigkeit, aber wie gesagt: Häufig gibt es gute Gründe, sich für einen bestimmten Prior zu entscheiden.

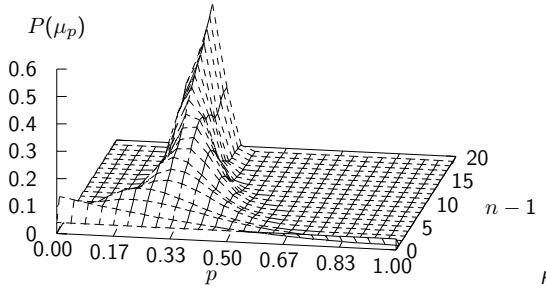

Fig. 15

24. Mehr Bayesianismus

Leitfragen zu Abschnitt 24

- Wie funktioniert Bayesian Updating?
- Wie fällen Bayesianer Entscheidungen? Was ist MAP?

Bayesian Updating

belief intervals

Bayesian Updating

Wenn wir über eine Familie von Priors verfügen, können wir unseren Prior den Daten anpassen: *Bayesian Updating*. Idee dabei: nach jeder Beobachtung ω wird $P_{\mathsf{neu}}(\mu_p) = P_{\mathsf{alt}}(\mu_p \mid \omega)$.

Das geht mit diskreten Verteilungen als Prioren, was den Vorteil hat, dass man sich viel hässliche Mathematik erspart. Im folgenden Bild wurde von einer Gleichverteilung auf 28 Werten für p zwischen 0 und 1 ausgegangen und für 20 Bernoulli-Experimente mit $p = 0.33$ die Verteilung $P(\mu_p)$ per Bayesian Updating geschätzt.

(cf. Fig. 15)

Will man wie im Beispiel den Parameter eines Bernoulli-Experiments schätzen, muss man nur $P_{\mathsf{neu}}(p) = b(n; N, p)P_{\mathsf{alt}}(b)$ ausrechnen (die Zahl der Erfolge n kommt natürlich aus einem Experiment, z.B. einer wiederholten Auswertung eines Zufallszahlengenerators, die Zahl der „Würfe“ N in einem Experiment ist im Prinzip frei wählbar, aber natürlich konvergiert der Schätzer schneller, wenn N größer ist) und das ausreichend lange iterieren.

Man kann nun die resultierenden Verteilungen so aufsummieren, dass $\sum_{p=p_1}^{p_2} P(\mu_p) > 1 - \alpha$ und hat so etwas wie einen Schätzer zum Niveau α. Das Intervall $[p_1, p_2]$ entspricht dann in etwa unseren Konfidenzintervallen, aber nicht ganz, weshalb man diese Intervalle *belief intervals* nennt. Der Hit dabei ist, dass hier die „natürliche“ Interpretation – wonach unser Parameter mit einer Wahrscheinlichkeit von $1 - \alpha$ im Intervall liegt – richtig ist.

Maximum a posteriori

Bayes'sche Entscheidungen

Bayes hilft auch, Entscheidungen zwischen Hypothesen zu treffen. Eine Hypothese kann etwas sein wie $p > 0.3$ bei einem Bernoulli-Experiment. In der Regel wird man aber eine ganze Menge von Hypothesen $H = \{h_1, h_2, ...\}$ haben, die dann etwas wie $h_1 : 0 < p < 0.2$, $h_2 : 0.2 < p < 0.4$ usf. sein könnten. In diesem Sinne ist die (diskrete) Schätztheorie ein Spezialfall des Entscheidens – allerdings werden wir uns beim Entscheiden nicht für Belief Intervals interessieren, sondern nur für die plausibelste Hypothese.

Gegeben sei eine Beobachtung $D \subset \Omega$ sowie ein Prior auf der Hypothesenmenge $P(h)$. Die Hypothese gibt die Wahrscheinlichkeit für die Beobachtung von D. Wenn wir die wahrscheinlichste Hypothese suchen, müssen wir

$$\begin{aligned} h_{\mathsf{MAP}} &= \operatorname*{argmax}_{h \in H} P(h \mid D) \\ &= \operatorname*{argmax}_{h \in H} \frac{P(D \mid h)P(h)}{P(D)} \end{aligned}$$

berechnen.

MAP steht hier für *Maximum a posteriori*.

Ein klassisches Beispiel kann wieder die medizinische Diagnostik sein. Anknüpfend an unser Burizystose-Beispiel von der bedingten Wahrscheinlichkeit wollen wir jetzt den Hypothesenraum $H = \{g, k\}$ (für Gesund und Krank) untersuchen. Die Verbreitung der Krankheit in der Bevölkerung stellt sich hier als Prior dar, nämlich $P(g) = 0.999$ und $P(k) = 0.001$. Wir brauchen zur Berechnung noch die $P(D \mid h)$.

Das Experiment läuft hier ab über $\Omega = \{p_P, n_P\}$, nämlich dem positiven und negativen Ergebnis des Plutopharma-Tests. Nach der Problemstellung gilt

$$\begin{aligned} P(p_P \mid g) &= 0.01 & P(p_P \mid k) &= 0.9 \\ P(n_P \mid g) &= 0.99 & P(n_P \mid k) &= 0.1 \end{aligned}$$

(natürlich müssen sich die bedingten Wahrscheinlichkeiten auch zu eins ergänzen, was formal einfach nachzuweisen ist, aber auch anschaulich klar ist: Egal, ob jemand krank oder gesund ist, fällt der Test auf jeden Fall positiv oder negativ aus).

Wir rechnen jetzt den Posterior aus; $P(D)$ müssen wir erstmal nicht ausrechnen, weil sich das nachher ohnehin aus der Normalisierung ergibt. Nehmen wir an, p_P tritt ein, der Test ist also positiv.

$$\begin{aligned} P(g \mid p_P) &= P(p_P \mid g)P(g)/P(D) = 0.00999/P(D) \\ P(k \mid p_P) &= P(p_P \mid k)P(k)/P(D) = 0.0009/P(D) \end{aligned}$$

Nach Normalisierung ergibt sich:

$$P_{\mathsf{postP}}(g) = 0.91735537 \quad P_{\mathsf{postP}}(k) = 0.0826446281$$

– mithin ist die wahrscheinlichste Hypothese nach wie vor, dass der/die PatientIn gesund ist (klar, weil die Wahrscheinlichkeit, dass sie krank ist, eben so niedrig ist), aber unser Glaube an die Gesundheit ist angekratzt. $P(D)$ hätte man hier natürlich auch direkt berechnen können als $P(p_P \mid g) + P(p_P \mid k) = 0.01089$ – der Weg über die Normalisierung ist aber in Anwendungen meistens bequemer.

Meist muss $P(D)$ aber gar nicht berechnet werden, denn es ist bei Variation von h konstant und spielt damit in der Berechnung von argmax keine Rolle.

Das Tolle an Bayesianischen Methoden ist nun, dass sie die Integration von Evidenz ermöglichen. Wir können nämlich nach diesem ersten, hoffentlich billigen Test, einen besseren und teureren machen, sagen wir die Buropsie. Für sie könnte

$$\begin{aligned} P(p_B \mid g) &= 0.01 & P(p_B \mid k) &= 0.99 \\ P(n_B \mid g) &= 0.99 & P(n_B \mid k) &= 0.01 \end{aligned}$$

gelten. Wir können den Posterior aus der letzten Untersuchung als Prior der neuen verwenden und rechnen diesemal von vorneherein nichtnormalisiert; dabei soll der Test wieder positiv ausgefallen sein. Wir haben dann

$$\tilde{P}(g \mid p_B) = P_{\text{postP}}(g)P(p_B \mid g) = 0.00917 \qquad \tilde{P}(k \mid p_B) = P_{\text{postP}}(k)P(p_B \mid k) = 0.0818$$

– also allen Grund, alarmiert zu sein, denn nun ist unsere MAP-Hypothese mit weitem Vorsprung k, in Worten krank.

Beachtet, dass die zweite, die Nobeluntersuchung, alleine nicht zu diesem Schluss gereicht hätte. Setzt man den ursprünglichen Prior ein, ergibt sich hier

$$\begin{aligned}\tilde{P}(g \mid p_B) &= P(p_B \mid g)P(g) = 0.00999\\ P(k \mid p_B) &= P(p_B \mid k)P(k) = 0.00099,\end{aligned}$$

die MAP-Hypothese wäre also weiterhin „gesund“

Aufgaben

(24.1)* Besorgt euch den Quellcode von `discPrior.py` von der Webseite zur Vorlesung. Lest den Docstring von `ScaledArray` und macht euch klar, was diese Klasse eigentlich tut und inwieweit sie für einen Prior über kontinuierlichen Verteilungen geeignet ist.

(24.2)* Lest den Quellcode für `getBeliefInterval` und vergleicht ihn mit dem, was ihr bei der automatischen Berechnung der Konfidenzintervalle (`confInter.py`) analysiert habt.

(24.3)* Wir wollen hier den Parameter einer Binomialverteilung schätzen. Das tut die Klasse `BayesianUpdateBernoulliTester`. Sie wird konstruiert mit der Zahl der Diskretisierungspunkte und dem Parameter. Die `toss`-Methode gibt dann die Zahl der Erfolge bei `numberOfTosses` individuellen Bernoulli-Experimenten zurück. Könnt ihr sehen, warum? Macht euch klar, wie in `updateFor` die Zahl der Erfolge `numberSucc` bei `numberExp` Einzelexperimenten zur Berechnung des neuen Priors verwendet wird.

(24.4)* Die Funktion `runIt` steuert mit ihren unzähligen Argumenten einen Testlauf des `BayesianUpdateBernoulliTesters`. Macht euch klar, was die Argumente bedeuten, seht zu, wie sich die Konfidenzintervalle mit der Zeit verändern. Probiert verschiedene Parameter durch, lasst das Programm (für kleine `totalIters`) auch mehrmals mit identischen Parametern laufen. Erklärt die Beobachtungen.

(24.5) Ihr könnt euch die „zeitliche“ Entwicklung der Prioren auch ansehen. Das Programm schreibt – so, wie es hier gemacht ist – den aktuellen Prior jeweils in eine Datei `priors.log`. Mit bloßem Auge ist darin praktisch nichts zu erkennen, gnuplot kann die Daten aber per

```
splot "priors.log" matrix with lines
```

visualisieren. Seht euch an, wie das mit verschiedenen Parametern aussieht. Ihr könnt die Ausgabe noch verschönern, etwa mit Anweisungen wie

```
set hidden  # "hidden lines" werden nicht angezeigt
set view 60, 10  # Richtung, aus der der/die BeobachterIn guckt
set ticslevel 0 # lässt die z-Achse am "Boden" anfangen
set contour # Malt am Boden Höhenlinien
```

`help splot` kann euch weitere Ideen geben.

Datei(en) im PDF-Anhang: discPrior.py

25. Bayesianische Klassifikation

Leitfragen zu Abschnitt 25

- Was ist das Ziel von Klassifikation?
- Wie klassifiziert man mit Bayes-Mitteln?
- Was nimmt man bei der naiven Klassifikation an und warum macht diese Annahme das Leben so viel leicher?

Ziel der Klassifikation ist, bestimmte Versuchsausgänge einer von potenziell vielen Klassen zuzuordnen. Beispiele wären: Artikel in Klassen wie „Klatsch“, „Politik“, „Feuilleton“ einteilen, Mails in Spam oder Ham, Spektren in verschiedene Stern- oder Galaxiensorten, Betriebszustände in normal, kritisch und „Good Bye, Heidelberg“ und so weiter.

Formal wir haben Klassen $v_j \in V$ und „Trainingsdaten“ $D \subset \Omega \times V$ (also Paare von Beobachtungen und gewünschten Klassen, z.B. (Artikel1, Klatsch), (Artikel2, Politik) usf.). Wir haben weiter Hypothesen $h_i \in H$, deren jede eine Verteilung $P(v_j, \omega \mid h_i)$ gibt (also letztlich sagt, zu welcher Klasse v_j ein ω gehören soll). Die *bayes-optimale Klassfikation* eines Ergebnisses ω ist dann

$$\operatorname*{argmax}_{v_j \in V} \sum_{h_i \in H} P(\omega, v_j \mid h_i)P(h_i \mid D).$$

Im Groben addieren wir also für jedes v_j einfach die von allen Hypothesen h_i ausgespuckten Wahrscheinlichkeiten, dass ω von der Klasse v_j ist mal die Wahrscheinlichkeit der Hypothese selbst unter den gegenbenen Trainingsdaten – diese Rechnung ist richtig, weil die Ereignisse „v_j ist die korrekte Klassifikation“ disjunkt sind.

Bei nur zwei Hypothesen ist das Ergebnis des Klassifizierers natürlich das Gleiche, als hätte man zunächst die MAP-Hypothese bestimmt und dann nach dieser klassifiziert. Schon bei drei Hypothesen kann das aber anders sein. Sei etwa $P(h_{1,2,3} \mid D) = 0.4, 0.3, 0.3$. Wir könnten nun für ein ω etwas wie $P(1, \omega \mid h_1) = 1$ und $P(0, \omega \mid h_2) = P(0, \omega \mid h_3) = 1$ haben, d.h. die Hypothese h_1 klassifiziert sicher als 1, die beiden anderen sicher nach 0. Die MAP-Hypothese ist damit h_1, die danach wahrscheinlichste Klassifikation also 1. Unser Bayes-optimaler Klassifzierer würde hingegen

$$\begin{aligned}P(1, \omega \mid D) &= 0.4 \cdot 1 + 0.3 \cdot 0 + 0.3 \cdot 0 = 0.4\\ P(0, \omega \mid D) &= 0.4 \cdot 0 + 0.3 \cdot 1 + 0.3 \cdot 1 = 0.6,\end{aligned}$$

also 0 vorhersagen.

Man kann zeigen, dass es für einen gegebenen Hypothesenraum und gegebene Trainingsdaten keinen im Mittel besseren Klassifizierer gibt, dass also kein anderes Verfahren „meistens“ bessere Ergebnisse liefert.

Leider ist der Bayes-Optimale Klassifizierer – wie eigentlich die meisten „optimalen“ Verfahren in der Regel völlig unpraktikabel, weil – jedenfalls ohne weitere Annahmen – H gigantisch sein sollte (versucht, einen Hypothesenraum zur wortbasierten Klassifikation von Dokumenten in Dokumentenklassen zu bestimmen) und damit die Summe nicht realistisch berechenbar ist, von den Problemen, die $P(h_i \mid D)$ schätzen, ganz zu schweigen.

Naive Bayesian

Unser typisches Klassifikationsproblem ist die Zuordnung einer Sammlung von Werten $x = \langle a_1, \ldots, a_n \rangle$ zu einer Klasse $v \in V$. Diese Werte können etwa alle word tokens in einem Dokument sein oder die word tokens, die in der Umgebung eines bestimmten word types vorkommen.

Zur Klassifikation müssen wir lediglich

$$\operatorname*{argmax}_{v \in V} P(a_1, \ldots a_n \,|\, v) P(v \,|\, D)$$

ausrechnen. Wir haben hier die Notation etwas vereinfacht – wenn wir nämlich unsere Hypothesen so wählen, dass die Hypothese h_i alles als v_i klassifiziert, können wir Hypothesen und Klassen identifizieren. Formal sagen wir $P(v_j \,|\, h_i) = \delta_{ij}$. Dann sagt die Bayes-Optimale Klassifikation die Klasse

$$\begin{aligned}
\operatorname*{argmax}_{v_j \in V} \sum_{h_i \in H} P(\omega, v_j \,|\, h_i) P(h_i \,|\, D) &= \operatorname*{argmax}_{v_j \in V} \sum_{h_i \in H} P(\omega \,|\, h_i, v_j) P(v_j \,|\, h_i) P(h_i \,|\, D) \\
&= \operatorname*{argmax}_{v_j \in V} \sum_{h_i \in H} P(\omega \,|\, h_i, v_j) \delta_{ij} P(h_i \,|\, D) \\
&= \operatorname*{argmax}_{v_j \in V} P(\omega \,|\, h_i, v_j) P(v_i \,|\, D)
\end{aligned}$$

voraus. Dabei haben wir im ersten Schritt $P(a, b \,|\, c) = P(a \,|\, b, c) P(b \,|\, c)$ ausgenutzt (das folgt aus der Definition von bedingter Wahrscheinlichkeit, wenn man wieder bedenkt, dass „,“ hier einfach der Schnitt ist), im zweiten Schritt unsere Beziehung für $P(v_j \,|\, h_i)$ eingesetzt, und schließlich die Eigenschaft des Kronecker-Delta, dass $\sum_i j \delta_{ij} = j$ ist, ausgenutzt sowie freihändig $P(h_i \,|\, D) = P(v_i \,|\, D)$ geschrieben, was aber wegen der Identifikation von vs und hs harmlos ist.

Diese Vereinfachung hilft noch nicht viel: Schon, wenn die a_i aus kleinen Mengen kommen, gibt es furchtbar viele $a_1, \ldots, a_n$. Nehmen wir etwa an, wir hätten 10 Klassen und wollten auf der Basis von 1000 Wörtern klassifizieren. Außerdem sehen wir nur auf Vorkommen von Wörtern (und nicht auf ihre Frequenzen), so dass wir eben 1000 a_i haben, die die Werte 0 oder 1 annehmen können. Wie viele Werte von $P(a_1, \ldots, a_n \,|\, v)$ müssen wir schätzen? Das Urnenmodell (hier haben wir „Ziehung von 1000 Kugeln aus 2 mit Zurücklegen und mit Anordnung, also den Fall I) sagt uns, dass wir pro Klasse 1000^2 und mithin insgesamt 10^7 Parameter schätzen müssten. Das sind furchtbar viele. Deshalb:

Naive Bayesian Assumption:

$$P(a_1, \ldots, a_n \,|\, v) = \prod_{i=1}^{n} P(a_i \,|\, v)$$

Das heißt nichts anderes, als dass das Auftreten des Attributs a_i von dem des Attributs a_j unabhängig ist. Das ist für unsere typischen Aufgaben grober Unfug, wie schon unser Beispiel mit Bundeskanzler und Schröder bei der Unabhängigkeit gezeigt hat. Trotzdem funktioniert der Naive Bayesian Classifier

$$v_{\mathsf{NB}} = \operatorname*{argmax}_{v \in V} P(v) \prod_{i=1}^{n} P(a_i \,|\, v)$$

häufig erstaunlich gut.

Hier reicht zum Training die Schätzung der $n|V|$ Parameter $P(a_i \,|\, v)$.

Als Beispiel mag das Filtern von Spam dienen. V besteht dann aus den beiden Klassen Ham (Mails, die wir bekommen möchten) und Spam (also unerwünschte Werbemail). Die Attribute können zunächst alle Wörter sein, die wir in den Mails vorfinden.

Wir brauchen also zwei Sammlungen von Spam und Ham (das D, sozusagen unser Korpus) und schätzen (ML tut es hier) die Verteilungen $P(w \,|\, \text{Spam})$ und $P(w \,|\, \text{Ham})$ für alle word types w, die wir in D finden. Kommt nun eine neue Mail ω, berechnen wir $P(\omega, \text{Spam} \,|\, D)$ und $P(\omega, \text{Ham} \,|\, D)$ – die Zahlen werden typischerweise sehr klein werden, es empfiehlt sich

also, hier mit logarithmischen Wahrscheinlichkeiten zu operieren – und entscheiden uns für die wahrscheinlichere Klasse.

Das funktioniert, ist aber nicht so sonderlich gut, weil die Naive Bayesian Assumption von Mails und unseren Fragen an sie erheblich verletzt wird. Wie es etwas besser geht, steht auf der nächsten Folie.

26. Anwendung: Ein Spamfilter

Leitfragen zu Abschnitt 26

- Was sind die Anforderungen an einen Spamfilter und warum ist das ein Klassifkationsproblem?
- Wie kommt Einsicht über die Anwendungsdomäne in den Klassifikationsalgorithmus?

Besser als der Naive Bayesian Classifier zumindest zur Spamerkennung ist der um Heuristiken angereicherte Plan for Spam[4] von Paul Graham. Interessant ist dabei vor allem die Kombination von „first principles“ und Einsichten über den Objektbereich. Ich werde hier nur ein paar grundsätzliche Punkte behandeln, mehr ist auf der oben zitierten Seite zu finden.

Nehmen wir zunächst an, wir wollten nur mit einem einzigen Wort w zwischen S und nicht S klassifizieren. Dann ist einfach

$$P(S \,|\, w) = \frac{P(w \,|\, S) P(S)}{P(w).}$$

Alle Größen auf der rechten Seite sind, z.B. per ML, aus D schätzbar; $P(S \,|\, \text{click})$ könnte beispielsweise 0.95 sein, wenn 95% aller Vorkommen von „click“ im Spam-Korpus sind.

Um Entartungen zu vermeiden, legen wir außerdem fest, dass die größten und kleinsten Wahrscheinlichkeiten 0.99 und 0.01 sind und inferieren nur Wahrscheinlichkeiten für Wörter, die mehr als etwa 5 Mal im Gesamtkorpus vorkommen. Hier sehen wir wieder mal einen nichterwartungstreuen Schätzer. Tatsächlich würden wir hier ja vielleicht lieber mit Good-Turing anrücken – aber unser Klassifikationsproblem verträgt diese heuristische Vorschrift in der Tat besser als einen ordentlich begründeten Schätzer.

Wir wollen jetzt $P(S \,|\, \omega)$ aus mehreren Wörtern schätzen. Wir haben weiter die Naive Bayesian Assumption, müssen also nur $\prod_{w \in \omega} P(w \,|\, S)$ multiplizieren.

Es fehlen noch der Prior und die Normalisierung. Als Prior verwenden wir $P(H) = P(S) = 0.5$. Das mag willkürlich wirken, aber die Annahme von Gleichverteilung gilt gemeinhin als harmlos. Könnten wir uns auf unseren Korpus in dem Sinn verlassen, dass er tatsächlich aus dem „typischen“ Mailaufkommen gezogen ist, also so viel Spam und Ham enthält wie eben an Mail reinkommt, könnten wir die Frequenzen im Korpus verwenden. Wirklich schlau wäre das aber nicht, weil dies viele in Wirklichkeit wichtige Parameter verdecken würde. Wenigstens in Europa kommt z.B. der größte Teil des Spam über Nacht an, man bräuchte also zeitabhängige Prioren. Allerdings stellt sich heraus, dass der Prior im Groben unwichtig ist und nur bei verschwindend wenigen Mails seine (vernünftige) Wahl einen Einfluss auf die Entscheidung hätte.

Die Normalisierung ist auch hier im Prinzip nicht nötig, aber es ist toller, wenn man eine Mail mit einer Wahrscheinlichkeit von 0.99 oder 0.71 als Spam klassifizieren kann.

Die Normalisierung könnten wir durch die volle Bayes-Formel machen (Division durch $P(w)$), einfacher aber ist es, $P(S \,|\, \omega)$ und $P(H \,|\, \omega)$ zu berechnen und durch die Summe zu teilen – wir wissen, dass $P(c \,|\, \omega)$ normalisiert sein muss.

[4] `http://www.paulgraham.com/spam.html`

Zusammen ergibt sich:

$$P(S \mid \omega) = \frac{\prod P(w \mid S) P(S)}{\prod P(w \mid S) + \prod \big(1 - P(w \mid S)\big)}$$

Das Ganze funktioniert noch erheblich besser, wenn man nur die N „signifikantesten“ Wörter verwendet, d.h. die mit den größten Abweichungen vom „neutralen“ 0.5. Graham empfiehlt $N = 15$. Dies beugt auch Numerikproblemen vor.

Aufgaben

(26.1)* Holt euch das Skript `naiveBayesian.py` von der Webseite zur Vorlesung. Macht euch klar, dass sowohl `NaiveClassifier` als auch `GrahamClassifier` das tun, was wir in der Vorlesung dazu gesagt haben. Dazu müsst ihr kurz sehen, was in den jeweiligen `learn`- und `classify`-Methoden steht. `rawData` wird von `TrainableClassifier.train` ausgefüllt; in dessen Docstring steht, was der Inhalt zu verstehen ist.

(26.2) Besorgt euch eine Trainingsmenge von Ham und Spam. Die testClassifier-Funktion unten erwartet Spam- und Ham-Korpora in *Dateien* spam.canned und ham.canned (das sollten schon einige 100k sein) und Validierungsdaten als Einzeldateien in Unterverzeichnissen validation.ham und validation.spam. Die vorgefertigten Daten bestehen aus Ausschnitten aus Mailinglistenarchiven von GnuPG und einer Mukoviszidose-Selbsthilfegruppe (ham) und an mich adressiertem Spam. Das Spielen macht mehr Spaß, wenn ihr eure eigenen Daten verwendet.

Ruft testClassifier sowohl mit NaiveClassifier als auch mit GrahamClassifier auf. In der Ausgabe könnt ihr sehen, wie gut das funktioniert hat.

(26.3) Sucht euch ein paar eigene Daten zum Klassifizieren. Das Ergebnis sollte schlechter bis katastrophal sein. Warum? Um dem Ganzen auf die Spur zu kommen, lasst euch beim `GrahamClassifier` die ersten paar (vielleicht 20) Elemente von `sortedEvidence` in `_computePosProb` ausgeben. Beim `NaiveClassifier` könnt ihr euch in der Schleife in `classify` `word` und die zugehörige `wDist` ausgeben lassen (bevorzugt, wenn die Dokumente, die ihr klassifizieren lassen wollt, nicht so lang sind).

(26.4) Fügt Trainingsdaten, die zu den Dokumenten, die ihr klassifiziert haben möchtet, an die gegebenen Trainingsdaten an. Wie viele Daten braucht ihr, um die Klassifizierer auf eure speziellen Varianten von Ham und Spam zu trainieren?

(26.5) Überlegt euch, ob ihr in eurem Datenleben andere, ähnlich zu behandelnde Klassifikationsprobleme habt. Vorschlag zu einer Implementierung: Umgebungen eines Wortes (z.B. Birne) als Dokumente nehmen und sehen, ob ihr die verschiedenen Lesungen mit den Klassifizierern auseinanderhalten könnt (dazu müsst ihr natürlich zunächst Trainingsdaten sammeln).

Datei(en) im PDF-Anhang: naiveBayesian.py naiveBayesianData.tar.gz

27. Markow-Ketten

stochastischer Prozess
Markow-Kette
stationär
homogen
Übergangswahrscheinlichkeit
Übergangsmatrix

Leitfragen zu Abschnitt 27

- Wie können wir Texte handhabbar modellieren?
- Was ist die Markov-Bedingung und warum macht sie unser Leben erheblich leichter?
- Wie kommt man auf Übergangsmatritzen?

Sei $T \subseteq \mathbb{N}$ eine Indexmenge (typisch: die Zeit, Position des Worts im Text, Position des Phons), $(X_t)_{t \in T}$ eine Familie von Zufallsvariablen mit Werten in $\mathcal{X}$. Dann heißt X_t *stochastischer Prozess*. Ein klassischer stochastischer Prozess ist die Brown'sche Bewegung (auch: Random Walk). In der einfachsten Version ist X dabei die Position des Teilchens im (der Einfachheit halber eindimensionalen) Raum, t die Zeit. Bei jedem Schritt geht das Teilchen jeweils mit Wahrscheinlichkeit 0.5 nach oben oder unten.

Für uns allerdings viel wichtiger ist die Interpretation eines Textes als stochastischer Prozess – wir hatten das bisher als „Folge von Zufallsvariablen“ gefasst, was eigentlich genau der stochastische Prozess ist. Da es uns jetzt aber verstärkt um Abhängigkeiten der X_t untereinander gehen soll (während wir bisher immer die i.i.d.-Annahme hatten), wollen wir die Begriffe etwas genauer definieren.

Eine *Markow-Kette* ist ein stochastischer Prozess, bei dem gilt:

$$P(X_{n+1} = x_{n+1} \mid X_0 = x_0, \ldots, X_n = x_n) = P(X_{n+1} = x_{n+1} \mid X_n = x_n),$$

wenn die Bedingung überhaupt erfüllbar ist.

Das bedeutet zunächst nichts weiter, als dass X_{n+1} nur von X_n bestimmt wird, das Verhalten des Systems also nur vom vorhergehenden Zustand abhängig ist. Die Brown'sche Bewegung ist so ein Fall: Wenn wir wissen, dass $X_4 = 2$, so ist die Verteilung von X_5

$$P(X_5 = x) = \begin{cases} 0.5 & x = 1 \\ 0.5 & x = 3 \\ 0 & \text{sonst,} \end{cases}$$

ganz egal, was $X_0, \ldots, X_3$ sind.

Markow wird in der englischen Literatur üblicherweise Markov geschrieben (historische Anmerkung: Die erste Markow-Kette, die in der Literatur verzeichnet ist, beschrieb die Abfolge der Zustände „Konsonant“ und „Vokal“ in russischer Literatur und sollte eigentlich dazu dienen, die Notwendigkeit der Unabhängigkeit für das Gesetz der großen Zahlen nachzuweisen).

Eine Markow-Kette heißt *stationär* oder *homogen*, wenn für alle $i, j \in \mathcal{X}$ die *Übergangswahrscheinlichkeit*

$$p_{ij} := P(X_{t+1} = j \mid X_t = i)$$

unabhänging von t ist.

Dies bedeutet, dass nicht nur die Geschichte keine Rolle für die Übergangswahrscheinlichkeiten spielt, sondern dass noch dazu auch die Zeit unwichtig ist. Die Brown'sche Bewegung ist homogen.

Homogene Markow-Ketten lassen sich offenbar allein durch die Zahlen p_{ij} charakterisieren, also einfach alle Übergangswahrscheinlichkeiten (bei nicht-homogenen Markow-Ketten wären die p_{ij} Funktionen und keine Zahlen). Wir haben damit schlimmstenfalls abzählbar unendlich viele Zahlen, die durch zwei Indizes organisiert sind, also eine Matrix. Diese spezielle Matrix heißt *Übergangsmatrix*.

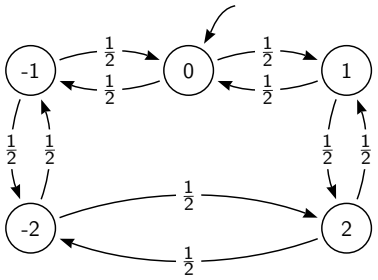

Fig. 16

Die Übergangsmatrix der Brown'schen Bewegung ist

$$(p_{ij})_{i,j\in\mathbb{Z}} = \begin{pmatrix} \ddots & \ddots & \ddots & \vdots & \\ \ddots & 0 & \frac{1}{2} & 0 & \cdots \\ \ddots & \frac{1}{2} & 0 & \frac{1}{2} & \ddots \\ \cdots & 0 & \frac{1}{2} & 0 & \ddots \\ & \vdots & \ddots & \ddots & \ddots \end{pmatrix}$$

Eine Übergangsmatrix ist ein Beispiel für eine *stochastische Matrix*. Da in solch einer Matrix zeilenweise Wahrscheinlichkeiten stehen, muss für ihre Koeffizienten p_{ij} gelten:

- $p_{ij} \geq 0$
- $\sum_j p_{ij} = 1$.

Stochastische Prozesse in der Computerlinguistik sind in aller Regel stationäre Markow-Ketten, die Werte der Zufallszahlen als „Zustände“ interpretierbar. Unsere Markow-Ketten sind meist endlich lang, während man sich im „Mainstream“ gerne mit unendlichen Ketten auseinandersetzt und z.B. nach *Rekurrenz* und *Transienz* von Zuständen oder *Absorptionswahrscheinlichkeit*en fragt.

Dabei heißt ein Zustand x rekurrent, wenn der Zustand x mit Wahrscheinlichkeit 1 beliebig oft angenommen wird, transient sonst. Eine Absorptionswahrscheinlichkeit entsteht, indem man eine Teilmenge J von $\mathcal{X}$ als „absorbierend“ definiert, der Prozess also abgebrochen wird, wenn das erste X_n einen Zustand aus J annimmt. Bei der Brown'schen Bewegung könnte das beispielsweise eine Wand bei j sein, so dass der Prozess endet, sobald $X_n < j$ wird. Die Frage ist nun, mit welcher Wahrscheinlichkeit der Prozess bei bestimmten Anfangsbedingungen endet.

(cf. Fig. 16)

Graph einer Brown'schen Bewegung mit fünf Zuständen und periodischen Randbedingungen

Eine direkte Folge der Markow-Eigenschaft ist die *Chapman-Kolmogorow-Gleichung*

$$P(X_n = t \,|\, X_k = s) = \sum_{x\in\mathcal{X}} P(X_m = x \,|\, X_k = s) P(X_n = t \,|\, X_m = x)$$

für alle $t, s \in \mathcal{X}$ und $k < m < n$.

Das bedeutet, dass die Wahrscheinlichkeit, „indirekt“ vom Zustand s („Source“) zum Zustand t („Target“) zu kommen, durch Summieren über alle möglichen Zwischenwege berechenbar ist.

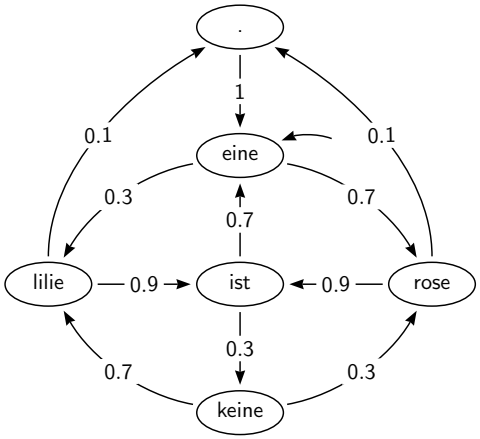

Fig. 17

Zum Beweis nehmen wir zunächst speziell $k = n + 2$ und $m = n + 1$ an. Es ist

$$\begin{aligned} P(X_k = s, X_n = t) &= \sum_x P(X_k = s, X_m = x, X_n = t) \\ &= \sum_x P(X_k = s, X_m = x) P(X_n = t \,|\, X_k = s, X_m = x) \end{aligned}$$

Wegen der Markow-Eigenschaft ist $P(X_n = t \,|\, X_k = s, X_m = x) = P(X_n = t \,|\, X_m = x)$ (hier brauchen wir unsere spezielle Annahme von oben). Eine Division durch $P(X_k = s)$ liefert die Behauptung für den Spezialfall. Die Verallgemeinerung folgt mit vollständiger Induktion (was wir hier nicht machen, aber eine nette Übung ist).

Die Gesamtwahrscheinlichkeit für eine bestimmte Zustandsfolge $P(x_1, \ldots, x_n)$ ist einfach

$$\begin{aligned} &P(x_1) P(x_2 \,|\, x_1) P(x_3 \,|\, x_2) \cdots P(x_n \,|\, x_{n-1}) \\ &= P(x_1) \prod_{i=1}^{n-1} p_{x_i, x_{i-1}}. \end{aligned}$$

Darin haben wir $P(X_i = x_i)$ als $P(x_i)$ abgekürzt.

Ein weiteres Beispiel für eine Markow-Kette ist folgenes Modell moderner Poesie:

(cf. Fig. 17)

Der Startzustand ist dabei – wie üblich – durch einen Pfeil ohne Quellzustand markiert.

In ein Programm verwandelt, das jeweils die Namen der Zustände, durch die das Modell läuft, ausgibt, liefert es Sätze wie: „eine rose ist eine lilie ist eine rose ist eine lilie.“ oder „eine rose.“ oder „eine rose ist eine rose ist eine rose ist keine lilie ist keine rose ist eine lilie ist eine rose ist keine rose ist keine rose ist eine lilie.“ Dabei wurde der „.“-Zustand als absorbierend definiert.

Die Wahrscheinlichkeit für den Satz „eine rose ist eine rose.“ ist nach unseren Überlegungen oben

$$1 \cdot 0.7 \cdot 0.9 \cdot 0.7 \cdot 0.7 \cdot 0.1 = 0.031.$$

Exkurs: Page Rank

Der Page Rank ist eines der zentralen Ordnungskriterien für Suchergebnisse bei Google und interpretiert Links als Empfehlungen, rankt also Seiten, auf die viele Links zeigen, hoch, wobei Links von Seiten mit hohem Page Rank wiederum als „wertvoller“ gerechnet werden. Brin und Page, die sich das ausgedacht haben, kamen eher aus der Algebra und berechneten die Page Ranks aus dem Eigenvektor zum Eigenwert 1 der Matrix, die die Linkstruktur des Webs repräsentiert (eine wahrhaft große Matrix mit etlichen Milliarden Zeilen und Spalten, wobei allerdings glücklicherweise die meisten Einträge Null sind, weil ja eine bestimmte Seite nur auf relativ wenig andere Seiten verweist).

Eine alternative Interpretation des Page Rank ist aber die als die asympotitsche Aufenthaltswahrscheinlichkeit eines „Random Surfers“, der auf irgendeiner Seite startet und dann immer einem zufälligen Link folgt. Das ist letztlich ein Markow-Prozess, wobei natürlich ein paar Tricks gemacht werden müssen, damit die (hier nicht entwickelte) Mathematik greifen kann (z.B. gibt es im Web Sackgassen, also Seiten ohne Links, die die Ergodizität dieses Prozesses kaputt machen).

Aufgaben

(27.1)* Holt euch von der Webseite zur Vorlesung das Skript `markovmodel.py`. Seht euch darin die Klasse `DistFunc` an – mit ihr werden wir nachher den jeweils nächsten Schritt ziehen. Sie wird mit einer Verteilung konstruiert (hier einfach eine Liste von Zahlen). Ruft man `draw()` oft genug auf, sollten die Ergebnisse auf Dauer entsprechend der übergebenen Verteilung verteilt sein. Könnt ihr sehen, wie das hier funktioniert? Probiert aus, ob das stimmt. Benutzt dafür Code, der etwa so aussieht:

```
for val in DistFunc([1,1,1]).sampleRandomly(1000):
  print val
```

Schickt die Ausgabe in einer Pipe durch `sort | uniq -c` – die Zahlen von 0 bis 2 sollten etwa gleich häufig vorkommen. Probiert das auch mit anderen Verteilungen.

(27.2)* Verschafft euch einen groben Überblick über `MarkovModel`. Der Konstruktor des `MarkovModels` bekommt ein Dictionary-Literal in einem String übergeben und baut daraus eine Listen von Listen, f, bei der in jeder Unterliste die Wahrscheinlichkeit für einen Übergang vom Zustand index-der-unterliste in den Zustand index-in-der-liste steht. Das brauchen wir z.B. in `computePathProb`, um die Wahrscheinlichkeit auszurechnen, dass eine bestimmte Zustandsfolge vom gegebenen Markov-Modell erzeugt wird. Daraus baut die Klasse anschließend eine Liste von `DistFuncs`, mit der wir z.B. in `driveOn` den nächsten Zustand ziehen können.

(27.3)* Seht euch an, wie das Markov-Modell einen (periodischen) Random Walk durchführt. Benutzt dafür das Modell

```
{
0: {1:0.5,-1:0.5},
1: {0:0.5,2:0.5},
2: {1:0.5,-2:0.5},
-1: {-2:0.5,0:0.5},
-2: {-1:0.5,2:0.5},
}
```

– ihr könnt das in eine Datei schreiben und ihren Inhalt dann beim Konstruieren an das `MarkovModel` übergeben. Seht euch zunächst ein paar kurze Zustandsfolgen an (vielleicht zehn Schritte lang), um ein Gefühl zu bekommen, was da so passiert. Erzeugt dann eine lange Zustandsfolge (vielleicht 1000 Schritte lang) und lasst euch die Verteilung der Zustände innerhalb dieser Folge ausgeben – am einfachsten geht das wieder mit den üblichen Unix-Tools durch

```
python markovmodel.py randomWalk.txt | sort | uniq -c
```

Verändert jetzt z.B. die 2 in der Zeile für den Zustand 1 in z.B. eine eins. Wie verändert sich die Verteilung der Zustände?

(27.4) Lasst jetzt 100 Mal hintereinander ein zufällig aufgesetztes Markovmodell für zehn Schritte laufen und lasst euch die summierte Verteilung der Zustandsfolgen ausgeben. Das geht z.B. mit folgendem Code und der Kommandozeile oben

```
  for i in xrange(100):
    m.setUpUniform()
    print "\n".join(map(str, m.getAPath(10)))
```


Auch hier sollte wieder eine Gleichverteilung herauskommen. Dies ist ein Spezialfall eines wichtigen Begriffs, nämlich der Ergodizität. Im Groben sind Systeme ergodisch, wenn das Mittel einer Eigenschaft über ganz viele Kopien (Scharmittel) dem Mittel dieser Eigenschaft über einen ganz langen Lauf eines Systems entspricht (Zeitmittel). Kurz: Zeitmittel=Scharmittel.

Die Ergodizität eines Markov-Modells ist sehr kritisch für seine einfache Trainierbarkeit. Im Allgemeinen sind Markov-Modelle, in denen alle Zustände in alle übergehen können, ergodisch und damit harmlos.

(27.5) Um ein einfaches Beispiel für nichtergodische Systeme zu haben, nehmt folgendes Markovmodell:

```
{
0: {1:0.5,-1:0.5},
1: {0:0.5,2:0.5},
2: {1:0.5,-2:0.5},
-1: {-2:0.5,0:0.5},
-2: {-2:1},
}
```

Wiederholt den Vergleich von Zeitmittel (eine lange Kette) zu Scharmittel (viele kurze Ketten) aus den letzten beiden Aufgaben. Woher kommt das nichtergodische Verhalten?

(27.6) Die `MarkovModel`-Klasse kann auch Fälle, in denen Zustände wirklich „absorbierend“ sind in dem Sinn, dass eine Exception ausgelöst wird, wenn das Modell in den betreffenden Zustand kommt. Nehmt folgende Übergangsverteilungen:

```
{
0: {0:0.5, 1:0.5,-1:0.5,-2:0.5,2:0.5},
1: {0:0.5, 1:0.5,-1:0.5,-2:0.5,2:0.5},
2: {0:0.5, 1:0.5,-1:0.5,-2:0.5,2:0.5},
-1: {0:0.5, 1:0.5,-1:0.5,-2:0.5,2:0.5},
-2: {},
}
```

– damit wir nachher leichter rechnen können, verlassen wir jetzt den Random Walk und lassen alles in alles übergehen (für die Normierung sorgt nachher MarkovModel selbst). Wir interessieren und jetzt für die Lauflänge eines Markow-Modells, bis es in den absorbierenden Zustand kommt. Das ist häufig eine interessante Frage (allerdings meines Wissens selten in der Sprachverarbeitung, ihre Untersuchung ist aber für das Verständnis von Markowprozessen trotzdem hilfreich).

Diese Lauflänge L ist eine Zufallsvariable (über welchem Ω?). $E(L)$ kann ganz gewöhnlich als $\sum_{l=1}^{\infty} lP(L=l)$ ausgerechnet werden. $P(L=l)$ lässt sich am besten als Wahrscheinlichkeit auffassen, dass der Pfad bis $l-1$ nicht absorbiert wurde, das aber bei l passiert.

Woher bekommen wir $P(L=l)$? Wegen der Gleichverteilung, die wir „erzwungen“ haben, ist das hier nicht schwer. Wir haben $l-1$ Schritte eine Wahrscheinlichkeit von 0.2 gehabt, absorbiert zu werden und haben das vermieden (mithin mit Wahrscheinlichkeit 0.8) und sind beim l-ten Mal in die Falle getappt (wofür die Wahrscheinlichkeit 0.2 war).

Wir haben damit $P(L=l) = 0.8^{l-1} \times 0.2$.

Im allgemeinen Fall ist das viel komplizierter – ihr könnt ja eine Weile grübeln, wie sowas mit dem Markovmodell aus der letzten Aufgabe aussehen würde. Aber grübelt nicht so lang.

Jetzt können wir die Summe oben als

$$0.2 \sum_{l=1}^{\infty} l 0.8^{l-1}$$

ausrechnen. Eine Formelsammlung verrät, dass

$$\sum_{l=0}^{\infty} lp^l = p/(p-1)^2$$

gilt. Wenn wir ein $1/0.8$ ausklammern, können wir die Formel anwenden und erhalten

$$E(L) = 0.2/0.8 \times 0.8/(0.8-1)^2 = 5.$$

Prüft dieses Ergebnis empirisch nach: Mit dem obigen Automaten wirft `driveOn` eine Exception, wenn es im absorbierenden Zustand landet. `getAPath` bricht dann die Generierung des Pfads ab, und die Länge des Pfades gibt gerade L. `computeLExpect` nutzt das aus. Interessant bei der Sache ist auch, das Gesetz der großen Zahlen in Aktion zu sehen, wenn ihr mit „großen Zahlen“ in die Zahl der Experiment geht.

Wenn euch das näher interessiert: Man kann hier allerlei Statistiken darüber anstellen, welche Pfade wie häufig vorkommen. Das ist mit unserem vereinfachten Modell langweilig, mit den nicht vollständig durchverbundenen Modellen oben aber recht spannend.

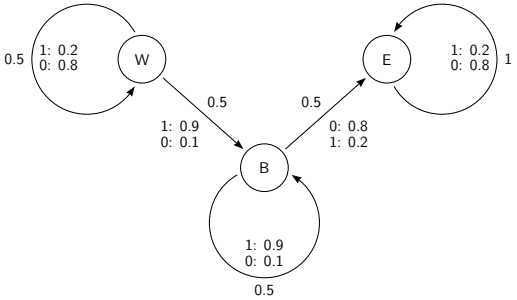

Fig. 18

Datei(en) im PDF-Anhang: markovmodel.py

28. Hidden Markov Models

Leitfragen zu Abschnitt 28

- Was kommt bei HMMs zu den normalen Markov-Modellen dazu? Warum macht man diese „Komplikation"?
- Wie ist der Zusammenhang zwischen der Ausgabe eines HMM und der Zustandsfolge, die er durchlaufen hat?

*Hidden Markov Model*s (HMMs) sind Markov-Ketten, in denen bei jedem Übergang etwas ausgegeben wird – was, wird wiederum von einer Wahrscheinlichkeitsverteilung bestimmt.

Allgemein: Ein HMM ist ein Tupel $\mu = (\Phi, \Sigma, S, \delta, \lambda)$. Dabei ist

- Φ eine Menge von Zuständen,
- Σ eine Menge von Ausgabesymbolen,
- $S \in \Phi$ ein Startzustand,
- $\delta\colon \Phi \to \Phi \times [0,1]$ eine Menge von Verteilungen, die die Übergangswahrscheinlichkeiten für jeden Zustand beschreibt,
- $\lambda\colon \Phi \times \Phi \to \Sigma \times [0,1]$ eine Menge von Verteilungen, die zu jedem Übergang die Wahrscheinlichkeiten für die Ausgabe eines Zeichens bestimmt.

Die Funktionen λ und δ haben wir hier etwas komisch notiert. Das kommt daher, dass es Abbildungen von „komischen" Mengen (etwa der Menge aller Paare von Zuständen) in andere „komische" Mengen (etwa die Menge aller Verteilungen über den Ausgabesymbolen) sind. In der Realität werden wir $\delta(i,j)$ für die Übergangswahrscheinlichkeit von i nach j schreiben und $\lambda(i,j,k)$ für die Wahrscheinlichkeit, dass bei diesem Übergang das Symbol k ausgegeben wird.

Es gibt viele Variationen zu diesem Thema, beginnend bei der Notation. Eine wichtige Variation ist, statt des Startsymbols S eine Verteilung Π auf Φ anzugeben, die die Wahrscheinlichkeiten angibt, dass das HMM in den jeweiligen Zuständen startet.

Manning und Schütze schreiben für unsere Übergangswahrscheinlichkeiten $\delta(X_1, X_2)$ ein a_{X_1,X_2} und für unsere Emissionswahrscheinlichkeiten $\lambda(X_1, X_2, o_t)$ ein b_{X_1,X_2,o_t}.

Außerdem könnte man statt *arc emission* (die Zeichen werden bei Übergängen geschrieben) auch *state emission* verwenden, bei der im Zustand geschrieben wird, nicht bei Übergängen. Dann wäre $\lambda\colon \Phi \to \Sigma \times [0,1]$.

Diese und weitere Variationen ändern nur Details, die resultierenden HMMs haben alle die gleichen Fähigkeiten und Beschränkungen.

Hidden Markov Model

arc emission

state emission

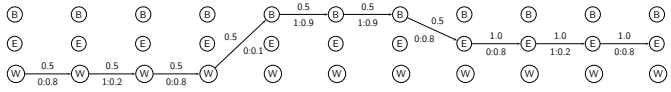

Fig. 19

trainieren

(cf. Fig. 18)

Darstellung eines HMM: Zustände als Kreise, Übergänge mit nichtverschwindender Wahrscheinlichkeit als Pfeile, die mit Übergangswahrscheinlichkeiten und der Verteilung der Ausgabesymbole gekennzeichnet sind.

Wichtig: HMMs können identische Sequenzen aus verschiedenen Zustandsfolgen generieren.

Das folgende HMM modelliert z.B. das Ergebnis der Übertragung eines senkrechten schwarzen Streifens durch einen verrauschten Kanal, der hin und wieder weiße (0) durch schwarze (1) Pixel ersetzt. Am Anfang werden (im Zustand W) mit hoher Wahrscheinlichkeit Nullen emittiert, dann kommt der Streifen (im Zustand B), wo mit hoher Wahrscheinlichkeit Einsen emittiert werden, und schließlich kommt weiß rechts vom Streifen im Zustand E.

(cf. Fig. 19)

Die Abbildung oben zeigt einen möglichen Durchlauf durch das HMM. Die Ausgabe ist dabei 0100110010 (d.h. auch in den „weißen" Teilen sind schwarze Pixel). Die totale Wahrscheinlichkeit dieses Durchlaufs ist dabei das Produkt *aller* auftretenden Wahrscheinlichkeiten, also $(0.5 \cdot 0.8) \cdot (0.5 \cdot 0.2) \cdots (1 \cdot 0.8) \approx 8.3 \times 10^{-6}$. Es gibt aber viele weitere Wege durch das Modell, die dieselbe Ausgabe erzeugen würden, die totale Wahrscheinlichkeit für diese spezielle Ausgabe ist also höher.

29. Standardmethoden von HMMs I

Leitfragen zu Abschnitt 29

- Was hat man von der Beschreibung eines Systems durch ein HMM?
- Wie kriegt man die Gesamtwahrscheinlichkeit für eine bestimmte Ausgabe des HMM heraus? Warum ist das überhaupt schwierig?

Praktisch alle Anwendungen von HMMs beruhen auf der Beantwortung von folgenden Fragen:

1. Wie wahrscheinlich ist eine Beobachtung O in einem gegebenen Modell μ?
2. Was ist die wahrscheinlichste Erklärung für die Beobachtung O unter einem Modell μ?
3. Sei eine Beobachtung O und ein Menge $M = \{\mu(\vartheta) \mid \vartheta \in \Theta\}$ möglicher Modelle gegeben. Welches Modell $\mu(\vartheta_0)$ erklärt O am besten?

Wir sprechen hier immer von Beobachtung, wo wir vorher von Ausgabe gesprochen haben; Hintergrund ist, dass HMMs fast nie zur Generierung von Sequenzen verwendet werden, sondern fast immer zur a-posteriori-Erklärung von beobachteten Sequenzen. Das ist auch der Hintergrund des Wortes „hidden": Man sieht nur das Ergebnis des HMM, die Folge der Zustände ist zunächst versteckt.

Die dritte Frage stellt sich im Allgemeinen als Lernproblem. Wir haben viele Beobachtungen und wollen zunächst ein HMM *trainieren*, die Parameter (ϑ wird meist vektorwertig sein) also so einstellen, dass sie den Beobachtungen maximal gut angepasst sind. Mit dem trainierten HMM können dann Fragen vom Typ 2 beantwortet werden.

Das erste Problem

dynamic programming

Trellis

Was ist die Wahrscheinlichkeit für eine Beobachtung O?

Eigentlich einfach: Wir rechnen für alle möglichen Zustandsfolgen $X = (X_1, \ldots, X_{T+1})$ die individuellen Wahrscheinlichkeiten $P(O \mid X, \mu)$ aus und addieren sie.

Diese Wahrscheinlichkeiten sind tatsächlich einfach auszurechnen.

$$\begin{aligned} P(O \mid X, \mu) &= P(X_1) \prod_{t=1}^{T} P(o_t \mid X_t, X_{t+1}, \mu) \\ &= P(X_1) \prod_{t=1}^{T} \lambda(X_t, X_{t+1}, o_t) \delta(X_t, X_{t+1}), \end{aligned}$$

wobei o_t das zwischen t und $t+1$ emittierte Zeichen ist. Die Bedingung auf μ ist hier natürlich eigentlich überflüssig, da wir im Augenblick ohnehin immer nur ein Modell haben.

Es ist nicht schwer zu zeigen, dass die einzelnen $P(O \mid X, \mu)$ unabhängig sind, also eine schlichte Addition dieser Größen die gesuchte Gesamtwahrscheinlichkeit liefert.

Problem: Laufzeit ist ewig, man braucht $(2T+1)N^{T+1}$ Multiplikationen für N Zustände.

Dynamic Programming

Abhilfe ist das *dynamic programming*. Idee dabei ist, einmal berechnete Werte in Tabellen zu halten. Der Begriff kommt aus einer sehr allgemeinen Form des Programmbegriffs, es lohnt nicht, allzu viele Gedanken auf ihn zu verschwenden.

Dazu: Tabelle (*Trellis*)

$$\alpha_i(t) = P(o_1 \cdots o_{t-1}, X_t = i \mid \mu)$$

Dazu zählen wir die Zustände des HMM einfach irgendwie durch. Der Index i von $\alpha_i(t)$ „durchläuft“ dann alle möglichen Zustände, der Parameter t die Zustandsfolge. Im fertigen Trellis stehen die Wahrscheinlichkeiten dafür, nach der Beobachtung von $o_1 \cdots o_{t-1}$ zum Zeitpunkt t im Zustand i zu landen.

Aufbauen der Tabelle: $\alpha_i(1) = \delta_{ik}$ mit k Index des Startzustands; von dort „nach vorne“ (d.h. zu späteren Zuständen hin) durcharbeiten:

$$\alpha_j(t+1) = \sum_{i=0}^{N} \alpha_i(t) \delta(i,j) \lambda(i,j,o_t).$$

δ_{ij} ist das Kronecker-Symbol

$$\delta_{ij} = \begin{cases} 1 & \text{i=j} \\ 0 & \text{sonst,} \end{cases}$$

so dass anfangs die Wahrscheinlichkeit für den Startzustand Eins ist, die für alle anderen Null. Dieses δ hat *nichts* mit unserer Übergangsfunktion δ zu tun!

Danach werden einfach die Wahrscheinlichkeiten aller Übergänge, die auf den augenblicklichen Zustand führen können und dabei o_t emittieren, aufsummiert.

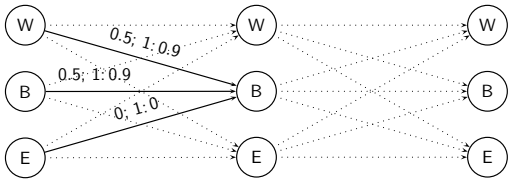

Fig. 20

Ein Bild mag helfen; hier die Berechnung von $\alpha_2(t+1)$:

(cf. Fig. 20)

Sei $\alpha_1(t) = 0.15$, $\alpha_2(t) = 0.05$, $\alpha_3(t) = 0.02$. Wenn von t auf $t+1$ eine 1 emittiert wurde, ist

$$\alpha_2(t+1) = 0.15 \cdot 0.5 \cdot 0.9 + 0.05 \cdot 0.5 \cdot 0.9 + 0.02 \cdot 0 \cdot 0 = 0.09.$$

Die totale Wahrscheinlichkeit für die Beobachtung O ist dann

$$P(O \mid \mu) = \sum_{i=1}^{N} \alpha_i(T+1).$$

Wir berechnen hier also einfach die Summe der Wahrscheinlichkeiten, dass das Modell bei der gegebenen Ausgabe im Zustand i endet. Bei vielen Markow-Modellen werden ziemlich viele der $\alpha_i(T+1)$ null sein, z.B. für alle Zustände, die gar keine ankommenden Übergänge haben, die das letzte Zeichen emittieren.

30. Standardmethoden von HMMs II

Leitfragen zu Abschnitt 30

- Woher kommen die magischen Formeln aus der forward- und backward-Procedure?
- Was ist die „beste“ Zustandsfolge zur Erklärung einer Ausgabe eines HMM? Warum will man die überhaupt haben?

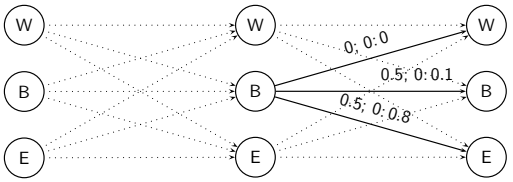

Fig. 21

Backward procedure

Wir können auch umgekehrt bedingen:

$$\beta_t(t) = P(o_t \cdots o_T \,|\, X_t = i, \mu).$$

Die Frage ist jetzt also: Wie groß ist die Wahrscheinlichkeit, dass wir $o_t \cdots o_T$ beobachten werden, wenn wir zum Zeitpunkt t im Zustand i sind? Im Vergleich zur „forward procedure“ haben wir jetzt Wahrscheinlichkeiten für etwas, das wir noch beobachten *werden, unter der Bedingung*, dass wir jetzt in einem bestimmten Zustand sind, während wir bei den α_i die Wahrscheinlichkeit dafür berechnet haben, dass wir in einem bestimmten Zustand *sind*, wenn wir etwas beobachtet *haben*.

Diese Umkehrung hat für sich keinen wirklichen Sinn, wird uns aber beim Trainieren von HMMs sehr helfen.

Am Ende der Beobachtung ist es sicher, dass wir nichts mehr beobachten, also

$$\beta_i(T+1) = 1 \quad \text{für alle } i.$$

Wir arbeiten uns von dort aus nach vorne durch:

$$\beta_j(t) = \sum_{i=0}^{N} \beta_i(t+1)\delta(j,i)\lambda(j,i,o_t).$$

Die totale Wahrscheinlichkeit ist dann $\beta_k(1)$, wenn k der Index des Startzustands ist.

Wieder mag ein Bild helfen:

(cf. Fig. 21)

Nehmen wir an, dass $t+2$ der letzte Schritt ist. Dann sind alle $\beta_i(t+2) = 1$. Die Wahrscheinlichkeit, dass wir 0 beobachten werden, wenn wir zum Zeitpunkt $t+1$ im Zustand 2 sind, ist

$$\beta_2(t+1) = 1 \cdot 0 \cdot 0 + 1 \cdot 0.5 \cdot 0.1 + 1 \cdot 0.5 \cdot 0.8 = 0.45.$$

Die beste Zustandsfolge

Die zweite Frage: Was ist die beste Zustandsfolge, um eine gegebene Beobachtung zu erklären?

Naiv: Einfach zu jedem Zeitpunkt t das x_t suchen, für das $P(X_t = x_t \,|\, O, \mu)$ maximal wird.

Im Einzelfall kann das sinnvoll sein. Im Allgemeinen wollen wir aber nicht $\prod_{t=0}^{T+1} P(X_t = x_t \,|\, O, \mu)$ maximieren, sondern $P(X_1 = x_1, \ldots X_{T+1} = x_{T+1} \,|\, O, \mu)$.

Diese beiden Größen sind verschieden, weil die X_t nach Definition der Markowkette nicht unabhängig voneinander sind.

Für drei Zustände könnte das so aussehen:

$$P(x_1, x_2, x_3) = P(x_3 \,|\, x_1, x_2)P(x_1, x_2)$$
$$P(x_3 \,|\, x_2)P(x_2 \,|\, x_1)P(x_1).$$

Dabei habe ich erstens alle „konstanten“ Abhängigkeiten weggelassen (O und μ ändern sich während der Berechnung nicht) und zweitens wieder $X_i = x_i$ als x_i abgekürzt. Solche Geschichten werden in der Literatur gern gemacht, weil sonst alle Ölquellen Saudi-Arabiens nur noch für die Produktion von Druckerschwärze sprudeln müssten.

Das ist nur dann gleich $P(x_1)P(x_2)P(x_3)$, wenn $P(x_{i+1} \,|\, x_i) = P(x_{i+1})$ ist. Das wiederum ist nur dann der Fall, wenn X_i und X_{i+1} unabhängig sind, wir also gar kein Markov-Modell, sondern i.i.d. haben. Wer es nicht sieht:

$$P(x_{i+1} \,|\, x_i) = \frac{P(x_{i+1}, x_i)}{P(x_i)} = P(x_{i+1})$$

ist äquivalent zu $P(x_{i+1})P(x_i) = P(x_{i+1})P(x_i)$.

Wenn wir also die „beste“ Zustandsfolge in einem Markov-Modell finden möchten, müssen wir einen Ausdruck mit all den bedingten Wahrscheinlichkeiten ausrechnen. Die Standardmethode wurde von Andrew Viterbi entwickelt und ist unser nächstes Thema.

31. Der Viterbi-Algorithmus

Leitfragen zu Abschnitt 31

- Wie kriegt man die wahrscheinlichste Zustandsfolge mit erträglichem Rechenaufwand raus?
- Warum braucht man dazu zwei Trellise?

Der *Viterbi-Algorithmus* findet zu jeder Beobachtung O die wahrscheinlichste Zustandsfolge X, die ein HMM bei der Ausgabe von O genommen hat.

Wir definieren uns ein Trellis

$$\varphi_j(t) = \max_{X_1 \cdots X_{t-1}} P(X_1 \cdots X_{t-1}, o_1 \cdots o_{t-1}, X_t = j \,|\, \mu).$$

$\varphi_j(t)$ enthält also an jeder Stelle die Wahrscheinlichkeit des wahrscheinlichsten Weges, der unter Emission von $o_1 \cdots o_{t-1}$ zum Zustand j zur Zeit t führt. Wir brauchen aber noch zusätzlich die Information, welcher Weg das war. Diese Information wird uns das unten definierte zweite Trellis ψ liefern.

Die Initialisierung ist wie gewohnt: $\varphi_i(t) = \delta_{ij}$ mit dem Startzustand j.

Wir arbeiten uns nach vorne durch,

$$\varphi_j(t+1) = \max_{i=1,\ldots,N} \varphi_i(t)\, \delta(i,j)\, \lambda(i,j,o_t),$$

und merken uns, woher wir jeweils gekommen sind:

$$\psi_j(t+1) = \operatorname*{argmax}_{i=1,\ldots,N} \varphi_i(t)\, \delta(i,j)\, \lambda(i,j,o_t).$$

Der Operator argmax bestimmt das Argument des Maximalwerts, wobei das Argument das ist, worüber die Maximierungsoperation läuft. Im Fall oben ist das das i, so dass wir in ψ gerade speichern, woher unser φ kam. Wir werden das brauchen, um uns wieder zurückzuhangeln.

Wenn unsere beiden Trellises gefüllt sind, suchen wir von hinten startend die wahrscheinlichste Sequenz $\hat{X}$:

$$\begin{aligned}\hat{X}_{T+1} &= \operatorname*{argmax}_{i=1,\ldots,N} \varphi_i(T+1)\\ \hat{X}_t &= \psi_{\hat{X}_{t+1}}(t+1)\\ P(\hat{X}) &= \varphi(\hat{X}_{T+1}).\end{aligned}$$

Trellise im Viterbi-Algorithmus für die Eingabe 01011100 und unser Schwarzweiß-HMM:

o_j		φ			ψ	
	W	B	E	W	B	E
0	0.0	-	-	-	-	-
1	0.40	1.3	-	W	W	-
0	1.4	0.74	2.6	W	W	B
1	1.8	2.0	1.1	W	B	B
1	2.8	2.1	1.8	W	W	E
1	3.8	2.5	2.5	W	B	E
0	4.8	2.8	3.2	W	B	E
0	5.2	4.1	3.2	W	B	B
-	5.6	5.4	3.3	W	B	E

Die Wahrscheinlichkeiten sind als negative dekadische Logarithmen gegeben, 4.61 bedeutet etwa $10^{-4.61} \approx 2.45 \times 10^{-5}$. Für so kleine Zahlen ist das bequemer, und in der Tat lohnt es sich, den Viterbi-Algorithmus komplett in Logarithmen zu implementieren.

Ein - als Quellzustand bedeutet, dass es keinen Weg zum entsprechenden Zustand gibt. Da unsere HMMs beim Übergang emittieren, durchlaufen sie für n Eingabezeichen $n+1$ Zustände, weshalb beim letzten Eingabezeichen nur ein Strich steht.

Readout (von hinten): E, E, B, B, B, W, W, W mit Wahrscheinlichkeit $10^{-3.3}$. Der schwarze Strich geht wahrscheinlich von Spalte 4 bis Spalte 6.

Dabei interessiert uns jeweils der Zustand, in den das System auf dem wahrscheinlichsten Pfad bei „Emission“ des jeweiligen Zeichens gegangen ist. Ganz am Schluss war das ein E. Dessen Quelle war laut ψ wieder ein E (in der Zeile darüber), dessen Quelle wiederum ein B und so weiter. Für die erste Zeile schreiben wir gar keinen Zustand auf, da ja das System hier noch gar kein Zeichen gelesen hatte, um in den Zustand zu kommen.

Aufgaben

(31.1)* Besorgt euch das Programm `hmm.py` von der Vorlesungsseite. Das Programm benötigt das `markovmodel`-Modul von der Folie „Markov-Ketten“.

Seht euch darin die HMM-Klasse an. Sie macht zunächst für die Ausgabesymbole, was die MarkovModel-Klasse schon für die Zustände gemacht hat: Die Ausgabesymbole werden aufgezählt, so dass wir intern nur noch Zahlen zwischen 0 und $n-1$ verwenden (`_computeEmissionMapping`).

Das ist für ein nacktes Markov-Modell kein großer Vorteil, hilft aber, wenn wir nachher unsere „Standardfragen“ beantworten wollen. Ebenfalls analog zu MarkovModel berechnen wir sowohl eine Übergangsverteilung (`_computeEmissionMatrix`) als auch Verteilungsfunktionen für die Ausgaben der Übergänge, die wir wieder beim Generieren brauchen (`_computeOutputDfs`) und die nach Notwendigkeit von `driveOn` aufgerufen wird. `driveOn` wiederum greift auf das `driveOn` des `MarkovModels` zu und berechnet zusätzlich die Ausgaben.

(31.2)* Probiert die HiddenMarkovModel-Klasse aus. Folgende Beschreibung baut einen HMM ähnlich dem in der Vorlesung besprochenen für einen schwarzen Streifen:

```
{
"whitebefore": {"blackstrip": 0.1, "whitebefore": 0.9},
"blackstrip": {"whiteafter": 0.1, "blackstrip": 0.9},
"whiteafter": {"whiteafter": 1},
}
---
{"whitebefore": 1}
---
+whitebefore-whitebefore+ {'0': 0.8, '1': 0.2}
+-+
+whitebefore-blackstrip+ {'1': 0.9, '0': 0.1}
+-+
+blackstrip-blackstrip+ {'1': 0.9, '0': 0.1}
+-+
+blackstrip-whiteafter+ {'1': 0.1, '0': 0.9}
+-+
+whiteafter-whiteafter+ {'0': 0.8, '1': 0.2}
+-+
```

Schreibt das in eine Datei, deren Inhalt ihr dem HMM-Konstruktor übergebt, ruft dessen `setUpInitDf`-Methode auf und lasst euch das Resultat der `getAnOutput`-Methode für, sagen wir, 30 Schritte ausgeben. Erklärt, was ihr seht.

(31.3)* Bei den meisten nützlichen Anwendungen von HMMs hat man als Trainingsdaten die versteckten Variablen zusammen mit der Ausgabe (z.B. handgetaggte Korpora). Unter diesen Umständen braucht man kein EM-Training, sondern kann einfach die relevanten Parameter des HMM (Übergangs- und Emissionswahrscheinlichkeiten) schätzen. Wir wollen das hier schlicht mit ML machen (warum wäre für das Pixelstreifen-Modell SGT nicht gut?).

Seht euch die Funktion `estimateModelParameters` an – sie will eine Liste von Listen der Art, wie sie `HMM.getAnOutput` zurückgibt. Erzeugt euch also einen Schwung Ausgaben, übergebt sie an `estimateModelParameters` und seht, wie gut die Schätzung so wird.

`estimateModelParameters` tut so, als würde es eine Eingabe für den Konstruktor von HMM ausgeben. Dabei fehlt allerdings die Anfangsverteilung. Vielleicht wollt ihr das ja noch nachrüsten?

(31.4) Um zu sehen, wie viel man von Viterbi erwarten kann, wollen wir jetzt die tatsächlich von einem gegebenen HMM durchlaufenen Zustandsfolgen mit dem vergleichen, was Viterbi ausgibt. Das wird von `probD` gemacht.

Versucht, zu verstehen, was diese Funktion tut und probiert sie dann aus. Damit die Ausgabe etwas übersichtlicher wird, ist es klug, die versteckten Zustände oben umzubenennen, etwa so:

```
{
"(": {"_": 0.1, "(": 0.9},
"_": {")": 0.1, "_": 0.9},
")": {")": 1},
}
---
{"(": 1}
---
+(-(+ {'0': 0.8, '1': 0.2}
+-+
+(-_+ {'1': 0.9, '0': 0.1}
+-+
+_-_+ {'1': 0.9, '0': 0.1}
+-+
+_-)+ {'1': 0.1, '0': 0.9}
+-+
+)-)+ {'0': 0.8, '1': 0.2}
+-+
```

Überlegt euch, warum es gar nicht zu erwarten ist, dass der Viterbi die tatsächliche Zustandsfolge immer findet. Versucht, die Irrtümer zu erklären.

Datei(en) im PDF-Anhang: hmm.py

32. Exkurs: Gaussian Mixtures

Gaussian mixture models

Leitfragen zu Abschnitt 32

- Was tut Clustering und wie hängt es mit Klassifikation zusammen?
- Wie sehen Normalverteilungen in mehreren Dimensionen aus? Was bedeuten die verschiedenen Möglichkeiten, die es da gibt?

Bevor wir das Problem des „besten“ Modells für eine Beobachtung angehen können, sehen wir zunächst ein – scheinbar ganz anderes – Problem an: Soft Clustering.

Idee des Clustering ist, zusammenhängende Datenpunkte zu erkennen. Anwendungen dafür gibt es viele: z.B. will man in der Spracherkennung „ähnliche“ Klänge gruppieren, bei der Disambiguierung word tokens den verschiedenen Lesungen des word types zuordnen, im information retrieval Dokumente klassifizieren usf. Die Idee dabei ist in der Regel, die Datensätze in Vektoren zu verwandeln, die feature vectors (für die Mixtures, die wir hier haben, ist das allerdings nicht notwendig – solange wir berechnen können, wie wahrscheinlich es ist, dass eine Beobachtung von einem „Cluster“ erzeugt wird, können wir rechnen).

Bei Dokumenten oder bei der Desambiguierung entspricht dabei jedem Eintrag im Vektor einfach, wie oft ein word type im Dokument oder in den Umgebungen der zu clusternden Wörter vorkommt – mithin braucht der Raum, in dem man clustert, so viele Dimensionen, wie man word types hat. Danach probiert man einfach, in irgendeinem Sinne benachbarte Punkte im Raum zusammenzufassen und von denen abzugrenzen, die nicht benachbart sind.

Wenn ihr findet, dass dieses Problem eng mit der Klassifikation verwandt ist, liegt ihr richtig. Etwas vereinfacht könnte man sagen, Clustering sei letztlich Klassifikation ohne Trainingsdaten.

Eine populäre Methode zum Clustering verwendet *Gaussian mixture models* (GMMs), Summen von gewichteten Normalverteilungen. Weil wir d-dimensionale Räume betrachten, brauchen wir

auch mehrdimensionale Normalverteilungen. Für uns reichen *Normalverteilungen mit diagonaler Kovarianzmatrix*:

Normalverteilungen mit diagonaler Kovarianzmatrix

$$\varphi(\vec{x};\vec{c},\vec{\sigma}) = \frac{\exp\left(-\sum_{i=1}^{d}\frac{(x_i-c_i)^2}{2\sigma_i^2}\right)}{\sqrt{(2\pi)^d\prod_{i=1}^{d}\sigma_i}}.$$

Das ist schlicht ein Produkt von d Normalverteilungen in den d Dimensionen (im Exponenten wird das Produkt zu einer Summe).

In Wirklichkeit ist die hier angegebene Familie von Normalverteilungen ein Spezialfall. Letztlich nämlich definieren die Werte von $\vec{\sigma}$ die Breiten der Gaußverteilungen in den verschiedenen Richtungen des Raumes. Wenn man sich das zunächst in der Ebene vorstellt, sieht man, dass die Linien gleicher Werte dieser Verteilungen etwas wie Ellipsen bilden, dass diese Ellipsen aber ihre Hauptachsen immer parallel zu den den Achsen des Koordinatensystems haben. Es ist aber einfach, sich ein paar Punkte auf die Ebene zu malen, die besser durch Normalverteilungen mit „schräg“ stehenden Ellipsen zu beschreiben wären. Was tun?

Die Lösung ist, von $\vec{\sigma}$ zu einer $d\times d$-Matrix Σ überzugehen (wären wir streng, hätten wir übrigens genau umgekehrt vorgehen müssen). Die Details werden hier etwas haarig, und wer sich dafür interessiert, sollte Vorlesungen in Linearer Algebra besuchen und dort besonders bei „Eigenwerttheorie“ aufpassen. Die Anwendung jedoch ist zwar recht rechenaufwändig, aber nicht wirklich schwer. Jedenfalls sehen diese von der Fixierung auf die Koordinatenachsen befreiten Normalverteilungen so aus:

$$\varphi(\vec{x};\vec{c},\Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\exp\left(-\frac{1}{2}(\vec{x}-\vec{c})^T\Sigma^{-1}(\vec{x}-\vec{c})\right).$$

Darin bezeichnen $|\Sigma|$ und Σ^{-1} die Determinante und die Matrixinverse von Σ (wiederum verweise ich auf die Lineare Algebra), das hochgestellte T bedeutet einfach Transposition (aus einem Spaltenvektor wird ein Zeilenvektor, was es hier braucht, um die Matrixmultiplikation durchführen zu können). Wer LA hinter sich hat, sollte recht schnell sehen, dass sich diese Formel auf die oben angegebene zurückführen lässt, wenn Σ nur Einträge in der Diagonalen hat, und daher kommt auch der Name der Verteilungen oben.

Wer wirklich optimale Anpassung einer mehrdimensionalen Normalverteilung auf gegebene Daten haben möchte, kommt natürlich nicht um die Verwendung der „richtigen“ Normalverteilung herum. In den meisten Fällen sind aber die diagonalen Kovarianzmatrizen „gut genug“ (d.h. bieten einen guten Kompromiss aus Anpassungsvermögen und Einfachheit).

Eine Gaussian Mixture Θ ist nun ein Modell von N solchen Verteilungen, die noch gewichtet sein können. Es ist die Gesamtwahrscheinlichkeit eines Datenpunktes $\vec{x}$ unter Θ:

$$P(\vec{x}\,|\,\Theta) = \sum_{i=1}^{N}\pi_i\varphi(\vec{x};\vec{c}_i,\vec{\sigma}_i).$$

Darin steht π_i für die a-priori-Wahrscheinlichkeit, dass gerade die i-te Verteilung „zum Zug“ kommt. Das Bild ist hier wieder, dass die Beobachtung vom Modell erzeugt wurde – wenn nun eine Komponente der mixture nur ein kleines π_i hat, wird sie nur seltener ausgewählt und weniger Punkte werden von ihr erzeugt.

Wir haben jetzt eine Beobachtung $O=\{x_i, i=1\ldots K\}$. Die beste Erklärung dieser Beobachtung durch ein GMM finden wir durch

$$\Theta_{\mathsf{max}} = \operatorname*{argmax}_{\Theta} P(\Theta\,|\,O).$$

Die Maximierung läuft dabei normalerweise über alle möglichen GMMs mit der gewählten Zahl von Komponenten und d Dimensionen. Es ist aber auch durchaus denkbar, verschiedene Komponentenzahlen zuzulassen. Wenn wir keine speziellen Erwartungen an Θ haben – dies ist der

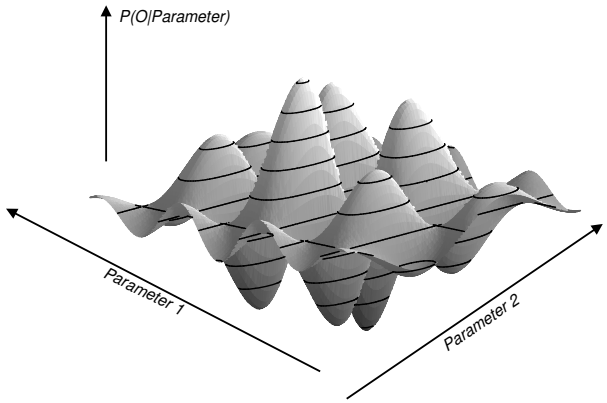

Fig. 22

Normalfall, wenn wir nur GMMs gleicher Länge betrachten –, also der Prior $P(\Theta)$ die Gleichverteilung ist, können wird auch gleich $P(O \mid \Theta)$ maximieren.

Das Problem:

(cf. Fig. 22)

Schon bei zwei Parametern kann die „Landschaft“, in der wir nach dem Maximum suchen, beliebig kompliziert sein. Es gibt kein analytisches Verfahren zur Suche nach dem Maximum.

33. Exkurs: Expectation Maximization

Leitfragen zu Abschnitt 33

- Warum rechnet man die beste Gaussian Mixture für eine Beobachtung nicht einfach aus?
- Warum führt Expectation Maximization zu einem „guten“ Ergebnis und warum kann es gut eines sein, das wir nicht haben möchten?

Wenn wir

$$\Theta_{\text{max}} = \operatorname*{argmax}_{\Theta} P(\Theta \mid O)$$

lösen wollen, müssen wir wissen, welche Datenpunkte zu welchen Gaussians gehören. Um das zu bestimmen, müssten wir allerdings die Gaussians schon kennen.

Ausweg aus diesem Henne-Ei-Problem: *Expectation Maximization*. Wir nehmen einfach ein paar Gaussians an und bestimmen daraus die Wahrscheinlichkeiten für die Zugehörigkeit (E-Step):

$$h_{ij} = \frac{P(\vec{x}_i \mid \varphi_j)}{\sum_{j=1}^{N} P(\vec{x}_i \mid \varphi_j)}.$$

$P(\vec{x}_i \mid \varphi_j)$ ist dabei einfach als der Wert von φ_j an der Stelle x_i zu berechnen, und der Nenner ist eine schlichte Normierung, die man in tatsächlichen Implementationen besser im Nachhinein macht.

Mit diesen Daten maximieren wir die Likelihood (M-Step). Das neue Zentrum eines φ_j ist das mit den Mitgliedswahrscheinlichkeiten gewichtete Mittel der Positionen der Datenpunkte:

$$\vec{c}'_j = \frac{\sum_{i=1}^{K} h_{ij}\vec{x}_i}{\sum_{i=1}^{K} h_{ij}}.$$

Die neuen Breiten ergeben sich aus dem gewichteten Mittel der Quadrate der *neuen* Ablagen:

$$\sigma'_{jk} = \sqrt{\frac{\sum_{i=1}^{K} h_{ij}(x_{ik} - c'_{jk})^2}{\sum_{i=1}^{K} h_{ij}}}.$$

Hier soll σ_{jk} die k-te Komponente von $\vec{\sigma}_j$ bedeuten und analog für die übrigen Vektoren.

Wenn man so abschätzt und das entstandene Modell Θ' nennt, kann man zeigen:

$$P(O \mid \Theta') \geq P(O \mid \Theta).$$

Bei jeder Anwendung von EM sollte sich also die Likelihood der Punkte unter dem Modell erhöhen (oder jedenfalls nicht schrumpfen). Damit läuft man immer auf einen „Gipfel“ der Landschaft auf der letzten Folie zu.

Es ist sehr gut möglich, dass dies nicht das globale Maximum ist – das hängt letztlich von der Stelle ab, an der man anfängt. Deshalb ist bei EM ein „gutes“ Anfangsmodell häufig extrem wichtig, um ein „Hängenbleiben“ in einem lokalen Maximum zu vermeiden.

Mehr zur Frage der Abhängigkeit des Ergebnisses der EM von den Anfangswerten kann man beim Spielen mit den Programmen im Anhang erfahren – im Allgemeinen muss die Wahl der Anfangswerte mit domänenspezifischen Methoden erfolgen. Häufig kann ein weniger empfindliches Verfahren wie K-Means helfen. Aber dies ist nicht mehr Thema dieser Veranstaltung.

Aufgaben

(33.1)* Holt euch das Programm `gaussMixtures.py` von der Webseite zur Vorlesung.

Macht euch zunächst mit der `Gaussian`-Klasse vertraut; sie implementiert im wesentlichen die mehrdimensionalen Normalverteilungen aus dem Skript.

Ein paar Tricks sind aber schon dabei: Erstens kocht `setSigmas` die Standardabweichungen vor, so dass die Berechnung des Wertes (in der `__call__`-Methode, die bewirkt, dass man einen Gaussian einfach als Funktion verwenden kann) nicht unnötig viel rechnen muss. Dann kann man eine Grenze angeben, unter die keine Komponente von Sigma schrumpfen kann. Das ist ein Trick, um zu verhindern, dass das Programm stirbt, wenn sich ein Gaussian auf einen einzelnen Punkt festfrisst (was er so oder so besser nicht tun sollte…). Die `draw`-Methode verwendet eine Funktion aus der random-Bibliothek von Python, um Punkte zu erzeugen, die, wenn man oft genug zieht, entsprechend dem Gaussian verteilt sind. Das geht hier einfach, weil wir effektiv unabhängige eindimensionale Verteilungen haben. Darüber hinaus kann ein Gaussian, wenn man ihm in der `update`-Methode sagt, für welche Punkte er sich wie verantwortlich fühlen soll, ein neues Zentrum und neue Sigmas für sich berechnen (das ist der Maximize-Step aus dem EM-Algorithmus)

(33.2)* Seht euch die `DistMixture`-Klasse an. Neben etlichen technischen Geschichten (`addDistribution`, `removeDistribution`, `__iter__`) ist hier die Logik für die EM enthalten. Seht euch `_runEStep` und `_runMStep` an und versucht zu sehen, dass diese gemeinsam mit `Gaussian.update` das tun, was im Skript beschrieben wurde. Wichtig dazu ist die Funktion `computeWeightedMean`, die eben ein gewichtetes Mittel ihres ersten Arguments berechnet. Dass wir hier mit den Punkten (also mit Vektoren) so bequem rechnen können, liegt daran, dass wir in `Point` die einschlägigen Operatoren Pythons so umdefiniert haben, dass sie das tun, was wir für Punkte gerne hätten. Die Details sind aber nicht so wichtig.

Schließlich solltet ihr noch einen Blick auf `getTotaLogL` werfen, das die log-likelihood für eine Punktmenge unter dem Modell liefert, und auf `getMostLikelyCluster`, das einfach den Gaussian zurückgibt, der für den übergebenen Punkt p den größten Wert hat (also $\text{argmax}_\varphi\, \varphi(p)$).

(33.3)* Sorgt dafür, dass am Fuß des Programms die Funktion `testWithGUI` aufgerufen wird und lasst das Programm laufen. `testWithGui` baut eine `GaussianMixture`, die absichtlich etwas ambig ist; die Komponenten dieser Mixture seht ihr als farbige Ellipsen, die die 1-σ-Linien repräsentieren. Dazu werden 100 Punkte erzeugt, die nach dieser GM verteilt sind. Seht euch zunächst an, was die EM in dieser Situation tut – der SStepKnopf (oder die Leertaste) macht jeweils einen Schritt. Die log likelihood in der Statuszeile sollte dabei wachsen (sie wird negativ sein, der Betrag der Zahl sollte also abnehmen). Das wird in der Realität nicht immer so sein, was an Numerikproblemen liegt.

Nach einigen (in der Regel wenigen) Schritten sollte sich nicht mehr viel tun. Warum hat sich überhaupt etwas verändert? Tipp: Seht mal, wie viel sich tut, wenn ihr, bevor ihr die EM laufen lasst, mit dem Knopf „1000 Points“ entsprechend mehr Punkte erzeugt und also die Verteilung besser beschreibt. Mit unseren öden Poolmühlen ist es allerdings kein Spaß, dem Rechner zuzusehen (Re-Implementation in C könnte helfen:-)

(33.4) Vielleicht hilft es beim Verständnis der EM, sich die vermutete Zugehörigkeit von Punkten zu einzelnen Gaussians zeigen zu lassen. Klickt dafür auf den „Colour“-Knopf. Die Farben der Punkte geben danach den Grad der Zugehörigkeit der Punkte zu den verschiedenen Gaussians an. Die Farbdarstellung braucht allerdings viel Rechenzeit. Es kann sich lohnen, das Programm auf ella laufen zu lassen.

(33.5)* Wenn ihr mit der rechten Maustaste auf eine Ellipse klickt, könnt ihr den zugehörigen Gaussian löschen. Mit der linken Maustaste und ziehen könnt ihr neue Gaussians erzeugen. Auf diese Weise könnt ihr die Anfangsbedingungen der EM verändern.

Versucht, mit verschiedenen Anfangsbedingungen verschiedene Ergebnisse für die gleiche Punktmenge herauszukitzeln – technisch landet ihr in verschiedenen lokalen Maxima, wenn ihr das hinkriegt. Es sollte nicht schwierig sein, wenn doch, könnt ihr mit der mittleren Maustaste Punkte hinzufügen.

Lustig ist auch, einen Gaussian mit großer und einen mit kleiner Breite zu erzeugen und zu sehen, wie der große Punkte „hinter“ dem kleinen adoptiert („Colour“ – das funktioniert übrigens nur bis zu drei Gaussians, weil es nicht mehr Grundfarben zum Mischen gibt).

(33.6) Wir wollen jetzt eine Art „semantisches Clustering“ machen. Die Idee dabei ist, aus den Kontexten von Wörtern auf Bedeutungszusammenhänge zu schließen; Wörter werden also durch die word types, die in ihrer Nachbarschaft gefunden werden, dargestellt, und deren Frequenzen sind die Featurevektoren. Ein Programm, das Featurevektoren aus einem Korpus zieht, ist auf der Webseite zu bekommen, ist aber eher zum daheim Spielen gedacht. Für Tutoriumszwecke habe ich bereits Featurevektoren für die Wörter

blue horse apple stream car nut pear lake unhappy sad creek carriage green boat merry orange white river red happy

aus einem Jahrgang Project Gutenberg extrahiert und ebenfalls auf die Webseite gestellt (die erste Zeile erlaubt die Zuordung der einzelnen Dimensionen zu diesem Raum zu den Kontextwörtern, deren Counts sie angeben – die Vektoren sind außerdem bereits normalisiert, so dass wir effektiv in einem Raum der Dimension $|\text{Kontextwörter}|-1$ Clustern).

Probiert zunächst selbst, die Wörter oben nach „Verwandtschaft“ zu gruppieren.

Probiert es danach mit EM. Zuständig dafür ist die Funktion `testWithFeatureVectors`, in der ihr den Haufen langweiliger Hilfsfunktionen ignorieren könnt. Für unsere Zwecke interessant ist nur `buildMixture`. Die Funktion nimmt von der Kommandozeile die Namen der Anfangszentren der neuen Cluster (sowas ließe sich im Prinzip auch automatisch bestimmen, aber besser ist es immer, wenn man dem Rechner hier ein wenig hilft). Das erste Kommandozeilenargument ist der Name der Datei mit den Featurevektoren.

Probiert also, was

```
python gaussMixtures.py features  car sad apple river blue
```

ausgibt – das sind zunächst die log likelihoods, bis das Programm beschließt, dass sich die Situation beruhigt hat. Danach werden die Werte der einzelnen Gaussians für die einzelnen zu clusternden Worte ausgegeben und schließlich die Cluster selbst.

Probiert, was passiert wenn ihr andere Wörter angebt, oder mehr oder weniger davon. Ihr solltet sehen, dass die Anfangsbedingungen bei EM entscheidend wichtig sind und die $P(\Theta \mid O)$-Landschaft wirklich viele Hügel hat.

Datei(en) im PDF-Anhang: gaussMixtures.py getContexts.py features

34. Der Forward-Backward-Algorithmus

Leitfragen zu Abschnitt 34

- Wann braucht man den Forward-Backward-Algorithmus?
- Warum braucht man EM dafür?

Zurück zum letzten HMM-Problem: Wir haben eine Familie von HMMs $(\mu_\vartheta)_{\vartheta\in\Theta}$ sowie eine Beobachtung O. Wir wollen ein ϑ_0 finden, so dass $P(O \mid \mu_{\vartheta_0})$ maximal wird. Dabei ist für uns ϑ die Sammlung aller Werte von $\delta(i,j)$ und $\lambda(i,j,o)$.

Warum sollte man das wollen? Nun, beim Schwarz-Weiß-HMM beispielsweise haben wir die Übergangs- und Emissionswahrscheinlichkeiten aus dem Bauch heraus geschätzt. Tatsächlich ließe sich aber die „Verrauschtheit“ des Druckers aus Beobachtungen schätzen, so dass man ein quasi auf den vorliegenden Drucker optimiertes HMM erhält.

Man ahnt schon, dass dies stark auf EM hinausläuft, und das ist auch so. Das Äquivalent zu den Mitgliedswahrscheinlichkeiten der GMMs sind hier die „Wichtigkeiten“ der einzelnen Übergänge im HMM unter einer bestimmten Beobachtung. Sie lassen sich so berechnen, wenn wir – wie gehabt – schon ein Anfangsmodell haben:

Sei $p_t(i,j) = P(X_t = i, X_t+1 = j \mid O, \mu)$. In unseren Forward-Backward-Variablen ist

$$p_t(i,j) = \frac{\alpha_i(t)\beta_j(t+1)\delta(i,j)\lambda(i,j,o_t)}{\sum_{m=1}^{N}\alpha_m(t)\beta_m(t)}.$$

Woher kommt diese Formel? Zunächst ist

$$P(X_t = i, X_t+1 = j \mid O, \mu) = \frac{P(X_t = i, X_t+1 = j, O \mid \mu)}{P(O \mid \mu)}.$$

Die Wahrscheinlichkeit für einen Übergang von i zum Zeitpunkt t nach j zum Zeitpunkt $t+1$ unter Emission von o_t (also der Zähler) ist einfach die die Wahrscheinlichkeit, dass wir zu t im Zustand i sind $(\alpha_i(t))$ mal der Wahrscheinlichkeit, dass es von j aus weitergeht $(\beta_j(t+1))$ mal der Übergangswahrscheinlichkeit von i nach j $(\delta(i,j))$ mal der Wahrscheinlichkeit, dass bei diesem Übergang das richtige Zeichen emittiert wird $(\lambda(i,j,o_t))$. Der Nenner ist einfach wieder eine Normierung.

Im M-Step nehmen wir wieder eine geeignete Mittelung vor, diesmal über die verschiedenen Schritte des HMM.

$$\hat\delta(i,j) = \sum_{t=1}^{T} p_t(i,j) \Big/ \sum_{t=1}^{T}\gamma_i(t)$$

$$\hat\lambda(i,j,o) = \sum_{\substack{t=1\\ o_t=k}}^{N} p_t(i,j) \Big/ \sum_{t=1}^{T} p_t(i,j)$$

Es ist nicht ganz leicht, einzusehen, dass das Modell $\hat\mu = (\hat\delta, \hat\lambda)$ wirklich die Ungleichung

$$P(O \mid \hat\mu) \geq P(O \mid \mu)$$

erfüllt, man kann das aber zeigen.

Wie schon bei den GMMs heißt Konvergenz des Forward-Backward-Verfahrens noch lange nicht, dass wir das beste oder auch nur ein gutes Modell gefunden haben. Dazu kommt, dass es je nach Typ des HMM nichttrivial ist, gleichzeitig auf mehrere Beobachtungen zu trainieren (was aber natürlich in der Regel nötig ist). Deshalb: Training von HMMs nur mit Vorsicht und am Besten nach Studium einschlägiger Literatur (der Industriestandard dazu ist Mitchell: Machine Learning).

Es gibt übrigens etliche Alternativen zu Forward-Backward, wenn man HMMs anpassen will, etwa geeignet eingeschränkte Monte-Carlo-Methoden, bei denen zufällig Punkte im Parameterraum probiert werden, oder genetische Algorithmen, in denen durch Mutation und Selektion Parametersätze quasi evolutionär verbessert werden – in der Regel dürfte Forward-Backward aber der geeignetste Algorithmus sein.

Der Nutzen von HMMs

HMMs werden in der Computerlinguistik zu vielen Zwecken eingesetzt. Ihren Durchbruch hatten sie wohl in der Erkennung gesprochener Sprache. Dabei sind die Symbole, die das HMM emittiert, „Merkmalsvektoren", die durch geeignete Filter aus kleinen Stückchen der Sprachaufnahme gewonnen werden, während die Zustände etwa Phonen oder gar Phonemen entsprechen können (in der Realität ist das nicht ganz so einfach, und in der Regel ist die Menge der Merkmalsvektoren auch nicht diskret, so dass die Emissionswahrscheinlichkeiten aus Dichten bestimmt werden müssen – unsere Algorithmen funktionieren aber normal weiter).

Große Popularität haben HMMs auch im Tagging, d.h. in der Erkennung von Wortklassen (Nomen, Verb, Artikel,...). Dabei ist klar, dass in einem grammatisch korrekten Satz normalerweise nicht jede beliebige Wortart hinter jeder beliebigen anderen kommen wird – und dies ist die Abhängigkeit, so dass schon klar ist, dass die Zufallsvariablen Werte in den Wortklassen annehmen. Die Ausgabesymbole können tatsächlich die word tokens sein, womit allerdings die Zahl der zu schätzenden Parameter extrem groß wird.

Tatsächlich empfehle ich zum Taggen eine andere, mathematisch nicht ganz so wohlfundierte (und deshalb nicht hierhergehörende...), dafür aber linguistisch wohlmotivierte, Technik namens Transformation Based Learning. Die bekannteste Implementation eines TBL-Taggers ist der Brill-Tagger.