

Spezielle Methoden und Anwendungen der Statistik in der Bioinformatik

Sven Rahmann

Abteilung Bioinformatik
Max-Planck-Institut für Molekulare Genetik
Innstraße 73, 14195 Berlin
Sven.Rahmann@molgen.mpg.de
Tel: (030) 8413-1173, Fax: (030) 8413-1152

und

Fachbereich Mathematik und Informatik
Freie Universität Berlin

6. Mai 2003

Vorbemerkungen

Dieses Skript enthält den von mir gehaltenen Teil der Vorlesungen über “Spezielle Methoden der Statistik” und “Molekulare Evolution” im Rahmen des Bioinformatik-Kurses der Akademie für Weiterbildung der Universitäten Heidelberg und Mannheim, und darüber hinaus einige zusätzliche Informationen. Die hier behandelten Themen waren auch Teil der Vorlesung “Algorithmische Bioinformatik” im Wintersemester 2002/03 an der Freien Universität Berlin.

Im Vordergrund steht die schnelle Vermittlung der wichtigen Begriffe und Methoden. Manche Resultate werden nur erwähnt, aber weder bewiesen noch vertieft. Wer sich näher mit einem Thema befassen möchte, sollte die entsprechenden Kapitel in den folgenden Lehrbüchern anschauen.

- Eine anschauliche und wenig voraussetzende Einführung in viele Gebiete der Bioinformatik (und damit auch in das benötigte statistische Wissen) ist: David W. Mount. *Bioinformatics – Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press. Für den eiligen Leser ist dieses Buch allerdings etwas zu textlastig.
- Etwas abstrakter geht es zu in: Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press. Insbesondere die in meinem Skript knappe Diskussion über HMMs ist dort sehr gut gelungen.
- Wer sich weitergehender mit statistischen Methoden in der Bioinformatik beschäftigen will, sollte auf jeden Fall einen Blick werfen auf: Warren J. Ewens and Gregory R. Grant. *Statistical Methods in Bioinformatics – An Introduction*. Springer-Verlag. Leider ist dieses Buch etwas unübersichtlich und mühsam zu lesen wegen vieler Querreferenzen.

Für Hinweise über Fehler oder Unklarheiten in diesem Skript bin ich jederzeit dankbar.

- Sven Rahmann

Inhaltsverzeichnis

1	Theorie: Markovketten	4
1.1	Grundlagen	4
1.2	Typische Fragestellungen	5
2	Anwendung: Sequenzmodellierung	9
3	Anwendung: Hidden Markov Modelle (HMMs)	10
4	Theorie: Markovprozesse	11
4.1	Grundlagen	11
4.2	Typische Fragestellungen	13
5	Anwedung: Scorematrizen	14
6	Theorie: Wichtige evolutionäre DNA-Modelle	15
6.1	Jukes-Cantor (JC)	15
6.2	Kimura 2-Parameter (K2P)	16
6.3	Weitere Modelle	17
7	Theorie: Zeitschätzung mit EMPs	17
7.1	Die Jukes-Cantor-Korrektur	18
7.2	Die Kimura-Formel	19
8	Anwendung: ML-Phylogenie-Rekonstruktion	19
8.1	Übersicht	19
8.2	Optimierung der Log-Likelihood eines Baumes	20
9	Theorie: Bootstrapping von Phylogenie-Algorithmen	21
9.1	Der Bootstrap	22
9.2	Bootstrapping von Bäumen	22
10	Theorie: Statistik von Sequenzalignments	23
10.1	Motivation und Grundlagen	23
10.2	Nullmodelle	25
10.3	BLAST-Statistik und Karlin-Altschul Theorie	26
10.4	Query-spezifische Signifikanzwerte	27
10.5	Mehrfaches Testen und E-Values	28
11	Theorie: Verfahren des Maschinenlernens	29
11.1	Grundbegriffe	29
11.2	Methoden	30

1 Theorie: Markovketten

1.1 Grundlagen

Markovketten sind vielseitige stochastische Prozesse. In der Bioinformatik sind sie in mindestens drei verschiedenen Bereichen von Bedeutung.

1. Modellierung von Sequenzevolution
2. Modellierung von Zufallssequenzen (als sogenannte Nullmodelle)
3. Grundlage für HMMs (Hidden-Markov-Modelle), die in der Sequenzanalyse unentbehrlich geworden sind.

Eine Markovkette beschreibt eine Größe, die mehrere Zustände annehmen kann, im Verlauf der Zeit. Man ist daran interessiert, wie sich der Zustand im Laufe der Zeit verändert. Die Übergänge zwischen den Zuständen werden stochastisch modelliert. Die grundlegende Eigenschaft der Markovketten ist dabei die *Gedächtnislosigkeit* (oder *Markov-Eigenschaft*) der Zustände. Das heißt, die Übergangswahrscheinlichkeiten von einem Zustand in den nächsten hängen nur vom derzeitigen Zustand ab, nicht aber von vorherigen.

Beispiel 1.1. Wir betrachten ein Nukleotid an einer festen Position in einer Sequenz im Laufe der Evolution. Die möglichen Zustände sind also A,G,C,T. Als Zeiteinheit wählen wir beispielsweise eine Million Jahre. Im Ausgangszustand liege ein A vor. Den Prozess beschreiben wir durch folgende Übergangswahrscheinlichkeiten: Wenn zur Zeit t ein bestimmtes Nukleotid vorliegt, so liegt zur Zeit $t + 1$ dasselbe Nukleotid mit Wahrscheinlichkeit 0.97 vor, und eines der drei anderen jeweils mit Wahrscheinlichkeit 0.01.

Die Daten werden in der sogenannten *Übergangsmatrix* P zusammengefasst.

$$P = \begin{pmatrix} & \rightarrow A & \rightarrow G & \rightarrow C & \rightarrow T \\ A \rightarrow & 0.97 & 0.01 & 0.01 & 0.01 \\ G \rightarrow & 0.01 & 0.97 & 0.01 & 0.01 \\ C \rightarrow & 0.01 & 0.01 & 0.97 & 0.01 \\ T \rightarrow & 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}$$

Die Reihenfolge der Nukleotide spielt im Prinzip keine Rolle. Wir wählen die von der alphabetsichen Ordnung abweichende Reihenfolge AGCT, weil sie die Purine AG von den Pyrimidinen CT trennt. Die gewählte Reihenfolge muss jedoch für Zeilen und Spalten dieselbe sein.

Übung 1. Diskutieren Sie die Annahme, dass die Markoveigenschaft im Rahmen der Sequenzevolution gilt.

Das Beispiel zeigt: Um eine Markovkette zu beschreiben, brauchen wir eine Reihe von Daten.

Definition 1.2 (Stochastischer Prozess). Ein stochastischer Prozess ist eine Zufallsvariable $X(t)$ im Verlauf der Zeit t . Die Zufallsvariable nimmt zu jeder Zeit Werte aus einem vorgegebenen Zustandsraum \mathcal{X} an. In den folgenden Betrachtungen ist dieser stets endlich und besteht aus n Zuständen, die wir oft einfach mit $\{1, 2, \dots, n\}$ bezeichnen (z.B. A=1, G=2, C=3, T=4). Die Zeit ist entweder diskret ($t = 0, 1, 2, \dots$) oder stetig ($t \geq 0$). Wahrscheinlichkeits-Aussagen über $X(t)$ schreiben wir mit dem Symbol \mathbb{P} ; beispielsweise bedeutet $\mathbb{P}(X(0) = A)$ die Wahrscheinlichkeit, dass A das Startsymbol ist.

Definition 1.3 (Markovkette). Eine (zeitdiskrete und homogene) *Markovkette* ist ein stochastischer Prozess, der vollständig durch eine *Startverteilung* $\pi^{(0)}$ auf \mathcal{X} und eine *Übergangsmatrix* P zwischen

den Zuständen gegeben ist. Der Wert P_{ij} gibt die bedingte Wahrscheinlichkeit an, zum nächsten Zeitpunkt in Zustand j zu landen, wenn i der jetzige Zustand ist. Insbesondere ist diese Wahrscheinlichkeit unabhängig von weiter zurückliegenden Zuständen (Gedächtnislosigkeit).

$$\pi_i^{(0)} = \mathbb{P}(X(0) = i)$$

$$P_{ij} = \mathbb{P}(X(t+1) = j \mid X(t) = i)$$

Übung 2. Zeigen Sie, dass die Zeilensummen der Matrix P stets 1 ergeben müssen. Eine solche Matrix heißt auch *stochastische Matrix*.

Verteilungen stellen wir grundsätzlich als Zeilenvektoren dar. In diesem Sinne ist eine Verteilung auf n Zuständen ein Zeilenvektor der Länge n mit nichtnegativen Einträgen, die sich zu 1 summieren. Die i -te Zeile von P gibt die Verteilung des nächsten Zustandes an, wenn die Kette gegenwärtig im Zustand i ist.

Die Beschreibung von zeitstetigen Markovketten (wir nennen sie Markovprozesse) lässt sich nicht mit einer schrittweisen Übergangsmatrix bewerkstelligen. Hier ist mehr Werkzeug aus der Analysis erforderlich. Wir kommen unten darauf zurück.

Wir haben homogene Markovketten definiert, d.h., die Übergangsmatrix bleibt zu jeder Zeit dieselbe. Es lassen sich auch nichthomogene Markovketten definieren; diese sind allerdings weniger wichtig. Wir werden sie nicht betrachten.

1.2 Typische Fragestellungen

Typische Fragestellungen im Zusammenhang mit Markovketten sind:

Problem 1.4. “Heute steht hier ein A, was steht dort vermutlich in 10 Millionen Jahren?” Also: Gegeben $X(t) = i$, wie ist die Verteilung von $X(t+k)$ für ein $k \geq 1$? Oder anders gefragt: Gegeben die 1-Schritt Übergangswahrscheinlichkeiten P_{ij} , was sind die k -Schritt Übergangswahrscheinlichkeiten $P_{ij}^{(k)}$?

Problem 1.5. “Wie häufig sieht man im Mittel jedes Nukleotid A,C,G,T?” Hier wird nach der Zustandsverteilung im Zeitmittel gefragt. Oft ist diese Frage äquivalent zur Frage nach einer stationären Verteilung, d.h., einer Verteilung π , die sich durch Anwenden der Übergangsmatrix P nicht ändert.

Problem 1.6. “Das Modell sieht sehr symmetrisch aus. Sind Mutationswahrscheinlichkeiten nicht eher unterschiedlich für Transitionen/Tranversionen?” Woher kommen die Übergangswahrscheinlichkeiten, d.h., wie findet man die Parameter des Markovmodells, wenn (eine) Beobachtungsreihe(n) von Zuständen gegeben sind?

Während sich die beiden ersten Fragen aus den Modellparametern heraus beantworten lassen, bezieht sich diese Frage auf die Schätzung eines Modells aus beobachteten Daten. Wir wollen die aufgeworfenen Probleme für Markovketten lösen; zunächst Problem 1.4.

Übung 3. Überlegen Sie sich, wie wohl sinnvollerweise $P^{(0)}$, die Matrix der 0-Schritt Übergangswahrscheinlichkeiten, aussieht.

Berechnung der Mehr-Schritt-Übergangswahrscheinlichkeiten. Fangen wir mit 2 Schritten an. Um in 2 Schritten von i nach j zu kommen, muss man im ersten Schritt von i irgendwohin (sagen wir, nach y), und dann im zweiten Schritt von y nach j gelangen. Man kann über jedes y gehen, und

wenn man erst einmal nach y gelangt ist, ist es wegen der Gedächtnislosigkeit unerheblich, dass man vorher in i war. Es folgt:

$$\begin{aligned} & \mathbb{P}[X(2) = j \mid X(0) = i] \\ &= \sum_y \mathbb{P}[X(2) = j, X(1) = y \mid X(0) = i] \\ &= \sum_y \mathbb{P}[X(1) = y \mid X(0) = i] \cdot \mathbb{P}[X(2) = j \mid X(1) = y, X(0) = i], \end{aligned}$$

also

$$P_{ij}^{(2)} = \sum_y P_{iy} \cdot P_{yj},$$

oder geschrieben als Matrixmultiplikation,

$$P^{(2)} = P \cdot P = P^2.$$

Analog lässt sich $P^{(k)}$ durch Matrixmultiplikation von $P^{(k-1)}$ und P berechnen.

$$P_{ij}^{(k)} = \sum_y P_{iy}^{(k-1)} \cdot P_{yj} \quad \text{oder} \quad P^{(k)} = P^{(k-1)} \cdot P = P^k,$$

die k -te Potenz von P .

Übung 4. Berechnen Sie die 10-Schritt Übergangsmatrix im obigen Evolutionsbeispiel.

$$P^{(10)} \approx \begin{pmatrix} 0.7486 & 0.0838 & 0.0838 & 0.0838 \\ 0.0838 & 0.7486 & 0.0838 & 0.0838 \\ 0.0838 & 0.0838 & 0.7486 & 0.0838 \\ 0.0838 & 0.0838 & 0.0838 & 0.7486 \end{pmatrix}.$$

Das Nukleotid, das ich heute beobachte, steht auch in 10 Millionen Jahren mit einer Wahrscheinlichkeit von 74.86% noch dort; das heißt nicht, dass es sich im Laufe dieser Zeit nicht mehrfach geändert haben kann.

Jetzt wird es ein wenig abstrakter. Angenommen, wir können das Nukleotid nicht direkt beobachten und wissen beispielsweise nur, dass es sich um ein Purin (A oder G) handelt. Mit gleicher Wahrscheinlichkeit liegt ein A oder ein G vor. Der momentane Zustand lässt sich also nicht durch ein Zustandssymbol, sondern nur durch eine Zustandsverteilung $\pi^{(0)}$ beschreiben, hier $\pi^{(0)} = (0.5, 0.5, 0, 0)$. Vollständig charakterisierte Zustände sind spezielle Verteilungen (sogenannte Dirac-Maße, z.B. $\mathbf{T} = (0, 0, 0, 1)$).

Wir fragen wieder: Wie sieht mit dieser Information die Nukleotidverteilung nach 10 Millionen Jahren aus? Für Dirac-Maße erhalten wir die Antwort unmittelbar aus der entsprechenden Zeile von $P^{(10)}$. Jetzt liegt eine "gewichtete Mischung" (das Fachwort lautet *Konvexkombination*) von Zuständen vor, und es ist intuitiv, dass sich die gesuchte Verteilung als gewichtete Summe über die Zeilen von $P^{(10)}$ berechnen lässt. Das folgt aus dem Satz von der totalen Wahrscheinlichkeit.

Allgemein sei $\pi^{(k)}$ die Verteilung nach k Schritten, wobei $\pi^{(0)}$ die Startverteilung ist. Dann gilt

$$\pi_j^{(k)} = \sum_i \pi_i^{(0)} \cdot P_{ij}^{(k)},$$

oder als Zeilenvektor-Matrix-Multiplikation geschrieben,

$$\pi^{(k)} = \pi^{(0)} \cdot P^{(k)}.$$

In unserem Beispiel erhalten wir

$$(0.5, 0.5, 0, 0) \cdot \begin{pmatrix} 0.7486 & 0.0838 & 0.0838 & 0.0838 \\ 0.0838 & 0.7486 & 0.0838 & 0.0838 \\ 0.0838 & 0.0838 & 0.7486 & 0.0838 \\ 0.0838 & 0.0838 & 0.0838 & 0.7486 \end{pmatrix} = (0.4162, 0.4162, 0.0838, 0.0838).$$

Übung 5. Zeigen Sie: Ist π eine Verteilung und P eine stochastische Matrix, so ist auch $\pi \cdot P$ eine Verteilung. Sind P und Q stochastische Matrizen, so ist auch $P \cdot Q$ eine stochastische Matrix. (Tipp für den ersten Teil: Sei $\tau := \pi \cdot P$. Man muss zeigen, dass die Komponenten von τ nichtnegativ sind und sich zu 1 summieren unter der Voraussetzung, dass dasselbe für π und jede Zeile von P gilt.)

Zusammenfassend halten wir fest: Die k -Schritt Übergangsmatrix $P^{(k)}$ überführt eine Zustandsverteilung $\pi^{(t)}$ zum Zeitpunkt t in die daraus folgende Zustandsverteilung $\pi^{(t+k)}$ zum Zeitpunkt $t+k$. Dies geschieht einfach durch Matrixmultiplikation von rechts,

$$\pi^{(t)} \cdot P^{(k)} = \pi^{(t+k)}.$$

Es gilt auch die sogenannte *Chapman-Kolmogorov-Gleichung*

$$P^{(s)} \cdot P^{(t)} = P^{(s+t)}.$$

Statt nacheinander einen Zeit- s - und einen Zeit- t -Übergang zu betrachten, kann man einen einzigen Zeit- $(s+t)$ -Übergang betrachten.

Stationäre Verteilungen. Um Problem 1.5 anzugehen, müsste man eine ganze Menge mathematischer Theorie behandeln. Uns genügt aber für die meisten Anwendungen das folgende Ergebnis, das wir nicht beweisen wollen.

Satz 1.7. Gibt es ein t , so dass $P_{ij}^{(t)} > 0$ für alle i, j ist (in der Regel gilt das bei unseren Anwendungen schon für $t = 1$, d.h., alle Übergänge sind in einem Schritt möglich), dann existiert genau eine Verteilung π , die unter der Übergangsmatrix invariant ist, d.h., für die $\pi \cdot P = \pi$ gilt. In diesem Fall heißt π die *stationäre Verteilung* zu P .

Unabhängig von der Startverteilung $\pi^{(0)}$ konvergiert die Verteilung nach t Schritten $\pi^{(t)} = \pi^{(0)} \cdot P^t$ gegen π für $t \rightarrow \infty$. Es gilt also

$$\lim_{t \rightarrow \infty} P_{ij}^{(t)} = \pi_j \quad \text{für alle } i.$$

Die stationäre Verteilung ist damit auch die Verteilung im Zeitmittel.

Zur Bestimmung von π muss man also im allgemeinen die Gleichung $\pi \cdot P = \pi$ oder $\pi \cdot (P - \text{Id}) = 0$ oder

$$(P - \text{Id})^T \cdot \pi^T = 0$$

lösen (das hochgestellte T bedeutet "transponiert").

Das ist ein lineares Gleichungssystem mit n Unbekannten (den Einträgen von π). Allerdings müssen sich diese zu 1 summieren, d.h., man hat eigentlich nur $n - 1$ Unbekannte. Das ist gut so, denn die Koeffizientenmatrix $(P - \text{Id})^T$ hat auch nur den Rang $n - 1$. Das heißt, man kann eine beliebige Zeile der Matrix streichen und durch die Nebenbedingung ersetzen. Im Evolutionsbeispiel ergibt sich

$$\begin{pmatrix} -0.03 & 0.01 & 0.01 & 0.01 \\ 0.01 & -0.03 & 0.01 & 0.01 \\ 0.01 & 0.01 & -0.03 & 0.01 \\ 0.01 & 0.01 & 0.01 & -0.03 \end{pmatrix} \cdot \begin{pmatrix} \pi_A \\ \pi_G \\ \pi_C \\ \pi_T \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \sum_i \pi_i = 1.$$

oder

$$\begin{pmatrix} -0.03 & 0.01 & 0.01 & 0.01 \\ 0.01 & -0.03 & 0.01 & 0.01 \\ 0.01 & 0.01 & -0.03 & 0.01 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \pi_A \\ \pi_G \\ \pi_C \\ \pi_T \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Wir wollen uns jedoch nicht zu sehr mit dem Lösen linearer Gleichungssysteme befassen und begnügen uns im weiteren mit dem folgenden wichtigen Resultat.

Satz 1.8. Ist die Übergangsmatrix P symmetrisch, so ist die Gleichverteilung stationär.

Der Beweis ist sehr einfach: Es sei π die Gleichverteilung mit $\pi_i = 1/n$ für alle i . Wir betrachten das j -te Element von $\pi \cdot P$. Es ist $(\pi \cdot P)_j = \sum_i \pi_i P_{ij} = 1/n \sum_i P_{ij} = 1/n \sum_i P_{ji} = 1/n$ für alle j , also ist $\pi \cdot P$ wieder die Gleichverteilung.

Parameterschätzung in Markovketten. Wir kommen jetzt zu Problem 1.6. Angenommen, wir beobachten nacheinander folgende Zustände (jeweils im Abstand einer Zeiteinheit):

AAGGATAATCCGTCCAAACCCATCTTCCCCGGAATGCGTTGGGAA

Wir wollen die Startverteilung, die stationäre Verteilung und die Übergangsmatrix schätzen.

- Die Startverteilung kann aus dieser einen Reihe schlecht geschätzt werden. Man könnte A als festen Startpunkt wählen, also $\pi^{(0)} = (1, 0, 0, 0)$, oder aber aufgrund von Informationsmangel sich für die Gleichverteilung $\pi^{(0)} = (1/4, 1/4, 1/4, 1/4)$ entscheiden oder stationär starten, d.h., die stationäre Verteilung als Startverteilung wählen.
- Die stationäre Verteilung lässt sich am einfachsten durch Zählen schätzen. Das obige Beispiel hat eine Länge von 45. Wir beobachten 13 mal A, 10 mal G, 13 mal C, und 9 mal T, also schätzen wir

$$\pi = \left(\frac{13}{45}, \frac{10}{45}, \frac{13}{45}, \frac{9}{45} \right) \approx (0.2889, 0.2222, 0.2889, 0.2000).$$

Diese Schätzung beruht auf der Annahme, dass die beobachteten Daten von Anfang an von dem stationären Prozess generiert wurden.

- Auch die 16 möglichen Übergänge lassen sich einfach schätzen, indem man sie zählt. Als Anzahlen erhält man:

$$C = \begin{pmatrix} & \rightarrow A & \rightarrow G & \rightarrow C & \rightarrow T \\ A \rightarrow & 6 & 1 & 1 & 4 \\ G \rightarrow & 3 & 4 & 1 & 2 \\ C \rightarrow & 2 & 3 & 7 & 1 \\ T \rightarrow & 1 & 2 & 4 & 2 \end{pmatrix}$$

$$P = \begin{pmatrix} & \rightarrow A & \rightarrow G & \rightarrow C & \rightarrow T \\ A \rightarrow & 0.5000 & 0.0833 & 0.0833 & 0.3333 \\ G \rightarrow & 0.3000 & 0.4000 & 0.1000 & 0.2000 \\ C \rightarrow & 0.1538 & 0.2308 & 0.5385 & 0.0769 \\ T \rightarrow & 0.1111 & 0.2222 & 0.4444 & 0.2222 \end{pmatrix}$$

Aus diesem geschätzten P lässt sich wiederum die zugehörige stationäre Verteilung berechnen. Man erhält

$$\pi \approx (0.2727, 0.2273, 0.2955, 0.2045).$$

Diese Schätzung entspricht ungefähr, aber nicht genau, der obigen.

Es lässt sich zeigen, dass das genannte Schätzverfahren für P der Maximum-Likelihood-Schätzer ist. Es ist jedoch üblich, zusätzlich zu den beobachteten Übergängen sogenannte Pseudo-Counts hinzuzugaddieren, z.B. um Nullen in der Übergangsmatrix zu vermeiden.

Übung 6. Eine andere denkbare Schätzmethode für P ist, statt der direkten Übergänge die 2-Schritt Übergänge zu zählen. Dies liefert eine direkte Schätzung für $P^{(2)}$. Man könnte versuchen, daraus P zu bestimmen. Diskutieren Sie mögliche Vor- und Nachteile dieser Methode.

2 Anwendung: Sequenzmodellierung

Markovketten werden nicht nur zur Modellierung von Evolution, sondern auch zur “horizontalen” DNA- oder Protein-Sequenzmodellierung eingesetzt. Dabei entspricht der “Zeit”-Parameter t der Position in der Sequenz.

Der Gedanke ist dabei meist folgender: Man hat Beispielsequenzen, in denen ein bestimmtes Signal auftritt (Positivbeispiele). Dieses Signal ist oft stochastischer Natur, etwa ein Sequenzprofil (engl. Profile). Mit Hilfe der Positivbeispiele wird ein Algorithmus entwickelt, der dieses Signal detektieren kann. Der Algorithmus bekommt als Eingabe eine unbekannte Sequenz und liefert als Ausgabe eine Entscheidung, ob das Signal in der Sequenz entdeckt wurde, oder die Positionen, an denen das Signal auftritt. In den meisten Fällen berechnet der Algorithmus eine Punktzahl oder einen Score, anhand dessen die Entscheidung getroffen wird. Natürlich führen aber auch zufällig ausgewürfelte Sequenzen zu einer Punktzahl; in den meisten Fällen sollte diese deutlich niedriger als bei den echten Signalen sein. Das heißt, ist der Score größer als ein Schwellenwert t , so entscheidet man sich für die Präsenz des Signals, ansonsten dagegen. Man steht dabei vor dem Problem, dass man den Schwellenwert t vernünftig wählen muss. Er sollte hoch genug sein, so dass die Scores der meisten zufälligen Sequenzen kleiner als t sind und die Scores der meisten echten Signale größer als t sind. Die zweite Bedingung lässt sich schwer überprüfen, weil man ja davon ausgeht, dass man viele Instanzen der Signale noch gar nicht kennt und nur wenige Beispiele hat. Zufällige Sequenzen kann man sich dagegen leicht beschaffen. Die Frage ist nur, nach welchem Modell?

Hier kommt die Sequenzmodellierung durch Markovketten ins Spiel. Aus den Positivbeispielen schätzt man zunächst ein Markov-Modell. Dieses wird dann zur Generierung oder Analyse neuer zufälliger Sequenzen verwendet. Da man eine einmal geschätzte Markovkette dazu benutzen kann, neue Sequenzen zu generieren, spricht man auch von einem *generativen Modell*. Die so generierten Sequenzen sind mit den ursprünglichen nicht verwandt, entsprechen aber in der Komposition ihren Vorbildern. Insbesondere werden die Paarhäufigkeiten bis auf zufällige Schwankungen erhalten. Die Zufallssequenzen benutzt man wie oben angedeutet als Negativbeispiele. Sie enthalten in der Regel nicht dasselbe Signal wie die Positivbeispiele, aus denen sie generiert wurden, behalten aber ansonsten viele Eigenschaften. Man kann jetzt viele solcher Negativbeispiele erzeugen und den Schwellenwert so wählen, dass z.B. nur 0.1% der Negativbeispiele einen höhere Score erreichen. So kann man effektiv falsch positive Signaldetektion ausschließen, aber schlecht kontrollieren, wie viele Sequenzen man fälschlich als negativ klassifiziert.

Übung 7. Ein noch einfacheres Modell für zufällige Sequenzen besteht darin, jede Position unabhängig von der vorhergehenden nach einer festen Verteilung auszuwürfeln. Man spricht von einem iid Modell (independently identically distributed = unabhängig und identisch verteilt). Ein iid Modell lässt sich auch als spezielle Markovkette darstellen. Wie sieht P in diesem Fall aus?

Wir betrachten noch ein zweites Beispiel für Sequenzmodellierung aus der Genomanalyse. Wenn man bei einem Organismus schon einen Teil der Gene annotiert hat und Exons und Introns kennt, kann man getrennt für Exons und Introns je eine Markovkette schätzen. Ist ein Stück unannotierte DNA gegeben, kann man einfach testen, welches der beiden Modelle besser passt. Man rechnet für

beide Modelle die Wahrscheinlichkeit aus, gerade dieses Stück zu generieren. Ist die Wahrscheinlichkeit im Intron-Modell wesentlich höher, wird man dieses Stück daraufhin als Intron klassifizieren. Zugegebenermaßen ist diese Methode für eine realistische Anwendung viel zu grob, aber in den Anfangszeiten der Sequenzanalyse wurden solche Verfahren durchaus eingesetzt.

3 Anwendung: Hidden Markov Modelle (HMMs)

Hidden Markov Modelle (HMMs) wurden Mitte der neunziger Jahre in der Bioinformatik populär; eigentlich stammen sie aus der Spracherkennung. Die Idee ist ebenso einfach wie mächtig: Beim Schätzen der Parameter einer Markovkette sind wir davon ausgegangen, dass wir den Zustand direkt beobachten konnten. Bei HMMs trennt man zwischen den eigentlichen Zuständen (states) der Kette und den beobachtbaren Symbolen (observables / symbols). Dadurch entstehen neue Probleme, aber auch neue Möglichkeiten, insbesondere das “ Parsen ” einer Sequenz von Beobachtungen in eine Sequenz von Zuständen. Zusätzlich zu den von der einfachen Markovkette bekannten Übergangswahrscheinlichkeiten zwischen den Zuständen gibt es jetzt sogenannte Emissionswahrscheinlichkeiten für jeden Zustand, d.h. eine Verteilung auf den beobachtbaren Symbolen.

Wichtig in Zusammenhang mit HMMs ist vor allem zu sehen, dass sie wie geschaffen für die biologische Sequenzanalyse sind, insbesondere durch das Parsing (Problem 3.2). Sobald man ein Modell für bestimmte Signale oder Sequenzbereiche hat, kann man diese in einem oder in mehreren Zuständen des HMMs kodieren und dann Signale in neuen Sequenzen über das Parsing aufspüren.

Ein gutes Beispiel ist TMHMM, ein Programm zur Vorhersage von Transmembranhelices in Proteinsequenzen. Für eine Proteinsequenz gibt es drei Zustände: I (inside the cell), T (transmembrane helix), und O (outside the cell)¹. In jedem Zustand kann im Prinzip jede Aminosäure vorkommen, nur mit teilweise sehr unterschiedlichen Wahrscheinlichkeiten. Außerdem werden die Übergänge zwischen den Zuständen geschickt modelliert: Die Übergangswahrscheinlichkeit $T \rightarrow T$ ist z.B. so gewählt, dass die sich ergebende durchschnittliche Verweildauer in T der durchschnittlichen Länge einer Transmembranhelix entspricht. Man kann nun auf der Webseite <http://www.cbs.dtu.dk/services/TMHMM/> eine Proteinsequenz eingeben und erhält einen Parse in die unterschiedlichen Regionen.

Typische Problemstellungen bei HMMs sind:

Problem 3.1. Gegeben ein HMM, eine Zustandssequenz und eine Beobachtungssequenz, berechne die gemeinsame Wahrscheinlichkeit dieser Zustände und Beobachtungen.

Problem 3.2. Gegeben ein HMM und eine Sequenz aus beobachteten Symbolen, welches ist die plausibelste Zustandssequenz (Maximum-Likelihood)?

Problem 3.3. Gegeben mehrere mit Zustandssequenzen annotierte Beobachtungssequenzen, schätze die Modellparameter (die Übergangswahrscheinlichkeiten und die Emissionswahrscheinlichkeiten).

Problem 3.4. Gegeben mehrere unannotierte Beobachtungssequenzen, schätze die Modellparameter.

Wir gehen hier nicht näher darauf ein, aber für jedes der genannten Probleme gibt es in der Literatur bekannte Algorithmen. Problem 3.1 ist mehr oder weniger trivial. Für Problem 3.2 gibt es einen dynamic programming Algorithmus (Viterbi-Algorithmus). Problem 3.3 lässt sich leicht durch Zählen der Übergänge und zustandsspezifischen Emissionssymbole lösen. Problem 3.4 ist schwierig: In der Regel startet man mit irgendeiner Modellvorstellung und iteriert dann die Verfahren zur Lösung der Probleme 3.2 und 3.3, bis das Modell konvergiert. Diese Algorithmen und Anwendungen werden im Modul “Sequenzanalyse” genauer besprochen.

¹In Wirklichkeit ist die Modellierung um einiges komplexer!

4 Theorie: Markovprozesse

4.1 Grundlagen

Wir definieren Markovprozesse als Markovketten mit stetigem Zeitparameter. Diese Modellierung ist oft realistischer als über Markovketten, benötigt aber tiefergehende mathematische Theorie. Die Hauptanwendung, auf die wir später genauer eingehen, sind Aminosäuresubstitutionsprozesse (Proteinevolution) und DNA-Evolution. Dabei nehmen wir an, dass sich alle Aminosäuren voneinander unabhängig entwickeln und dass der Prozess (also die Ratenmatrix) an jeder Stelle gleich ist. Das heißt, wir betrachten nur eine einzelne Aminosäure im Laufe der Zeit.

Anmerkungen: (a) Die Unabhängigkeitsannahme ist hier mit Sicherheit das größte Problem, was Realismus angeht. Diese Annahme findet sich aber immer wieder, z.B. auch beim Sequenzalignment (BLAST, Smith-Waterman-Algorithmus), in dem jede Spalte des Alignments einzeln und unabhängig von den anderen bewertet wird. (b) Die Annahme, dass der Prozess überall gleich aussieht, lässt sich dadurch umgehen, dass wir verschiedene Prozesse jeweils für bestimmte strukturelle Abschnitte betrachten (α -Helix, β -Sheet, o.ä.).

Bei zeitstetigen Markovprozessen gibt es keine "kleinste Zeiteinheit", auf die man sich beziehen könnte. Daher beschreibt man den Prozess mit Hilfe der Analysis infinitesimal. Wenn überhaupt keine Zeit vergeht, können auch keine Übergänge stattfinden. Daher gilt stets $P^{(0)} = \text{Id}$, die Einheitsmatrix. Für sehr kleine Zeiträume h kann man annehmen, dass sich die Wahrscheinlichkeiten in P nahezu linear mit h ändern. Wir können also

$$P^{(h)} \approx \text{Id} + hQ \quad (h \text{ klein})$$

schreiben. Dabei ist Q die Matrix der Änderungsraten von $P^{(h)}$ zum Zeitpunkt $h = 0$. Genauer

$$Q = \lim_{h \searrow 0} \frac{P^{(h)} - \text{Id}}{h}.$$

Es stellt sich heraus, dass man mit Q bereits den gesamten Prozess beschrieben hat. Man nennt Q die *Ratenmatrix* des Prozesses.

Lemma 4.1. Die Ratenmatrix besitzt folgende Eigenschaften.

- $Q_{ij} \geq 0$ für $i \neq j$ und $Q_{ii} \leq 0$
- $\sum_j Q_{ij} = 0$ oder $Q_{ii} = -\sum_{j \neq i} Q_{ij}$
- Ist π stationäre Verteilung, gilt $\pi \cdot Q = 0$.

Übung 8. Begründen Sie die in Lemma 4.1 genannten Eigenschaften von Q .

Die Ratenmatrix Q definiert implizit eine Zeiteinheit. Ersetzt man Q durch $2Q$, ist eine Einheit danach doppelt so lang. Das heißt, Q und $2Q$ beschreiben im Grunde denselben Prozess, nur auf einer anderen Zeitskala. Wir müssen daher eine Zeiteinheit festlegen.

Definition 4.2 (Die Zeiteinheiten PEM und PAM).

PEM Ein PEM (percent of expected mutations) ist die Zeit, in der pro hundert Sites ein Übergang ("Mutation") erwartet wird.

Bei gegebener Ratenmatrix Q erwartet man pro Zeiteinheit

$$E := \sum_i \pi_i \sum_{j \neq i} Q_{ij} = -\sum_i \pi_i Q_{ii}$$

Übergänge pro Site. Man sagt, Q ist auf 1 PEM kalibriert, wenn $E = 1/100$ gilt.

PAM Ein PAM (percent of accepted mutations) ist die Zeit, nach der sich im Erwartungswert eine von hundert Sites verändert hat.

Bei gegebener Zeit-1 Übergangsmatrix P erwartet man

$$E' := \sum_i \pi_i \sum_{j \neq i} P_{ij}^{(1)} = 1 - \sum_i \pi_i P_{ii}$$

veränderte Sites nach einer Zeiteinheit. Man sagt, P (und die dazugehörige Ratenmatrix Q) ist auf 1 PAM kalibriert, wenn $E' = 1/100$ gilt.

Leider ist Q aus P oft nur numerisch berechenbar. Daher ist die Zeiteinheit 1 PAM unpraktischer als 1 PEM, wenn man mit stetigen Prozessen rechnet. Dagegen ist 1 PAM die natürliche Zeiteinheit, wenn man mit zeitdiskreten Ketten rechnet. Der Unterschied ist folgender: Wenn beispielsweise während einer Zeiteinheit A nach C und zurück nach A mutiert, sind dies zwei Mutationen (im Sinne von PEM), aber nach Ablauf der Zeiteinheit wird keine Mutation beobachtet (im Sinne von PAM liegen also 0 Mutationen vor). Es folgt, dass 1 PEM eine etwas kürzere Zeiteinheit als 1 PAM ist. Die Bezeichnung PEM ist übrigens nicht sehr üblich; leider wird oft beides mit PAM bezeichnet. Die Bezeichnung PEM wurde hier gewählt, um Verwechslungen zu vermeiden.

Es muss betont werden, dass man keinen festen Bezug zwischen PEM/PAM und Realzeit herstellen kann. Die definierten Einheiten messen eine "Menge an Evolution", keine physikalische Zeit. Den Umrechnungsfaktor könnte man als *Evolutionsgeschwindigkeit* bezeichnen: Viele PEM in wenig physikalischer Zeit bedeuten eine hohe Evolutionsgeschwindigkeit.

Im Rahmen der molekularen Evolution fordern wir insgesamt folgende Eigenschaften von einem Markovprozess.

Definition 4.3 (π -EMP). Ein zeitstetiger zeithomogener Markovprozess $(X_t)_{t \geq 0}$ auf dem (endlichen) Zustandsraum \mathcal{X} heißt *evolutionärer Markovprozess* mit stationärer Verteilung π (π -EMP), wenn folgendes gilt:

- Die stationäre Verteilung des Prozesses ist π , und π ist die einzige stationäre Verteilung.
- Die Startverteilung ist ebenfalls π ; damit ist $X(t)$ für jedes t nach π verteilt.
- Die Ratenmatrix Q des Prozesses ist auf 1 PEM kalibriert.
- Der Prozess ist zeitreversibel.

Nur der Begriff *zeitreversibel* wurde noch nicht erklärt. Zeitreversibilität bedeutet, dass der Prozess derselbe ist, wenn man ihn vorwärts oder rückwärts in der Zeit betrachtet. Das heißt, es muss für alle $s, t > 0$ und alle $i, j \in \mathcal{X}$ eine der folgenden äquivalenten Bedingungen gelten (sogenannte *detailed-balance-Gleichungen*).

$$\begin{aligned} \mathbb{P}(X(s) = i, X(s+t) = j) &= \mathbb{P}(X(s) = j, X(s+t) = i), \\ \pi_i P_{ij} &= \pi_j P_{ji}, \\ \pi_i Q_{ij} &= \pi_j Q_{ji}. \end{aligned}$$

Der Grund für diese Forderung ist folgender. Wir betrachten zwei heutige Sequenzen S_1 und S_2 . Diese haben sich nach unserem Modell vor t Zeiteinheiten aus einem gemeinsamen Vorfahren entwickelt. Dieser Vorfahr ist aber unbekannt. Um die evolutionären Ereignisse zu beschreiben, die S_1 und S_2 trennen, muss man den Prozess von S_1 aus zeitlich rückwärts bis zum gemeinsamen Vorfahren und von dort aus vorwärts bis S_2 betrachten. Die Annahme der Zeitreversibilität erlaubt, dies als einen einzigen Vorwärtspfad der Länge $2t$ zu betrachten (siehe Abbildung 1).

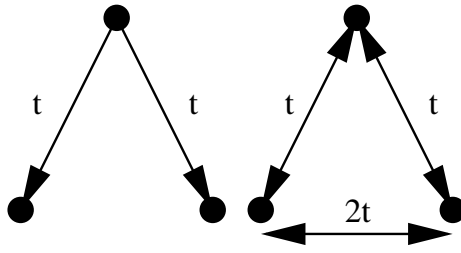


Abbildung 1: Links: Keine Zeitreversibilität. Rechts: Zeitreversibilität.

4.2 Typische Fragestellungen

Problem 4.4. Gegeben Q , berechne $P^{(t)}$, insbesondere $P = P^{(1)}$.

Man weiß $P^{(0)} = \text{Id}$, $\frac{d}{dt}P^{(t)}|_{t=0} = Q$, und $P^{(s+t)} = P^{(s)} \cdot P^{(t)}$. Um P approximativ zu berechnen, teilen wir das Zeitintervall $[0, 1]$ in m Teilstücke der Länge $h = 1/m$. Ist m groß genug, können wir $P^{(h)} \approx \text{Id} + hQ$ annehmen, also

$$P = (P^{(h)})^m \approx \left(\text{Id} + \frac{Q}{m} \right)^m \rightarrow \exp(Q) \quad (m \rightarrow \infty).$$

Die Matrix $\exp(Q)$ heißt *Matrixexponential* von Q . Man kann es formal wie bei reellen Zahlen über die Exponentialreihe berechnen, die sehr gute Konvergenzeigenschaften besitzt:

$$\exp(Q) = \sum_{k=0}^{\infty} \frac{Q^k}{k!}.$$

Problem 4.5. Gegeben $P = P^{(1)}$, berechne die Ratenmatrix Q .

Leider gibt es nicht zu jeder Übergangsmatrix P eine Ratenmatrix Q . Anders gesagt, nicht jede (zeitdiskrete) Markovkette lässt sich in einen (zeitstetigen) Markovprozess einbetten. Eine einfache Charakterisierung der Markovketten, für die das möglich ist, ist bis heute nicht bekannt. Man hat nur die Möglichkeit, numerisch den Matrixlogarithmus Q von P zu berechnen. Dabei kann es zwei Probleme geben. Erstens, es stellt sich heraus, dass dieser nicht existiert, d.h., es gibt keine Matrix Q mit $P = \exp(Q)$. Zweitens, Q existiert, hat aber nicht die Eigenschaften einer Ratenmatrix, sondern z.B. negative Raten.

Im folgenden gehen wir davon aus, dass wir nur Markovprozesse mit strikt positiven Raten Q_{ij} für $i \neq j$ betrachten.

Problem 4.6 (Parameterschätzung). Wie schätzt man einen Markovprozess aus gegebenen Daten? Es ist sinnvoll, hier eine allgemeinere Form der Daten zuzulassen als bei Markovketten. Wir gehen davon aus, dass wir eine Reihe von unabhängigen Übergängen $i \xrightarrow{t} j$ beobachten, d.h., i geht über in j in der Zeit t . Dabei ist unbekannt, ob es bei diesem Übergang weitere unbeobachtete Zwischenschritte gibt.

Das Schätzverfahren kann sehr aufwändig werden. Wir beschreiben hier einen Maximum-Likelihood Ansatz. Für die praktische Durchführung wird man auf numerische Optimierungsverfahren zurückgreifen müssen. Wir gehen davon aus, dass die i - und t -Werte in $i \xrightarrow{t} j$ gegeben sind und dass die j -Werte durch den zu schätzenden Prozess generiert wurden. Wenn insgesamt n unabhängige Beobachtungen $i_k \xrightarrow{t_k} j_k$ ($k = 1, \dots, n$) vorliegen, beträgt die Gesamtwahrscheinlichkeit aller Beobachtungen

$$L(Q) = \prod_{k=1}^n [P^{(t_k)}]_{i_k, j_k} = \prod_{k=1}^n [\exp(t_k \cdot Q)]_{i_k, j_k}.$$

Die Einträge von Q sollen in Übereinstimmung mit den Eigenschaften von Lemma 4.1 so gewählt werden, dass L maximiert wird. Die Abhängigkeit der Einträge von $\exp(tQ)$ von denen von Q ist jedoch sehr kompliziert.

Problem 4.7 (Zeitschätzung). Die Modellparameter seien bekannt. Gegeben seien wieder n Übergänge $i_k \xrightarrow{t} j_k$, sämtlich innerhalb derselben unbekanntem Zeit t , die zu schätzen ist.

Man verwendet wieder den Maximum-Likelihood Ansatz. Die Wahrscheinlichkeit, alle Übergänge in der Zeit t zu beobachten, ist

$$L(t) = \prod_{k=1}^n [\exp(tQ)]_{i_k, j_k}.$$

Da Q in diesem Fall bekannt ist, kann man einfach $\exp(tQ)$ für mehrere Werte von t berechnen, den obigen Ausdruck für $L(t)$ auswerten, und somit ein t finden, das ihn maximiert. Im Falle einiger spezieller Ratenmatrizen können wir den Zeitschätzer mit expliziten Formeln aufschreiben; darauf kommen wir später zurück.

5 Anwendung: Scorematrizen

Um Informationen über ein bisher unbekanntes Protein zu erhalten, wird oft eine Datenbanksuche verwendet. Man hofft, in der Datenbank ähnliche und annotierte Sequenzen zu finden, so dass man die Informationen, die über die Datenbanksequenzen (Subjects) vorliegen, auf die Anfragesequenz (Query) übertragen kann. Um eine Datenbanksuche durchführen zu können, benötigt man also ein Ähnlichkeitsmaß für Sequenzen. Zum Beispiel berechnet der Smith-Waterman Algorithmus den Score des besten lokalen Sequenzalignments zwischen Query und allen Subjects. Der Grundbaustein für so ein Maß ist dabei ein Ähnlichkeitsmaß auf den individuellen Symbolen, in diesem Fall auf den Aminosäuren. Diese werden in einer sogenannten Scorematrix gespeichert, d.h., in einer Matrix (S_{ij}) , die jedem Aminosäurepaar (i, j) einen Ähnlichkeitswert zuordnet. Wir beschäftigen uns jetzt mit der Herleitung solcher Ähnlichkeitswerte.

Es bieten sich sofort einige ad-hoc-Methoden zu Berechnung dieser Werte an.

- Man setzt $S_{ii} = 1$ und $S_{ij} = -1$ für $i \neq j$. So lassen sich aber nur lange Strecken von identisch konservierten Aminosäuren finden.
- Man setzt $S_{ij} = 1 - x$, wobei x die minimale Anzahl von Mutationen in einem für i codierenden DNA-Triplett ist, um das Triplett in ein für j codierendes zu verwandeln. In diesem Falle gilt also auch immer $S_{ii} = 1$, und S_{ij} liegt zwischen -2 und 0 für $i \neq j$.
- Man beauftragt einen Experten, Ähnlichkeitswerte anhand der physikalischen und chemischen Eigenschaften der Aminosäuren festzulegen. Das ist sicher im Prinzip eine gute Methode, doch es wird schwer, die Verhältnismäßigkeit aller 400 Werte zu gewährleisten.

In der Tat hat sich eine andere Methodik zur Berechnung von Ähnlichkeitswerten durchgesetzt, die auf Markovprozessen beruht. Die bekanntesten Scorematrizen sind die PAM- und die BLOSUM-Familie, wobei die PAM-Matrizen in der Tat auf einem Markovmodell basieren, während die BLOSUM-Matrizen anders berechnet wurden, sich aber auch in diesen Rahmen einordnen lassen.

Die Grundidee ist folgende: Ähnliche Aminosäuren werden häufiger durcheinander ersetzt als nichtähnliche. Diese Idee kann man umkehren und Ähnlichkeit über die Ersetzungswahrscheinlichkeiten definieren. Wenn man aber nun das Paar (i, j) häufig (selten) aligniert beobachtet, kann das

zwei Gründe haben: Erstens, i und j sind ähnlich (unähnlich). Zweitens, i und j sind selbst häufige (seltene) Aminosäuren. Daher nimmt man als Maß den Quotienten

$$\frac{\mathbb{P}(X(0) = i, X(t) = j)}{\mathbb{P}(X(0) = i) \cdot \mathbb{P}(X(t) = j)} = \frac{\pi_i P_{ij}^{(t)}}{\pi_i \pi_j} = \frac{\pi_j P_{ji}^{(t)}}{\pi_i \pi_j}.$$

Der Zähler gibt die Wahrscheinlichkeit an, das Paar (i, j) in einem Sequenzpaar mit Zeitabstand t zu beobachten, die nach dem Modell evolviert sind, und der Nenner gibt die Wahrscheinlichkeit an, das Paar in unabhängigen Sequenzen zu beobachten. Die Gleichheit $\pi_i P_{ij}^{(t)} = \pi_j P_{ji}^{(t)}$ gilt wegen der Zeitreversibilität.

Der Score ist nun definiert als Logarithmus dieses Verhältnisses.

$$S_{ij}^{(t)} := \log \frac{\pi_i P_{ij}^{(t)}}{\pi_i \pi_j},$$

bzw. als ein ganzzahliges gerundetes Vielfaches davon.

Der Score ist positiv, wenn das Paar (i, j) in evolutionär verwandten Sequenzen häufiger auftaucht als in unabhängigen und negativ, wenn es dort seltener auftaucht.

Zur Berechnung einer Scorematrix muss man eine passende Ratenmatrix Q , die den Evolutionsprozess beschreibt, aus bekannten Alignments schätzen. Im Prinzip wurden die auftretenden Probleme oben diskutiert; allerdings gibt es eine weitere Schwierigkeit. Der zeitliche Abstand des gemeinsamen Vorfahren zweier alignierter Sequenzen ist nicht bekannt. Haben sich die Sequenzen vor t Zeiteinheiten auseinander entwickelt, haben sie im zeitreversiblen Modell den Abstand $2t$. Dieser muss für jedes Sequenzpaar gleichzeitig mit den Modellparametern Q bestimmt werden. Das heißt, wir haben gleichzeitig ein Parameter- und ein Zeitschätzproblem. Hier greift man oft auf eine iterative Methode zurück, die mit sinnvollen Startwerten beginnt und dann zwischen beiden Schätzungen alterniert, bis Konvergenz erreicht wird.

Übung 9 (Scorematrix-Projekt). Dies ist ein langfristiges Projekt und eignet sich, wenn Sie z.B. eine bestimmte Protein-Domäne analysieren wollen. Sammeln Sie Alignments von Instanzen dieser Domäne. Schätzen Sie daraus einen Markovprozess und eine domänenspezifische Scorematrix. Wenn Sie jetzt mit einer Query, die diese Domäne enthält, Datenbanksuche betreiben, verwenden Sie an den entsprechenden Stellen in der Query Ihre Scorematrix anstelle der BLOSUM o.ä. Inwieweit verbessern sich dadurch die Suchergebnisse in der Praxis?

6 Theorie: Wichtige evolutionäre DNA-Modelle

Ein EMP ist durch seine zeitreversible kalibrierte Ratenmatrix Q vollständig beschrieben, denn aus Q lässt sich die stationäre Verteilung π , die auch Startverteilung ist, durch Lösen der Gleichungen $\pi \cdot Q = 0$, $\sum_i \pi_i = 1$ gewinnen. Im folgenden betrachten wir zunächst einige wichtige Modelle für EMPs auf Nukleotiden.

6.1 Jukes-Cantor (JC)

Das einfachste Modell, das sogenannte *Jukes-Cantor* Modell, für einen EMP erhält man durch der Annahme, dass jede Art von Mutation mit der gleichen Rate α auftritt. Die Ratenmatrix hat dann die Gestalt

$$Q_\alpha = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}.$$

Die stationäre Verteilung ist die Gleichverteilung $\pi = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$; die Bedingung der Zeitreversibilität ist offensichtlich erfüllt. Jetzt muss die Matrix noch kalibriert werden. Das führt auf $\frac{3 \cdot 4}{4} \cdot \alpha = \frac{1}{100}$ oder $\alpha = \frac{1}{300}$. Es ist also für beliebige Zeiten $t > 0$:

$$tQ = \begin{pmatrix} \frac{-t}{100} & \frac{t}{300} & \frac{t}{300} & \frac{t}{300} \\ \frac{t}{300} & \frac{-t}{100} & \frac{t}{300} & \frac{t}{300} \\ \frac{t}{300} & \frac{t}{300} & \frac{-t}{100} & \frac{t}{300} \\ \frac{t}{300} & \frac{t}{300} & \frac{t}{300} & \frac{-t}{100} \end{pmatrix}.$$

Aufgrund der einfachen Struktur von Q_α kann man $\exp(Q_\alpha)$ und daher auch $\exp(tQ)$ (durch Diagonalisierung und Eigenwertanalyse) explizit berechnen. Man erhält für die Zeit- t -Übergangsmatrix das Resultat

$$P^{(t)} = \begin{pmatrix} 1 - 3a_t & a_t & a_t & a_t \\ a_t & 1 - 3a_t & a_t & a_t \\ a_t & a_t & 1 - 3a_t & a_t \\ a_t & a_t & a_t & 1 - 3a_t \end{pmatrix},$$

wobei

$$a_t = \frac{1 - \exp(-4t/300)}{4}. \quad (1)$$

6.2 Kimura 2-Parameter (K2P)

In Wirklichkeit treten nicht alle Arten von Mutationen gleich häufig auf. Die Nukleotide teilen sich auf in Purine (A,G) und Pyrimidine (C,T). Eine Mutation innerhalb einer Gruppe ($A \leftrightarrow G$; $C \leftrightarrow T$) bezeichnet man als eine *Transition*, eine Mutation zwischen den Gruppen als *Transversion*. Transversionen treten seltener auf als Transitionen. Man legt daher verschiedene Raten für Transitionen (α) und Transversionen ($\beta < \alpha$) zugrunde. Ist die Nukleotidreihenfolge (abweichend von der alphabetischen Ordnung) AGCT, schreibt sich die Ratenmatrix als

$$Q_{\alpha,\beta} = \begin{pmatrix} -(\alpha + 2\beta) & \alpha & \beta & \beta \\ \alpha & -(\alpha + 2\beta) & \beta & \beta \\ \beta & \beta & -(\alpha + 2\beta) & \alpha \\ \beta & \beta & \alpha & -(\alpha + 2\beta) \end{pmatrix}.$$

Dabei muss die Kalibrierungsbedingung $\alpha + 2\beta = \frac{1}{100}$ gelten. Schreibt man $\beta = \tau\alpha$ mit $\tau < 1$ (z.B. mit $\tau = 1/2$ ist eine Transversion nur halb so wahrscheinlich wie eine Transition), so folgt

$$\alpha = \frac{1}{100(1 + 2\tau)}, \quad \beta = \frac{\tau}{100(1 + 2\tau)}.$$

Man kann nachrechnen, dass für $P^{(t)} = \exp(tQ_{\alpha,\beta})$ gilt:

$$P^{(t)} = \begin{pmatrix} 1 - (a_t + 2b_t) & a_t & b_t & b_t \\ a_t & 1 - (a_t + 2b_t) & b_t & b_t \\ b_t & b_t & 1 - (a_t + 2b_t) & a_t \\ b_t & b_t & a_t & 1 - (a_t + 2b_t) \end{pmatrix},$$

wobei

$$a_t := (2E_t - e_t)/4, \quad E_t := 1 - \exp(-2t(\alpha + \beta)), \quad (2)$$

$$b_t := e_t/4, \quad e_t := 1 - \exp(-4t\beta). \quad (3)$$

6.3 Weitere Modelle

Man kann die schon beschriebenen Modelle in zwei Richtungen verallgemeinern.

1. Man kann die Gleichverteilung als stationäre Verteilung beibehalten, aber für jeden Mutationstyp eine andere Rate zulassen. Wegen der Detailed-Balance-Gleichung ist dann immer noch $Q_{ij} = Q_{ji}$; man betrachtet also allgemeine symmetrische Ratenmatrizen (6 Parameter, abhängig durch die Kalibrierungsbedingung, also 5 freie Parameter). Man spricht dabei vom symmetrischen Modell (SYM).
2. Man kann auch vorschreiben, dass bestimmte (relative) Raten übereinstimmen, aber die stationäre Verteilung freigeben. Dies definiert das Felsenstein-Modell (analog zu Jukes-Cantor) und das Hasegawa-Kishino-Yano-Modell (analog zu K2P).
3. Im allgemeinen zeitreversiblen Modell (GTR; general time reversible) sind stationäre Verteilung π und relative Raten beliebig. Man hat 3 freie Parameter für π und wie oben 5 freie Parameter für die relativen Raten, also insgesamt 8 freie Parameter.

Auf Proteinsequenzen (20 Aminosäuren) verwendet man entweder nur ein (viel zu vereinfachendes) Jukes-Cantor-Modell (bei der Zeiteinheit 1 PEM ist die gemeinsame Rate dann $\alpha = 1/1900$), oder besser gleich ein GTR-Modell. Dieses lässt sich wegen der großen Parameterzahl aber nur noch numerisch am Computer behandeln.

7 Theorie: Zeitschätzung mit EMPs

Sei ein EMP in Form einer Ratenmatrix Q gegeben. Seien $A = (A_1, \dots, A_n)$, $B = (B_1, \dots, B_n)$ zwei Sequenzen gleicher Länge n , die sich nach diesem Prozess entwickelt haben. Gemäß dem Modell betrachten wir nur Substitutionen, keine Insertionen und Deletionen. Unser Ziel ist, die Zeit t (in PEM), die vermutlich zwischen A und B liegt, zu schätzen. Achtung: Die Zeit t hier entspricht $2t$ in Abbildung 1. Die geschätzten Zeiten lassen sich dann als Distanzen für distanzbasierte Baumrekonstruktionsmethoden verwenden.

Ein erster naiver Ansatz besteht darin, einfach die Anzahl d der Unterschiede zwischen A und B zu zählen. Da im Mittel pro 1 PEM ein hundertstel Mutationsereignis stattfindet, erhält man die Zeitschätzung

$$t \approx \frac{100d}{n} \text{ PEM.}$$

Diese Schätzung ist relativ grob, kann jedoch bei kurzen Divergenzzeiten ausreichend sein. Der Ansatz berücksichtigt aber nicht, dass im Laufe längerer Zeiträume manche Mutationen, die stattgefunden haben, durch Rücksubstitution wieder verschwinden. Das heißt, man beobachtet weniger Mutationen als stattgefunden haben. Daher wird t systematisch unterschätzt.

Bessere Schätzer leiten wir aus einem EMP-Modell mit Hilfe des Maximum-Likelihood-Prinzips ab. Die Gesamtwahrscheinlichkeit des Auftretens von A und B mit t Zeiteinheiten Abstand beträgt

$$p(A, B; t) := \prod_{k=1}^n \mathbb{P}(X(0) = A_k, X(t) = B_k) = \prod_{k=1}^n \pi_{A_k} P_{A_k, B_k}^{(t)}.$$

Man beachte, dass die Zeitreversibilität erlaubt, A und B zu vertauschen; die Wahrscheinlichkeit ändert sich dadurch nicht. Wir fassen $p(A, B; t)$ als Likelihood-Funktion der zu schätzenden Zeit t auf und suchen das t , mit dem die Wahrscheinlichkeit der Beobachtungen maximal wird (Maximum-Likelihood-Schätzer). Dort wo $t \mapsto p(A, B; t)$ maximal wird, wird auch der Logarithmus maximal. Die

Terme π_{A_k} hängen gar nicht von t ab. Gesucht ist also das Maximum der Funktion

$$\mathcal{L}(t) := \sum_{k=1}^n \log \left(P_{A_k, B_k}^{(t)} \right).$$

Dies kann im allgemeinen Fall nur numerisch bestimmt werden: Man wertet die Funktion für viele Werte von t am Computer aus und sucht dann das Maximum.

Für einfache Modelle lässt sich jedoch der Ausdruck $\mathcal{L}(t)$ direkt angeben. Wir betrachten das Jukes-Cantor-Modell und das Kimura-Modell.

7.1 Die Jukes-Cantor-Korrektur

Im JC-Modell gilt nach Gleichung (1): $P_{ij}^{(t)} = a_t$ für $i \neq j$ und $P_{ii}^{(t)} = 1 - 3a_t$. Sei d die Zahl der Unterschiede in den Sequenzen A und B und n ihre jeweilige Länge. Es folgt

$$\mathcal{L}(t) = d \log a_t + (n - d) \log(1 - 3a_t).$$

Diese Funktion wird maximal für $a_t = d/(3n)$, also nach Gleichung (1) für

$$t^*(d) = -\frac{300}{4} \log \left(1 - \frac{4d}{3n} \right) \text{ PEM.} \quad (4)$$

Ist $d \ll n$, so gilt $\log(1 - (4d)/(3n)) \approx -(4d)/(3n)$ und daher in etwa die naive lineare Schätzung $t^*(d) \approx (100d)/n$. Für größere Werte von d wächst t^* aber schneller als der lineare Schätzer (siehe Abbildung 2). Man spricht daher bei Gleichung (4) auch von der Jukes-Cantor-Korrektur (der linearen Schätzung).

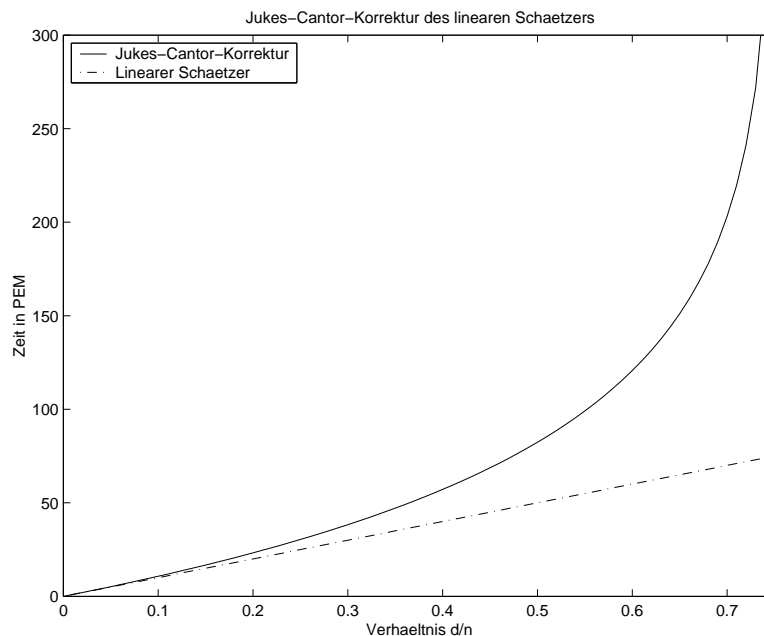


Abbildung 2: Linearer (naiver) Zeitschätzer im Vergleich zur Jukes-Cantor-Korrektur.

Achtung: Ist $d/n \geq 3/4$, so ist $t^*(d)$ nicht definiert (negatives Argument im Logarithmus). Das macht auch Sinn: Selbst wenn B und A zufällig ausgewürfelt sind (also sicher keine evolutionäre

Verwandtschaft aufweisen), stimmen sie bereits nach dem Gesetz der großen Zahl im Erwartungswert in einem Viertel ihrer Positionen überein. Beobachtet man also $d/n \geq 3/4$, so ist $t^* = \infty$ (nicht verwandt) die sinnvollste Schätzung. Aus Sequenzen, die offensichtlich gar nichts miteinander zu tun haben, lässt sich ohnehin kein vernünftiger Baum schätzen.

7.2 Die Kimura-Formel

Im Kimura-Modell werden Transitionen und Transversionen getrennt betrachtet. Ihre jeweiligen Raten seien α und $\beta = \tau\alpha$ mit $\tau < 1$. Angenommen, man beobachtet s Transitionen und v Transversionen (und damit $n - (s + v)$ Identitäten). Es folgt

$$\mathcal{L}(t) = s \log a_t + v \log b_t + (n - s - v) \log(1 - a_t - 2b_t),$$

wobei a_t und b_t durch die Gleichungen (2) und (3) gegeben sind. Fassen wir a_t und b_t als unabhängige Parameter auf, so wird \mathcal{L} maximal, wenn (was intuitiv einleuchtet)

$$a_t = \frac{s}{n}, \quad b_t = \frac{v}{2n}.$$

Man erhält

$$\begin{aligned} t\beta &= -\frac{1}{4} \log \left(1 - \frac{2v}{n} \right) && \approx \frac{v}{2n} = b_t, \\ t\alpha &= -\frac{2}{4} \log \left(1 - \frac{2s+v}{n} \right) + \frac{1}{4} \log \left(1 - \frac{2v}{n} \right) && \approx \frac{s}{n} = a_t. \end{aligned}$$

Mit der Kalibrierungsbedingung $\alpha + 2\beta = 1/100$ folgt

$$t = \frac{t(\alpha + 2\beta)}{1/100} = -\frac{100}{4} \left[2 \log \left(1 - \frac{2s+v}{n} \right) + \log \left(1 - \frac{2v}{n} \right) \right] \text{ PEM.}$$

Approximativ sind das wiederum

$$t \approx \frac{100(s+v)}{n} \text{ PEM} = \frac{100d}{n} \text{ PEM,}$$

wobei d die Anzahl der Unterschiede der beiden Sequenzen ist.

8 Anwendung: ML-Phylogenie-Rekonstruktion

8.1 Übersicht

Wir haben gesehen, dass man mit Hilfe eines EMP-Modells die evolutionären Distanzen zwischen gegebenen Sequenzen schätzen kann. Eine so gewonnene Distanzmatrix kann als Eingabe für distanzbasierten Baumrekonstruktionsmethoden dienen (z.B. für den Neighbor-Joining-Algorithmus). Man kann aber auch über einen EMP einen Baum direkt mit Hilfe der Maximum-Likelihood-Methode schätzen. Dabei geht man in mehreren Schritten vor.

1. Wähle eine Ratenmatrix Q . Es wird angenommen, dass der Prozess an jeder Stelle innerhalb des Baumes derselbe ist und durch Q beschrieben wird.
2. Wähle eine Baumtopologie \mathcal{T} . Zu einer Baumtopologie gehört in diesem Fall die Baumstruktur und die Zuordnung der einzelnen Sequenzen zu den Blättern; siehe auch Abbildung 3.

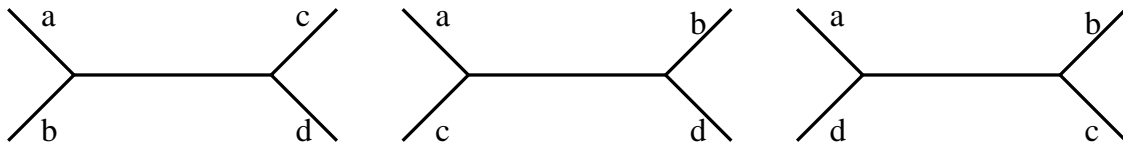


Abbildung 3: Die drei möglichen Topologien auf ungewurzelten Bäumen mit 4 Sequenzen a, b, c, d (Quartette). Von links nach rechts: $ab||cd$, $ac||bd$, $ad||bc$.

3. Für die gewählte Baumtopologie \mathcal{T} , bestimme mit Maximum-Likelihood die Längen aller Kanten (also die Distanzen zwischen benachbarten Knoten).

Im dritten Schritt erhält man für eine gegebene Baumtopologie die beste Zuordnung an Kantenlängen, die unter Modell Q möglich ist. Um den insgesamt plausibelsten Baum unter Q zu finden, muss man diesen Schritt für *alle Topologien* durchführen. Wegen der exponentiell großen Anzahl an Topologien muss man in der Regel auf Heuristiken (z.B. Quartet-Puzzling, siehe unten) zurückgreifen. Und wenn man den besten Baum unter Q gefunden hat, kann man wiederum versuchen, die Parameter von Q zu verändern, um ein neues Modell zu bekommen, in dem noch bessere Bäume möglich sind (in der Regel beschränkt man sich aber auf ein festes vorgegebenes Q).

Eine Heuristik zum Bestimmen einer guten Topologie ist das *Quartet-Puzzling*. Bei N Sequenzen insgesamt gibt es insgesamt $\binom{N}{4}$ Möglichkeiten, 4 Sequenzen (Quartette) auszuwählen. Bei 4 Sequenzen a, b, c, d gibt es drei mögliche Topologien: $ab||cd$, $ac||bd$, und $ad||bc$ (siehe Abbildung 3). Man führt zunächst die Maximum-Likelihood-Schätzung für alle $3\binom{N}{4}$ Quartett-Topologien durch und versucht dann mit geeigneten Methoden, den Gesamtbaum aus den besten der jeweils drei Quartetttopologien konsistent zusammenzupuzzeln.

Die Maximum-Likelihood-Methode ist relativ langsam. Wenn die Modellannahmen stimmen, ist sie konsistent (sie rekonstruiert bei unendlich vielen Daten den richtigen Baum), und erzielt schon bei relativ kleinen Datenmengen annehmbare Ergebnisse. Allerdings ist die Methode relativ hilflos, wenn die Annahmen (unabhängige Sites, Ratenmatrix konstant im gesamten Baum) verletzt werden.

Zurück zum Algorithmus. Insgesamt ist ein Baum im ML-Modell durch drei Arten von Parametern gegeben: den EMP Q , die Topologie \mathcal{T} , und die Kantenlängen L . Im folgenden gehen wir auf den dritten Schritt des ML-Verfahrens genauer ein. Wie bestimmt man den Vektor L der Kantenlängen bei gegebenem (Q, \mathcal{T}) , so dass die Wahrscheinlichkeit der beobachteten Daten X unter allen (Q, \mathcal{T}, L) für freies L maximal wird? Sind v, w zwei benachbarte Knoten des Baumes (innere Knoten oder Blätter), so sei $L(v \leftrightarrow w)$ die Länge der sie verbindenden Kante.

8.2 Optimierung der Log-Likelihood eines Baumes

Ist ein Baum (Q, \mathcal{T}, L) gegeben, kann man die Wahrscheinlichkeit der Daten (m Sequenzen S_1, \dots, S_m der Länge n , also n Alignmentsspalten X_1, \dots, X_n jeweils der Länge m) berechnen. Dazu nimmt man an, dass die Spalten (*sites*) unabhängig sind. Im folgenden genügt es also, jeweils nur eine Site zu betrachten. Die Gesamtwahrscheinlichkeit ergibt sich dann als Produkt über alle Sites.

Die Likelihood eines Baumes (Q, \mathcal{T}, L) für gegebene Daten X ist die Wahrscheinlichkeit der Daten unter dem Baum (Q, \mathcal{T}, L) . Sind Q und \mathcal{T} fest, betrachten wir die Likelihood-Funktion nur als Funktion von L . Aus praktischen Gründen (Additivität) betrachtet man den Logarithmus. Zu maximieren ist also die Funktion

$$L \mapsto \sum_{i=1}^n \log \mathbb{P}(X_i \mid (Q, \mathcal{T}, L)).$$

Die Maximierung muss in der Regel numerisch durchgeführt werden. Wir wollen nur beschreiben, wie man $\mathbb{P}(X_i | (Q, \mathcal{T}, L))$ effizient ausrechnen kann (als Unterroutine für das numerische Optimierungsverfahren). Wir beschreiben das Verfahren für gewurzelte Bäume. Bei ungewurzelten Bäumen mache man einen beliebigen inneren Knoten zur Wurzel (dieser hat dann drei Kinder statt zwei). Wegen der angenommenen Zeitreversibilität hängt das Ergebnis nicht vom ausgewählten Knoten ab.

Sei v ein innerer Knoten und $C(v)$ die Menge der Kinder von v (Blätter oder wieder innere Knoten). Für $w \in C(v)$ sei $L(v \rightarrow w)$ die durch L gegebene Länge der Kante $v \rightarrow w$. Mit $\mathcal{P}_Y(v)$ bezeichnen wir die Wahrscheinlichkeit des Teilbaumes unter v unter der Bedingung, dass das Symbol in v ein Y ist. Ist b_j das Blatt, an dem Sequenz j steht, so ist $\mathcal{P}_{X_{i,j}}(b_j) = 1$ und $\mathcal{P}_Y(b_j) = 0$ für $Y \neq X_{i,j}$. Ist v ein innerer Knoten, so folgt

$$\mathcal{P}_Y(v) = \prod_{w \in C(v)} \sum_Z P_{Y,Z}^{(L(v \rightarrow w))} \mathcal{P}_Z(w).$$

Auf diese Weise kann man von den Blättern aus bottom-up $\mathcal{P}_Y(v)$ für jeden Knoten v und jedes Symbol Y berechnen. Schließlich kennt man $\mathcal{P}_Y(r)$ für die Wurzel r des Baumes und alle Symbole Y . Die Symbolverteilung an der Wurzel ist (wie überall) die stationäre Verteilung π . Mit der Formel der totalen Wahrscheinlichkeit folgt

$$\mathbb{P}(X_i | (Q, \mathcal{T}, L)) = \sum_Y \pi_Y \mathcal{P}_Y(r).$$

Diese Prozedur muss man für jede Spalte X_i des Alignments durchführen und die resultierenden $\mathbb{P}(X_i | (Q, \mathcal{T}, L))$ multiplizieren, bzw. ihre Logarithmen addieren.

Es ist offensichtlich, dass die Abhängigkeit dieses Ausdrucks von L relativ kompliziert ist, und dass numerische Methoden zur Maximierung erforderlich sind. Man benutzt ein iteratives Verfahren: Man hält alle Längen bis auf eine fest und optimiert die ausgewählte Länge. Dies wiederholt man mehrfach für alle Kanten nacheinander, bis Konvergenz eintritt.

9 Theorie: Bootstrapping von Phylogenie-Algorithmen

Ein phylogenetischer Algorithmus antwortet nach der Eingabe mehrerer alignierter Sequenzen schließlich mit einem Baum. Dieser Baum ist eine Schätzung des “wahren” Baumes, aber man kann natürlich nicht garantieren, dass der wahre Baum gefunden wird. Es gibt viele Gründe, die dagegen sprechen.

- Die Anzahl der beobachteten Daten (Alignment-Spalten) ist endlich.
- Die (expliziten oder impliziten) Modellannahmen, auf denen der Algorithmus beruht, treffen nicht oder nur zum Teil zu.
- Der wahre Baum erfüllt kein greifbares Optimalitätskriterium.²

Um es klar zu sagen: Gegen all dies ist man machtlos. Ebenso lassen sich folgende Fragen *nicht* beantworten: Mit welcher Wahrscheinlichkeit rekonstruiert ein Algorithmus den wahren Baum? Mit welcher Wahrscheinlichkeit liegt ein Split S im wahren Baum? Es gibt zwei Ausnahmefälle: (1) Man hat die Welt erschaffen. (2) Man arbeitet auf nach einem bekannten Modell erzeugten *synthetischen Daten*. Da (1) nicht zutrifft, werden in der wissenschaftlichen Literatur Algorithmen nach dem zweiten Kriterium beurteilt; allerdings immer unter der Annahme, dass man das wahre Modell kennt. Das ist bei realen (“natürlichen”) Daten nicht der Fall.

Der Grund für die obigen Bemerkungen ist, dass leider Bootstrap-Werte bei phylogenetischen Bäumen manchmal so missverstanden werden, als beantworteten sie die genannten Fragen. In diesem

²Auch Charlie Chaplin ist einmal in einem Charlie-Chaplin-Contest nur Zweiter geworden.

Abschnitt geht es jedoch um ein anderes Problem, und zwar die Störanfälligkeit des verwendeten Algorithmus gegenüber kleinen Änderungen in den Eingabedaten. Ein Bootstrap-Wert liefert eine Aussage über die dem Algorithmus inhärente Störanfälligkeit auf den gegebenen Daten. Ist ein Bootstrap-Wert schlecht (klein), sollte man dem Ergebnis wenig Vertrauen schenken. Ist ein Bootstrap-Wert gut, sagt er wenig aus über das Verhältnis zwischen berechnetem Ergebnis und “Wahrheit”.

Das wird noch einmal besonders an folgendem Beispiel klar. Angenommen, ein Algorithmus ignoriert die Eingabesequenzen bis auf ihre Anzahl m , und gibt stets einen festen Baum mit m Blättern aus. Dieser Algorithmus ist gegenüber Änderungen der Eingabedaten in keiner Weise störanfällig und alle Bootstrap-Werte werden bei 100% liegen. Es ist klar, dass der ausgegebene Baum fast nie dem wahren Baum entspricht.

9.1 Der Bootstrap

Sei $T = T(X)$ ein Schätzer, der aus Eingabedaten X ein Objekt T schätzt. Zwei Beispiele:

1. $X = (X_1, \dots, X_n)$ ist ein Vektor von n Stichproben aus einer unbekanntem Verteilung, und T ein Schätzer für den Median der Verteilung (etwa der Median der Stichproben).
2. $X = (X_1, \dots, X_n)$ ist ein multiples Alignment der Länge n von m Sequenzen (X_i ist die i -te Spalte des Alignments und somit ein Vektor von m Nukleotiden), und T ist ein Schätzer (Algorithmus) für den phylogenetischen Baum dieses Alignments.

Ein Schätzer ist in der Regel um so besser, je mehr Daten zum Schätzen zur Verfügung stehen. Wäre die Verteilung, aus der X erzeugt wurde, bekannt, könnte man sich beliebig viele weitere Daten beschaffen, um (a) die Schätzung zu verbessern, und (b) durch wiederholtes Ziehen von n Stichproben zu messen, wie stark der Schätzer bei n Stichproben noch variiert. Die wahre Verteilung ist aber unbekannt. Daher hilft man sich mit einem Trick: Man zieht n “neue” Stichproben aus den gegebenen Stichproben X (mit Zurücklegen). Sei $Y = (Y_1, \dots, Y_n)$ eine solche Replizierung (*re-sample*). Ein bestimmtes X_i kann in Y mehrmals oder auch gar nicht enthalten sein.

Dieses re-sampling kann man beliebig oft durchführen. Angenommen, wir beschaffen uns R Replizierungen Y^1, \dots, Y^R . Auf jedes Y^r ($r = 1, \dots, R$) kann man nun den Schätzer T anwenden und erhält R Schätzwerte $T^r := T(Y^r)$. Die Verteilung dieser R Schätzungen im Vergleich zur ursprünglich Schätzung $T(X)$ kann man benutzen, um die inhärente Variabilität von T bei der Eingabe X zu quantifizieren. Zum Beispiel kann man für ein Intervall $I := [T(X) - \frac{\epsilon}{2}, T(X) + \frac{\epsilon}{2}]$ der Breite ϵ angeben, welcher Anteil der Schätzungen T^r in I liegt. Diesen Anteil nennt man den Bootstrap-Wert von I . Wie bereits erwähnt bedeutet ein Bootstrap-Wert von 1.0 aber nicht, dass der wahre gesuchte Wert in I liegt.

Bootstrap heißt Stiefelschlaufe. Man zieht sich an der Stiefelschlaufe $(Y^r)_{r=1, \dots, R}$ aus dem Sumpf der mangelnden Kenntnis der wahren Verteilung wie Münchhausen an seinen Haaren. Die Idee wurde ursprünglich 1979 von Bradley Efron vorgeschlagen.

Der Bootstrap ist nicht der einzige *resampling plan*. Es gibt zum Beispiel auch das Klappmesser (Jackknife), bei dem man n Replizierungen der Größe $n - 1$ betrachtet. Bei der i -ten Replizierung wird X_i aus den Daten weggelassen (leave-one-out). Mit diesen Replizierungen kann man genauso verfahren wie beim Bootstrap. Nachteile beim Jackknife ist allerdings, dass die Anzahl der Replizierungen vorgegeben ist ($R = n$) und dass die Replizierungen eine andere Stichprobenzahl ($n - 1$) als das Original (n) haben. Dafür hängen die Replizierungen deterministisch von X ab.

9.2 Bootstrapping von Bäumen

Zum Bootstrappen von phylogenetischen Bäumen benutzt man Replizierungen Y^r der Alignmentsspalten X , die die Eingabe des Originalproblems bilden. Im folgenden Beispiel betrachten wir $m = 3$

Sequenzen der Länge $n = 4$ und $R = 2$ Replizierungen. Unter den Replizierungen sind die Spaltennummern angegeben, die dem Originalalignment entsprechen.

$$X = (X_1, X_2, X_3, X_4) = \begin{array}{c} \text{ACGT} \\ \text{AAGT} \\ \text{ACTT} \\ \hline 1\ 2\ 3\ 4 \end{array} \quad Y^1 = \begin{array}{c} \text{AAAC} \\ \text{AAAA} \\ \text{AAAC} \\ \hline 1\ 1\ 1\ 2 \end{array} \quad Y^2 = \begin{array}{c} \text{CGCA} \\ \text{AGAA} \\ \text{CTCA} \\ \hline 2\ 3\ 2\ 1 \end{array}$$

In der Realität hat man natürlich mehr und längere Sequenzen, und auch mehrere Replizierungen. Aus jeder Replizierung Y^r wird nun mit demselben Algorithmus jeweils ein Baum T^r geschätzt und mit dem Baum $T = T(X)$ aus den Originaldaten verglichen.

Üblicherweise geschieht dieser Vergleich wie folgt: Jede Kante in T definiert einen *Split* der m Eingabesequenzen. Ein Split ist eine Partition einer Menge M in zwei nichtleere disjunkte Teilmengen M_1, M_2 , deren Vereinigung M ergibt. In diesem Fall gehören zu M_1 alle Sequenzen, die sich auf derselben Seite des Baumes wie ein Endpunkt der Kante befinden, und zu M_2 die anderen Sequenzen. Einen Split schreibt man als $M_1||M_2$, wie wir das auch bei den Quartetten in Abbildung 3 getan haben. Für jede Kante von T gibt man nun den Bootstrap-Wert des von ihr definierten Splits an. Ein Wert von 0.8 bedeutet, dass derselbe Split in 80% aller Replizierungen vorkommt, also einigermaßen robust gegenüber Resampling der Eingabedaten ist.

Bootstrap-Werte werden manchmal dazu benutzt, einen strikten Konsensus-Baum zu erzeugen. Dabei entfernt man alle Kanten mit Bootstrap-Werten von weniger als 50% und verschmilzt deren Endpunkte. Der resultierende Baum ist nicht vollständig aufgelöst. Wenn zum Beispiel bei einem Quartett in Abbildung 3 die mittlere Kante einen Bootstrap-Wert von 0.49 hat, wird aus dem Baum ein sogenannter Stern-Baum (star tree) mit nur einem inneren Knoten, an dem die vier Blätter hängen.

10 Theorie: Statistik von Sequenzalignments

10.1 Motivation und Grundlagen

Wie bereits erwähnt werden Datenbanken oft durchsucht, um zu einer Anfragesequenz (Query) homologe Datenbanksequenzen (Subjects) zu finden, um gegebenenfalls Informationen transferieren zu können. Nun ist Homologie nicht dasselbe wie Sequenzähnlichkeit, aber in der Regel wird man bei genügend großer Ähnlichkeit sehr wohl auf Homologie schließen können. Eine natürliche Frage ist nun: Wie groß muss die Ähnlichkeit (beispielsweise ausgedrückt in dem Smith-Waterman Alignment Score oder BLAST-Score oder in einem anderen Maß) sein, damit dieser Schluss korrekt ist? Diese Frage ist aber schwierig zu beantworten. In der Regel fragt man andersherum: Wenn zufällige Sequenzen mit der Query aligniert werden, wie groß wird der Score dann noch? Man nennt alle Scores, die einen Schwellenwert T überschreiten, signifikant, wenn T so gewählt ist, dass zufällige Sequenzen nur eine geringe Chance haben, diesen Score zu erreichen.

Betrachten wir diesen Rahmen noch etwas genauer: Wenn wir die Query mit einem Subject vergleichen, gibt es zwei mögliche *Wahrheiten* ("true state of nature"): homolog oder nicht homolog. Das Alignment liefert uns einen Score und damit eine Entscheidung: Ist der Score mindestens T , entscheidet man sich, die Sequenzen als homolog anzusehen, ansonsten als nicht homolog. Vier Fälle sind jetzt möglich.

1. Die Sequenzen sind homolog und der Score ist mindestens T . Die Sequenzen werden korrekt als homolog klassifiziert (true positive).
2. Die Sequenzen sind nicht homolog und der Score ist kleiner als T . Die Sequenzen werden korrekt als nicht homolog klassifiziert (true negative).

3. Die Sequenzen sind nicht homolog und der Score ist mindestens T . Die Sequenzen werden inkorrekterweise als homolog klassifiziert. Man spricht von einem Fehler erster Art oder auch von einem false positive.
4. Die Sequenzen sind homolog, aber der Score ist kleiner als T . Die Sequenzen werden fälschlicherweise als nicht homolog klassifiziert. Man spricht von einem Fehler zweiter Art oder von einem false negative.

Natürlich möchte man, dass beide Fehler möglichst selten vorkommen, aber man muss einen Kompromiss eingehen: Erhöht man den Schwellenwert, bekommt man weniger falsch positive, aber auch weniger echt positive (also mehr falsch negative). Der einzige Weg, beide Fehler gleichzeitig zu verkleinern, besteht darin, die Entscheidungsgröße möglichst gut zu wählen. Die Smith-Waterman Alignment Score hat sich in dieser Hinsicht prinzipiell bewährt, aber es ist wichtig, eine möglichst gute Scorematrix zu wählen. Dazu sind die oben diskutierten Methoden sehr gut geeignet.

Um das Signifikanzproblem statistisch zu behandeln, müssen wir noch einige Dinge präzisieren. Dazu zwei wichtige Bemerkungen: Erstens, der Smith-Waterman Algorithmus sucht nach lokalen Ähnlichkeiten, nicht nach globalen. Zweitens, eine Scorematrix $S^{(t)}$ bezieht sich, wie oben beschrieben, immer auf eine bestimmte evolutionäre Divergenzzeit t . Daher sind die Optionen, die man bei der Entscheidung hat, nicht wirklich “homolog” und “nicht homolog”, sondern K_t : “lokal verwandt mit Divergenzzeit t ” und H : “nirgends verwandt, d.h. überall verwandt mit Divergenzzeit ∞ ”. Natürlich werden H und K_t für große Zeiten t zunehmend schwieriger zu trennen, d.h., es wird schwieriger, beide Fehlerwahrscheinlichkeiten klein zu halten. Das Hauptproblem ist, dass man gerade an großen Divergenzzeiten t interessiert ist, um entfernt verwandte Sequenzen zu finden.

Im statistischen Sinne führt man einen Test zwischen der Nullhypothese H und einer Alternative K_t durch. Das Signifikanzniveau α ist gerade die Fehlerwahrscheinlichkeit erster Art, d.h. die Wahrscheinlichkeit, bei Sequenzen, für die H wahr ist, die Entscheidung K_t zu treffen. Die gesamte Theorie in diesem Abschnitt bezieht sich darauf. Insbesondere lassen sich folgende Fragen *nicht* beantworten, da sie “inkorrekt gestellt” sind.

- Bei beobachtetem Score s , wie groß ist die Wahrscheinlichkeit, dass H bzw. K_t wahr ist? Diese Frage kann nicht beantwortet werden, denn H ist entweder wahr oder nicht wahr; dafür gibt es keine Wahrscheinlichkeit. Man muss andererseits fragen: Wenn H wahr ist, wie groß ist dann die Wahrscheinlichkeit, mindestens den Score s zu beobachten? Genau darum geht es in diesem Abschnitt.
- Angenommen, K_t ist wahr. Wie groß ist die Fehlerwahrscheinlichkeit zweiter Art, also die Wahrscheinlichkeit einen Score kleiner als s zu beobachten? Diese Frage lässt sich nicht (ohne weiteres) beantworten, weil K_t ungenügend spezifiziert ist, um daraus ein Modell abzuleiten. Das Problem liegt in der lokalen Verwandtschaft. Erstreckt sich diese über einen kurzen oder einen längeren Bereich der Sequenzen? Um typische Scores für verwandte Sequenzen auszurechnen, müsste man sich hier festlegen. Obwohl Betrachtungen dieser Art sicherlich sinnvoll sind, findet sich dazu bisher nichts in der Literatur.

Die einzige Frage, die sich mit statistischen Methoden behandeln lässt, ist also die oben erwähnte nach der Fehlerwahrscheinlichkeit erster Art. Praktisch geht man wie folgt vor. Man simuliert Sequenzen unter der Annahme H und berechnet mit diesen den Smith-Waterman Alignment Score. Tut man dies oft genug, kann man die Verteilung des Scores in einem Histogramm aufzeichnen und den Schwellenwert T so bestimmen, dass genau der gewünschte kleine Anteil der Scores rechts von T liegt. Nicht nur das: Für jeden beobachteten Score-Wert s lässt sich unter dem gewählten Zufallsmodell mit Hilfe des Histogramms die Wahrscheinlichkeit $\mathbb{P}_H(\text{Score} > s)$ abschätzen.

Definition 10.1 (p-value, Signifikanz). Der p-value eines beobachteten Scores s ist definiert als die Wahrscheinlichkeit, mindestens diesen Score im Nullmodell zu beobachten,

$$\text{p-value}(s) := \mathbb{P}_H(\text{Score} > s).$$

Ein Score mit einem p-value von weniger als 0.05 wird in der Regel *signifikant* genannt; ein Score mit einem p-value von höchstens 0.01 wird *hochsignifikant* genannt.

Übung 10. Angenommen, die Scores sind ganzzahlig und im Wertebereich $0, 1, 2, \dots$. Angenommen weiter, durch Betrachten eines Scorehistogramms stellen wir fest, dass in etwa $\mathbb{P}(\text{Score} = s) = 0.02 \cdot (0.98)^s$ gilt. Man zeige, dass dies in der Tat eine Verteilung definiert ($\sum_{s=0}^{\infty} \mathbb{P}(\text{Score} = s) = 1$) und bestimme den minimalen Schwellenwert T , so dass $\mathbb{P}(\text{Score} \geq T) \leq 0.01$.

Bemerkung: Man kann natürlich nicht erwarten, dass die Scoreverteilung einer so regelmäßigen Form wie in dieser Aufgabe folgt. Allerdings liegt diese Form näher an der Realität, als man auf den ersten Blick vermuten würde.

10.2 Nullmodelle

Oben haben wir gesagt, wir simulieren Sequenzen unter der Nullhypothese H : “Die Sequenzen sind nirgends verwandt”. Um dieses aber konkret durchführen zu können, brauchen wir ein Sequenzmodell, das sogenannte *Nullmodell*. Ein Beispiel für Nullmodelle haben wir schon in Abschnitt 2 kennengelernt, als wir Sequenzen mit Markovketten generiert haben.

Es liegt auf der Hand, dass am Ende die Signifikanzwerte der Datenbanksuchergebnisse (unter Umständen stark) vom Nullmodell abhängen. Man kann also niemals pauschal sagen, ob etwas signifikant oder nicht signifikant ist, sondern immer nur signifikant unter einem gegebenen Nullmodell. Zum Glück liegt bei vielen Anwendungen ein Nullmodell auf der Hand. Bei der Datenbanksuche ist das leider nicht so, und es lohnt sich, darüber nachzudenken.

Was passiert bei einer Datenbanksuche? Man vergleicht eine feste Query Q mit vielen Subjects S_1, \dots, S_N . Das Ziel ist zu entscheiden, ob ein Score von s signifikant ist. Die Frage ist, signifikant unter welchen Bedingungen? Allgemein für alle denkbaren Querys und Subjects (fester Länge)? Für festes Q und alle Subjects? Für festes Q und festes Subject? Hier sind ein paar Möglichkeiten, ein Nullmodell zu definieren.

Komplett randomisiert. Man erzeugt zwei zufällige Sequenzen fester Länge und mit vorgegebener Aminosäureverteilung (Komposition) und berechnet die Scoreverteilung, d.h. man mittelt über viele Querys und Subjects mit als bekannt angenommener Komposition.

Randomisierte Datenbank. Man hält Q fest und erzeugt zufällige Subjects fester Komposition und Länge.

Individuell randomisierte Subjects. Man verwendet N Nullmodelle, eins für jedes Subject. In jedem Nullmodell hält man Q fest, und im i -ten Nullmodell verwendet man zufällige Sequenzen als Subjects, die man durch Randomisierung von S_i erzeugt.

Umgekehrte Subjects. Man hält die Query fest und vergleicht sie mit den rückwärts gelesenen Subjects der Datenbank. (Dieses Modell ist nicht stochastisch.)

In einigen Nullmodellen muss man sich Gedanken über die Komposition machen. Es ist sicher unrealistisch, die Sequenzen gleichverteilt auszuwürfeln (jede Aminosäure mit Wahrscheinlichkeit $1/20$ auszuwählen). Man könnte jetzt auf die Ergebnisse aus Abschnitt 2 zurückgreifen und aus der Query bzw. den Subjects ein Markovmodell schätzen und dieses zur Randomisierung verwenden. Im Regelfall

wird aber das einfachere iid Modell verwendet, in dem Aminosäure i an jeder Stelle mit der gleichen Wahrscheinlichkeit π_i auftritt.

Alle hier genannten Nullmodelle haben verschiedene Vor- und Nachteile.

Übung 11. Diskutieren Sie die Modelle unter Gesichtspunkten wie Eignung für die Datenbanksuche, Aufwand, Verhalten unter verschiedenen Queries mit sehr unterschiedlicher Komposition, etc.

10.3 BLAST-Statistik und Karlin-Altschul Theorie

Das Datenbanksuchprogramm BLAST verwendet das komplett randomisierte Nullmodell. Das bedeutet insbesondere, dass die Signifikanzwerte überhaupt nicht von der Query abhängen, mit der die Datenbanksuche durchgeführt wird. Die BLAST p-values und E-values hängen nur von der Länge der Query und der jeweiligen Subjects ab. Das bedeutet unter anderem, dass einer ungewöhnlichen Querykomposition keine Rechnung getragen werden kann. Im Gegenzug bietet dieses Modell den Vorteil, dass man nicht für jede Query die Parameter der Scoreverteilung neu schätzen muss. Sind die Parameter einmal bestimmt (sie hängen nur von der Scorematrix und den Gapkosten ab), erhält man Signifikanzwerte ohne großen Rechen- oder Simulationsaufwand.

Ein weiter Grund für die Wahl dieses Modells ist teilweise historisch: In der ersten Version fand BLAST nur Alignments ohne Gaps. Für diese lässt sich im komplett randomisierten Modell eine geschlossene mathematische Theorie entwickeln, die ganz ohne Simulationen auskommt. Die Hauptergebnisse stammen von Amir Dembo, Samuel Karlin und Stephen Altschul; wir stellen sie hier zusammen.

Satz 10.2 (Karlin-Altschul Theorie für gaploses Alignment). Für zwei zufällige Sequenzen der Längen m und n mit nach π iid verteilten Aminosäuren und eine Scorematrix S mit negativem erwartetem Score ($\sum_{i,j} \pi_i \pi_j S_{ij} < 0$) gilt:

1. Die erwartete Anzahl der unabhängigen Matches (Alignments) mit Score größer als T zwischen den beiden Sequenzen beträgt

$$\mu_T := Kmn \exp(-\lambda T).$$

Dabei ist $\lambda > 0$ die eindeutige positive Lösung der Gleichung $\sum_{i,j} \pi_i \pi_j \exp(\lambda S_{ij}) = 1$ und $K > 0$ eine Konstante, die ebenfalls nur von π und S abhängt, und sich durch eine (komplizierte) Reihenentwicklung darstellen lässt.

2. Die Anzahl der unabhängigen Matches ist approximativ Poisson-verteilt. Insbesondere gilt für die Wahrscheinlichkeit, mindestens einen Match mit Score größer als T zu beobachten,

$$\mathbb{P}(\text{Bestscore} > T) = 1 - \exp(-\mu_T).$$

Diese Form der Verteilung in T nennt man auch Extremwertverteilung (EVD, extreme value distribution).

Ein paar Kommentare dazu:

- Die Resultate gelten asymptotisch, d.h. für $n, m \rightarrow \infty$. Für endliche Sequenzen sind sie nur approximativ richtig und desto besser, je länger die Sequenzen sind.
- An der Form von μ_T sieht man, dass die erwartete Anzahl von Matches linear mit dem Produkt der Sequenzlängen wächst (also mit der Größe des Suchraums). Das macht Sinn: Ein solcher Match kann an jeder der mn möglichen Kombinationen von Startpositionen in den Sequenzen beginnen, wenn man von Endeffekten absieht.
- Erhöht man den Scoreschwellenwert T um 1, fällt die erwartete Anzahl der Matches um den Faktor $\exp(-\lambda)$.

- Wenn T so groß ist, dass $1 - \exp(-\mu T)$ sehr viel kleiner als 1 wird, gilt approximativ $1 - \exp(-\mu T) \approx \mu T$, also

$$\mathbb{P}(\text{Bestscore} > T) \approx Kmn \exp(-\lambda T),$$

d.h. auch die Wahrscheinlichkeit, dass das beste gaplose Alignment einen Score größer als T aufweist, fällt schließlich exponentiell mit T .

- Eine Verteilung, die sich für große T in der Form $\mathbb{P}(\text{Score} > T) \approx C \cdot q^T$ schreiben lässt (hier $C = Kmn$ und $q = \exp(-\lambda)$) nennt man im Englischen auch “*geometric-like*”, weil sie wie die geometrische Verteilung fällt (einfach exponentiell mit T).
- Aus den beiden letzten Kommentaren folgt insbesondere, dass es überhaupt keinen Sinn macht, die Scoreverteilung durch eine Normalverteilung zu approximieren!

Der obenstehende Satz lässt sich als exaktes mathematisches Resultat formulieren. Er ist jedoch nicht auf Alignment mit Gaps anwendbar. Für diesen Fall ist die Theorie sehr viel schwieriger, und es gibt bisher nur wenige Ergebnisse. In der Praxis hilft man sich wie folgt: Man nimmt an, dass auch bei Alignments mit Gaps eine Extremwertverteilung der Score vorliegt (dies wird in der Tat durch Simulationen bestätigt, ist aber nicht mathematisch bewiesen) und steht nun vor der Aufgabe, die Parameter K und λ zu berechnen bzw. zu schätzen. In dem gewählten Nullmodell (komplette Randomisierung) hängen diese Parameter nur von der Scorematrix und den Gapkosten ab. Sie müssen also nur einmal für jede Scorematrix geschätzt werden und können dann immer wieder verwendet werden. Die Schätzung geschieht, indem eine möglichst große Zahl von Sequenzen zufällig erzeugt und miteinander aligniert wird, und durch Betrachten des Scorehistogramms. Es gibt mehrere Verfahren, die Parameter aus dem Histogramm heraus zu schätzen, doch diese sollen hier nicht vertieft werden.

Die Parameter K und λ sind für die wichtigsten Scorematrizen im BLAST-Programm fest verdrahtet (d.h. sie stehen als Konstanten im Programmcode). Das ist auch der Grund dafür, warum man bei BLAST nur wenige Scorematrizen zur Auswahl hat.

10.4 Query-spezifische Signifikanzwerte

In vielen Fällen ist es besser, die Signifikanzwerte (p-values) Query-spezifisch zu berechnen, d.h. anstelle des komplett randomisierten Nullmodells das Modell mit randomisierter Datenbank, aber fester Query zu verwenden. Im übrigen erlaubt dieser Rahmen auch, PSSMs (position specific score matrices) zu behandeln, so dass der Score-Vektor an jeder Stelle der Query ein anderer sein kann (mit gewissen Einschränkungen).

Es ist unwahrscheinlich, dass man in diesem Fall eine gute mathematische Theorie für die Scoreverteilung entwickeln kann, da sehr viele Parameter (neben der Scorematrix die gesamte Querysequenz bzw. die PSSM) berücksichtigt werden müssen. Andererseits ist man an der genauen Form der Verteilung nicht interessiert. Man möchte ja den Schwellenwert T berechnen, so dass $\mathbb{P}(\text{Score} \geq T) = 0.01$ gilt; statt 0.01 ist oft ein noch kleinerer Wert von Interesse. Das bedeutet, man interessiert sich nur für das Verhalten der Verteilung für große T . Hier kann man fragen: Wenn man schon weiß, dass der Score des besten Alignments mehr als s beträgt, wie groß ist dann die Wahrscheinlichkeit, dass er sogar mehr als $s + 1$ beträgt? Nennen wir den zusätzlichen Faktor q :

$$\mathbb{P}(\text{Score} > s + 1 \mid \text{Score} > s) = \frac{\mathbb{P}(\text{Score} > s + 1)}{\mathbb{P}(\text{Score} > s)} =: q < 1$$

Man kann nun argumentieren, dass dieses Verhältnis unabhängig von s ist, sobald s groß genug ist. Statt q schreibt man $\exp(-\lambda)$ mit einem $\lambda > 0$ und erhält

$$\mathbb{P}(\text{Score} > T) = C \exp(-\lambda T) \quad (T \text{ groß})$$

mit einer Konstanten $C > 0$. Jetzt hat man fast dieselbe Form wie in der Karlin-Altschul Statistik für große T . Man kann auch noch $C = Kn$ setzen, wobei n die Subject-Länge ist. Den Faktor m (Query-Länge) aus der Konstanten herauszuziehen, macht in diesem Fall keinen Sinn, da die Konstanten K und λ jetzt von der gesamten Querysequenz abhängen und nicht nur von ihrer Länge.

Es stellt sich die Frage nach der praktischen Berechnung der Query-spezifischen Parameter K und λ . Man muss in diesem Fall auf Simulationen zurückgreifen. Das Problem ist, dass das Hauptinteresse den seltenen Ereignissen gilt ($\text{Score} > T$ für große T), d.h., man muss sehr lange simulieren, bis ein solches Ereignis zufällig auftritt und noch länger, um die Wahrscheinlichkeit präzise schätzen zu können. Die Simulationen zur Bestimmung der Parameter könnten in diesem Fall länger dauern als die Datenbanksuche selbst, was in der Praxis sicherlich unverträglich ist. Das ist auch der Grund, warum momentan in der Praxis keine Query-spezifischen p-values verwendet werden.

10.5 Mehrfaches Testen und E-Values

Bei der Datenbanksuche ergibt sich (unabhängig vom gewählten Nullmodell) ein weiteres Problem: Man vergleicht die Query ja nicht mit nur einem Subject, sondern mit (hundert)tausenden. Man hat also das Problem des mehrfachen Testens.

Nehmen wir an, wir suchen mit der Query gegen eine Datenbank, die nur aus zufälligen Sequenzen besteht. In diesem Sinne kann kein echter Treffer dabei sein, aber einige Scores können durch Zufall so hoch sein, dass sie wie wirkliche Treffer aussehen. Selbst wenn der Schwellenwert T für jedes einzelne Subject so gewählt ist, dass die Chance nur p (der p-value, z.B. 0.01) ist, ist die Chance, dass es unter allen N Subjects mindestens ein solches gibt, natürlich viel höher. Die erwartete Anzahl an Subjects mit Score größer als T ergibt sich als

$$E := Np.$$

Die Wahrscheinlichkeit, dass es mindestens ein solches Subject gibt, beträgt

$$P := 1 - (1 - p)^N \approx 1 - \exp(-Np) \approx Np.$$

Die letzte Approximation gilt für kleines Np (sehr viel kleiner als 1).

In diesem Zusammenhang nennt man p auch oft den Sequenz-p-value, P den Datenbank-p-value, und E den (Datenbank-)E-value. Der Scoreschwellenwert T wird z.B. so gewählt, dass $E = 0.01$ gilt, so dass man zufällige Treffer weitgehend ausschließen kann. Betrachten wir dies am Beispiel der BLAST-Statistik. Wir hatten gesehen, dass hier mit als bekannt vorausgesetzten Konstanten $\lambda > 0$ und $K > 0$ für eine Query der Länge m und ein Subject der Länge n gilt:

$$p = p(T) \approx Kmn \exp(-\lambda T) \quad (T \text{ groß}).$$

Eine Datenbank der Gesamtgröße D kann man beim Suchen gegen ein Subject der Länge n als eine Menge von $N = D/n$ Sequenzen derselben Länge n auffassen. Damit ergibt sich

$$E = E(T) \approx K D m \exp(-\lambda T) = 0.01,$$

und daraus der Schwellenwert

$$T = \frac{\log(100 K D m)}{\lambda}.$$

Übung 12. Ermitteln Sie die Gesamtzahl D der Aminosäuren in der aktuellen SWISSPROT Release und die Konstanten K und λ für die BLOSUM 62. Bestimmen Sie den Schwellenwert T für Queries der Längen 100, 200 und 500.

Übung 13. Vor zehn Jahren war eine hypothetische Datenbank DBX noch so klein, dass ein Score von 57 als signifikant bezeichnet wurde ($T = 55$). Durch Experimente wurde bestätigt, dass es sich um eine echte Homologie handelte. Inzwischen ist die Datenbank (also die Größe D) jedoch soweit gewachsen, dass $T = 60$ gilt, und die genannte Homologie durch die Datenbanksuche nicht mehr gefunden wird. Das Signal ist gewissermaßen im Rauschen der Datenbank untergegangen. Sollte man also den Datenbanken statt dessen keine neuen Sequenzen hinzufügen?

Die Politik hat das Problem erkannt und berät einen Gesetzesentwurf, die Maximalgröße von Datenbanken zu begrenzen und lieber mehrere kleine Datenbanken zu verwenden. Als Bioinformatik-Experte werden Sie um eine Stellungnahme gebeten.

11 Theorie: Verfahren des Maschinenlernens

Der Begriff “Maschinenlernen” (machine learning) ist eigentlich nur ein besser klingender Ersatz für “Datenanalyse” (data analysis) oder “Approximation von Funktionen”.

Nach einer Einführung der Grundbegriffe stellen wir neuronale Netze und Support Vector Machines als Methoden vor. Das Anwenden fertig trainierter Modelle ist dabei relativ einfach; das Trainieren selbst geht über den Rahmen dieses Kurses hinaus.

11.1 Grundbegriffe

Klassifikation und Regression. Wir unterscheiden zunächst zwei Arten von Lernproblemen: Klassifikation und Regression.

Bei der Klassifikation geht es darum, Eingabedatensätze (Stichproben, samples) einer von endlich vielen verschiedenen Klassen zuzuordnen. Insbesondere betrachten wir das *binäre Klassifikationsproblem*, in dem es nur zwei Klassen gibt (oft + und – genannt). In der Krebsdiagnose könnten die Stichproben zum Beispiel Bilddaten oder Genexpressionsdaten von Zellen sein, und die möglichen Klassen könnten “gesund (–)” und “Tumor (+)” sein. Klassifikationsprobleme treten sehr häufig in der Bioinformatik auf. Zum Beispiel lässt sich auch eine Datenbanksuche so auffassen; jede Datenbanksequenz soll als verwandt (+) oder nicht verwandt (–) zur Query klassifiziert werden.

Regressionsprobleme treten weniger häufig auf. Hier soll statt einer Klasse eine beliebige Funktion der Eingabedaten zu schätzen, zum Beispiel die voraussichtliche Lebenserwartung.

Wir beschränken uns auf binäre Klassifikationsprobleme. Mehrere Klassen können oftmals durch wiederholte binäre Klassifikation behandelt werden (one-versus-rest, all-pairs-one-versus-one).

Überwachtes und unüberwachtes Lernen. Bei der Klassifikation unterscheiden wir zwischen *überwachtem Lernen* (*supervised learning*) und *unüberwachtem Lernen* (*unsupervised learning*).

Der überwachte Lernproblem ist in der Regel einfacher. Wir bekommen einige Datensätze mit ihrer korrekten Klasse zur Verfügung gestellt und können daraus unsere Schlüsse ziehen. Der Lernende (das Computerprogramm) kann so lange trainiert werden, bis er zumindest diese *Trainingsdaten* einigermaßen korrekt klassifizieren kann.

Beim unüberwachten Lernen sind die korrekten Klassen unbekannt. Das Programm soll von sich aus feststellen, ob sich die gegebenen Samples “auf natürliche Weise” in zwei oder mehr Klassen einteilen lassen. Was “auf natürliche Weise” bedeutet, lässt sich häufig nur mit problemspezifischem Fachwissen genau definieren. In der Regel verwendet man hier *Clustering-Verfahren*, die sukzessive “ähnliche” Samples zu einer Klasse zusammenfassen. Auf diese gehen wir hier nicht näher ein.

Betrachten wir das überwachte Klassifizierungsproblem etwas genauer: Ein Datensatz x stammt aus dem *Merkmalsraum* (*feature space*) \mathcal{X} und gehört einer Klasse $y = y(x) \in \{-1, +1\}$ an. Der Merkmalsraum \mathcal{X} kann relativ kompliziert aufgebaut sein (Bilddaten, reelle Zahlen wie Größe oder Gewicht,

Sequenzdaten, etc.) und umfasst alle potenziell möglichen beobachtbaren Datensätze. Eine *Entscheidungsregel* ordnet jedem möglichen $x \in \mathcal{X}$ eine Klasse $y \in \{0, 1\}$ zu, partitioniert also den Raum \mathcal{X} in einen “positiven” Bereich \mathcal{X}^+ und einen “negativen” Bereich \mathcal{X}^- . Wann ist eine Entscheidungsregel gut? Sie sollte auf jeden Fall die gegebenen Trainingsdaten relativ korrekt klassifizieren.

Das führt auf folgende (schlechte) Idee für eine Entscheidungsregel: Für jeden zu klassifizierenden Datensatz x , schaue nach, ob x in den Trainingsdaten vorkommt. Wenn ja, klassifiziere x dementsprechend; ansonsten klassifiziere x als “negativ”. Offensichtlich werden alle Trainingsdaten korrekt klassifiziert, aber alle neuen Datensätze werden blind als “negativ” klassifiziert. Wir haben also aus den Trainingsdaten nichts gelernt; wir haben nicht *generalisiert*. Man spricht hier auch von *overfitting*. Wir haben die Trainingsdaten perfekt gelernt, aber vermutlich nicht die essentiellen Merkmale der positiven und negativen Klasse extrahiert.

Es folgt: Eine gute Entscheidungsregel ist gekennzeichnet durch

1. Korrekte Klassifizierung der Trainingsdaten
2. Generalisierungsfähigkeit (geringe Komplexität der Teilräume \mathcal{X}^+ und \mathcal{X}^-).

Der zweite Punkt ist im Grunde äquivalent zu Occam’s Razor: Wenn es zwei Entscheidungsregeln gibt, die auf den Trainingsdaten gleich gut abschneiden, so ist die “einfachere” vorzuziehen.

Wir gehen im weiteren davon aus, dass sich die Eingabedaten als Vektoren in $\mathcal{X} = \mathbb{R}^n$ (also als n -dimensionale reellwertige Vektoren) beschreiben lassen. Dies kann durch geeignete Einbettungsverfahren erreicht werden. Das Problem, z.B. Sequenzdaten einzubetten, ist allerdings subtil und erfordert oft problemabhängige Lösungen.

11.2 Methoden

Lineare Trennung. Die einfachste Art, $\mathcal{X} = \mathbb{R}^n$ in zwei Bereiche zu teilen, ist, den Raum mit einer Hyperebene zu zerschneiden. Abbildung 4 veranschaulicht dies für $n = 2$. Wenn sich die Daten $x \in \mathcal{X}$, die zu verschiedenen Klassen gehören, überhaupt linear trennen lassen, ist dies eine gute Lösung. Eine Hyperebene im \mathbb{R}^n wird durch eine Gleichung der Form $\langle x, w \rangle + b = 0$ beschrieben. Dabei ist $w \in \mathbb{R}^n$ der Normalenvektor der Ebene und b ein Lageparameter. Die Ebene wird also durch $n + 1$ Parameter beschrieben.

Um die Klassifikation möglichst robust zu machen, möchte man die Ebene so legen, dass ihr Abstand (*margin*) zu den nächstgelegenen Trainingspunkten möglichst groß wird. Dies lässt sich durch Lösen eines quadratischen Optimierungsproblems erreichen.

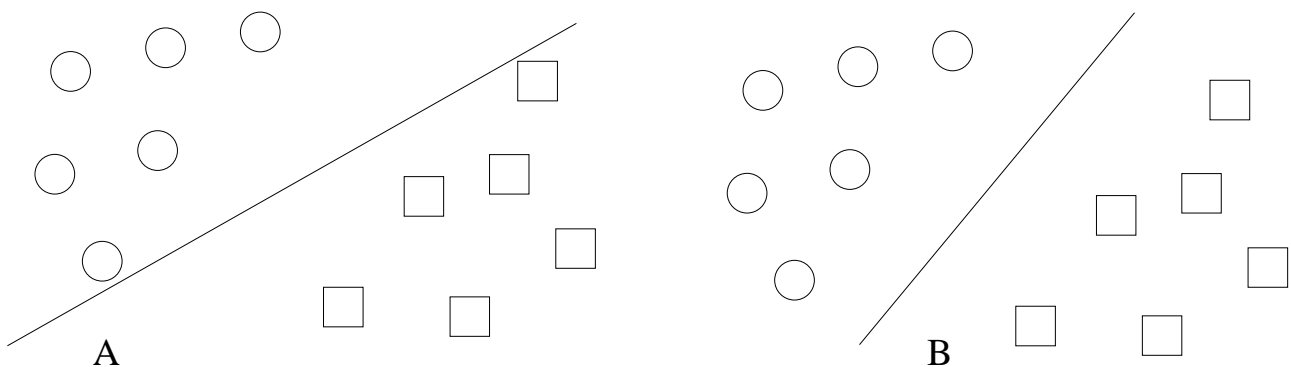


Abbildung 4: Lineare Trennung der Klassen durch eine Hyperebene. Links: Unrobuste Trennung (kleiner Margin). Rechts: Robuste Trennung (großer Margin).



Abbildung 5: Lineare Trennung ist unmöglich.

Leider eignet sich die lineare Trennung kaum in der Praxis, denn bereits sehr einfache Trainingsdatensätze im \mathbb{R}^2 sind nicht linear trennbar (Abbildung 5). Wir benötigen also nichtlineare Entscheidungsregeln (die aber wegen der Gefahr des Overfittings nicht zu kompliziert werden dürfen).

Neuronale Netze. Die typische Architektur eines neuronalen Netzes ist in Abbildung 6 dargestellt. In der Eingabeschicht (input layer) finden wir die Merkmalskomponenten $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. Diese werden über verschiedene Gewichte w_{ik} und nichtlineare Transfer-Funktionen f_k wie folgt zu neuen Werten y_k im sogenannten hidden layer verknüpft:

$$y_k = f_k(b_k + \sum_i x_i \cdot w_{ik}).$$

Die Nichtlinearität entsteht also nur durch die Transferfunktion. Ein typisches Beispiel ist die Funktion

$$f(x) = \frac{2}{1 + e^{-x}} - 1,$$

die $[-\infty, +\infty]$ nichtlinear auf $[-1, 1]$ mit $f(0) = 0$ abbildet. Statt nur einem hidden layer kann man mehrere hintereinander verwenden. Die Anzahl der Knoten in den hidden layers sollte zur Ausgabeschicht (output layer) hin immer kleiner werden. Dies entspricht intuitiv einer Informationsverdichtung der Eingabedaten auf die essentiellen Merkmale, die für die Klassifikation entscheidend sind.

Zur Klassifikation in der Ausgabeschicht verwendet man in der Regel den Wert

$$z = \text{sign}(c + \sum_k y_k \cdot v_k).$$

Dabei sind y_k die Werte in den Knoten des letzten hidden layers, v_k deren Gewichte, c eine additive Konstante, sign die Vorzeichenfunktion, und $z \in \{-1, 0, +1\}$ die Klassenvorhersage. Im Falle $z = 0$ oder $|z| \approx 0$ ist das neuronale Netz unsicher.

Beim Trainieren des Netzes müssen die Gewichte so gewählt werden, dass die Trainingsdaten möglichst korrekt klassifiziert werden. Man fängt dabei meist mit zufälligen Gewichten an. Das führt zu Klassifikationsfehlern. Durch Zurückverfolgen (backpropagation) des Fehlers zu seiner Ursache (ein falsch eingestelltes Gewicht) werden die Gewichte nach und nach angepasst.

Bei neuronalen Netzen wird oft bemängelt, dass sich die Werte in den Hidden layers und die Gewichte nicht direkt interpretieren lassen. Man spricht auch von einer Black-Box-Methode.

Support Vector Machines. Support Vector Machines beschreiten einen anderen Weg, um eine nichtlineare Entscheidungsregel zu finden. Die Idee ist, eine lineare Entscheidungsregel auf geeignet transformierten Eingabedaten anzuwenden. Dazu wählt man eine geeignete Abbildung $g : \mathcal{X} \rightarrow H$, wobei H ein hochdimensionaler (eventuell unendlichdimensionaler) Hilbertraum ist. Der Ausdruck Hilbertraum besagt, dass auf H wieder ein Skalarprodukt $\langle u, y \rangle$ für $u, y \in H$ definiert ist. In diesem Zusammenhang nennt man \mathcal{X} auch Eingaberaum (input space) und H Merkmalsraum (feature

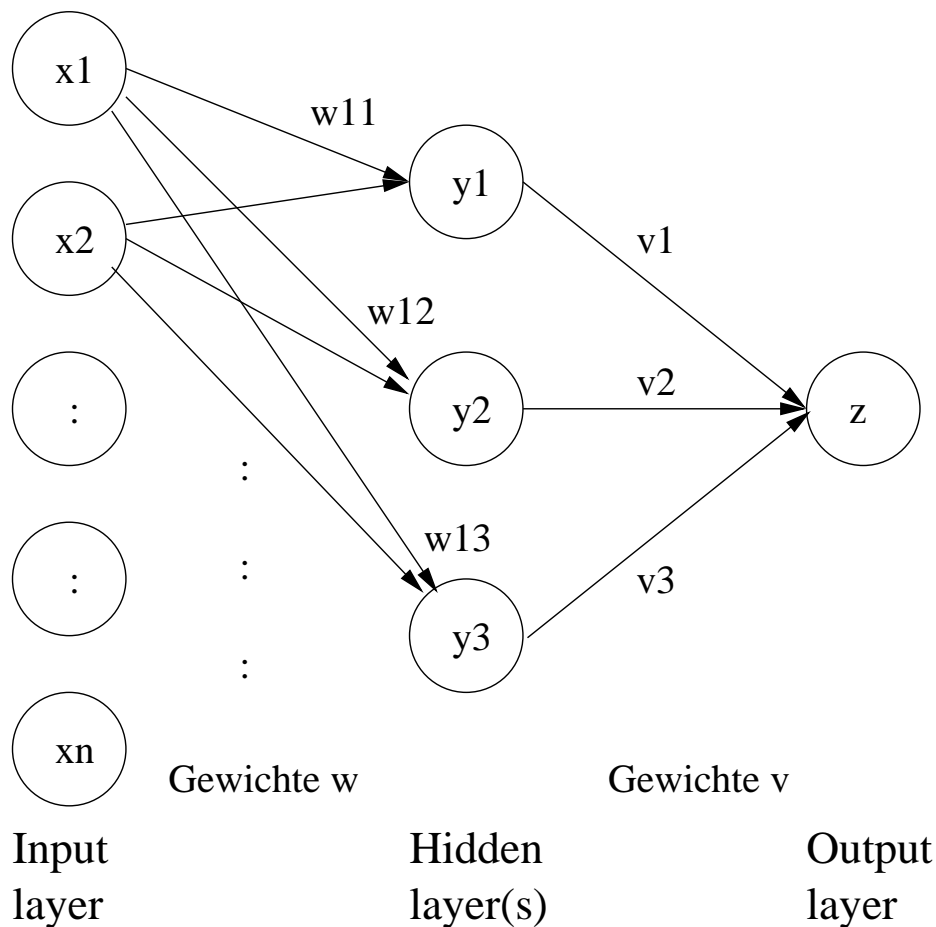


Abbildung 6: Ein typisches neuronales Netz.

space). Merkmale sind hier also die schon transformierten Eingabedaten. Ist H von genügend großer Dimension, ist die Chance groß, dass sich die beiden Klassen in H linear trennen lassen, obwohl dies im Eingaberaum \mathcal{X} nicht möglich war.

Die Klassifikation erfolgt in die Klasse

$$y(x) := \text{sign}(c + \langle w, g(x) \rangle)$$

mit einem durch Training zu bestimmenden Gewichtsvektor $w \in H$. Hier ist $\langle \cdot, \cdot \rangle$ das in H definierte Skalarprodukt. Eine schöne Eigenschaft dieser Formulierung ist, dass sich das Argument der Signum-Funktion als Summe von Skalarprodukten der transformierten Trainingsdaten (x_t, y_t) , $t = 1, \dots, T$, mit $x_t = (x_{t,1}, \dots, x_{t,n})$ schreiben lässt (hier ohne Beweis):

$$y(x) = \text{sign} \left(c + \sum_{t=1}^T \alpha_t \cdot y_t \cdot \langle g(x_t), g(x) \rangle \right).$$

Daraus folgt, dass man sich über die Transformation g keine Gedanken machen muss. Es genügt, die Kernel-Funktion

$$K(x_i, x) := \langle g(x_i), g(x) \rangle$$

zu studieren, die gewissermaßen das Skalarprodukt auf dem unbekanntem hochdimensionalen Raum H emuliert, ohne dass man H kennen muss. Wenn man nachweisen kann, dass eine Funktion K einige

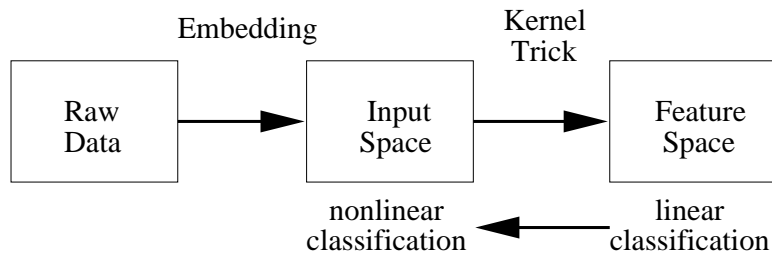


Abbildung 7: Der Kernel-Trick bei SVMs.

leicht nachzurechnende Eigenschaften hat, so gibt es automatisch irgendeinen Raum H mit den genannten Eigenschaften. Das Vorgehen, eine lineare Trennung in einem unbekanntem hochdimensionalen Raum durchzuführen, nennt man auch den *Kernel-Trick*, siehe Abbildung 7.

Häufig verwendete Kernelfunktionen sind

$$\begin{aligned}
 K(x, y) &:= (1 + \langle x, y \rangle)^p, \\
 K(x, y) &:= \tanh(\langle x, y \rangle - \delta), \\
 K(x, y) &:= \exp(-\|x - z\|^2 / (2\sigma^2)).
 \end{aligned}$$

Achtung: Das Skalarprodukt in diesen Kernels ist das im Raum \mathcal{X} .

Man erhält also die Entscheidungsregel

$$y(x) = \text{sign} \left(c + \sum_{t=1}^T \alpha_t \cdot y_t \cdot K(x_t, x) \right).$$

Ein weiterer positiver Aspekt dabei ist die Tatsache, dass viele α_t verschwinden. Die Trainingsdaten (x_t, y_t) mit $\alpha_t \neq 0$ sind gerade diejenigen, die in H am Rand der Klassen liegen, also die trennende Hyperebene definieren. Man nennt die entsprechenden x_t daher auch die *support vectors*, daher der Name “Support Vector Machine”.

Das Bestimmen der Gewichte α_t lässt sich leicht durch Lösen eines quadratischen Optimierungsproblems bewerkstelligen. Als undurchsichtig bei SVMs gestaltet sich die Wahl des Kernels. Ist dieser jedoch einmal gewählt, sind alle verbleibenden Parameter geometrisch interpretierbar. Es kann natürlich auch der Fall eintreten, dass die Klassen für keine Kernel-Wahl linear trennbar sind. In dem Fall kann man Klassifikationsfehler zulassen und beispielsweise die beste trennende Hyperebene mit maximal einem oder zwei Klassifikationsfehlern bestimmen.