Hidden Markov Modell zum unehrlichen Casino

Algorithmen der Bioinformatik I

Konrad Buchmann

01.01.2021

Programmiersprachen und Bibliotheken

Für die Lösung der Aufgaben wurde Rust verwendet. Außerdem die folgenden Bibliotheken:

- Plotters 0.3

In der `README.md` findet sich eine Build-Anleitung.

Ergebnisse

Viterbi Algorithmus

Der berechnete Viterbi-Pfad ist der folgende:

```
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLFFFFFFFF
FFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLFFFFFFFFFF
```

Dieser stimmt exakt mit dem aus Durbin überein, allerdings fehlt das letzte Element. Einen Grund hierfür konnte ich leider nicht finden.

Die Sensitivität beträgt 79.2%, die Spezifität 95.59%.

Da im Viterbi-Algorithmus sehr viele Wahrscheinlichkeiten multipliziert werden, kommen bei langen Sequenzen sehr kleine Zahlen zu Stande. Um einen möglichen Underflow des Exponenten zu verhindern und Rundungsfehler zu minimieren, bietet es sich an die Berechnung der Viterbi-Variable im log-space durchzuführen. Die Multiplikation wird im log-space zu einer Addition, der restliche Algorithmus ist identisch, da der logarithmus monoton ist. In diesem Fall ist es allerdings nicht zu einem Underflow gekommen.

Forward- und Backward-Algorithmus

Der Viterbi-Algorithmus ist nicht ideal in Situationen, bei denen die Zustände an einer oder mehreren Stelle ähnliche Wahrscheinlichkeiten haben. Hier ist es nicht einfach zu entscheiden durch welchen Zustand der optimale Pfad verläuft. Der mit Viterbi gefundene optimale Pfad ist in diesem Fall nicht der einzige mit einer signifikant hohen Wahrscheinlichkeit.

Eine Möglichkeit dieses Problem zu lösen ist der Forward-Algorithmus. Dafür betrachten wir an jeder Stelle die Wahrscheinlichkeit, dass das HMM in diesem Zustand die gegebene Beobachtung an dieser Stelle produziert, nachdem es alle vorherigen Beobachtungen über einen beliebigen Weg produziert hat. Dafür führen wir analog zum Viterbi-Algorithmus eine Forward-Variable $f_l(i) = e_l(x_i) \sum_k f_k(i-1)a_{kl}$. Mit dynamischer Programmierung wird diese nun für alle Zustände und alle Positionen der beobachteten Sequenz berechnet, genau wie beim Viterbi-Algorithmus.

Mit logarithmischen Wahrscheinlichkeiten zu rechnen ist nun aufgrund der Summation nicht mehr trivial. Man kann allerdings, wie in der Vorlesung beschrieben, die Identität $log(p+q) = p+log(1+exp(q-p))$ verwenden. Rust bietet hierfür die Funktion `ln_1p()` (siehe `https://doc.rust-lang.org/std/primitive.f64.html#method.ln_1p`) für noch mehr numerische Präzision.

Kehrt man den Forward-Algorithmus um, erhält man den Backward-Algorithmus. Hierbei wird die Wahrscheinlichkeit betrachtet, dass eine Beobachtung von einem spezifischen Zustand erzeugt wird, wenn alle nachfolgenden Beobachtungen produziert wurden.

Kombiniert man die Ergebnisse des Forward- und Backward-Algorithmus erhält man für jede einzelne Beobachtung und für jeden Zustand eine Wahrscheinlichkeit, dass der Pfad, der die gesamte Sequenz erzeugt durch diesen Zustand verläuft. Diese sog. posterioren Wahrscheinlichkeiten sind für die gegebene Sequenz in Abb. 1 gezeigt. Sie berechnen sich aus den Forward- und Backward-Variablen mit der Formel $P(\pi_i = k|x) = \frac{f_k(i)b_k(i)}{P(x)}$, wobei $P(x)$ die Wahrscheinlichkeit des Endzustandes entspricht, welche mit dem Forward-Algorithmus berechnet wurde.

Der Plot stimmt mit dem Beispiel aus Durbin überein, allerdings lassen sich bei genauerem hinschauen kleine Unterschiede erkennen. Dies könnte auf propagierte Rundungsfehler zurückzuführen sein, da meine Implementation nicht mit log-Wahrscheinlichkeiten rechnet, und damit numerisch nicht ganz präzise ist.

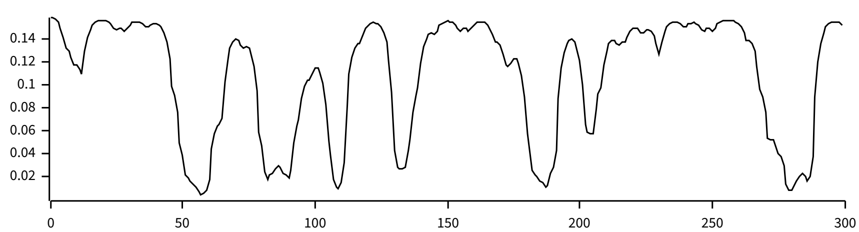

Figure 1: A-Posteriori Wahrscheinlichkeiten für den fairen Würfel. Die x-Achse zeigt die Nummer des Wurfs, die y-Achse die Wahrscheinlichkeit, dass für diesen Wurf ein fairer Würfel verwendet wurde.