Programmering og Modellering (PoM)
Ugeseddel 15 — Uge 2 — Deadline Ingen aflevering

Kim Steenstrup Pedersen, Katrine Hommelhoff Jensen, Knud Henriksen,
Mossa Merhi og Hans Jacob T. Stephensen

4. januar 2015

1 Plan for ugen

Denne uge tager vi fat på emnerne: Deterministiske og stokastiske tilstandsmaskiner, grammatikker og regulæreudtryk. Vi kan anvende tilstandsmaskiner til modellering af forskellige aspekter og vi skal se nærmere på nogle eksempler. Vi vil også kort kigge på kontekstfri grammatikker og anvendelse af disse samt på regulæreudtryk og hvorledes disse anvendes i Python til at lede efter mønstre i strenge.

Vi vil kigge nærmere på en type af stokastiske modeller kaldet Markov kæder. Denne type modeller finder anvendelser i så forskellige områder som maskinoversættelse af naturlige sprog, analyse af DNA sekvenser, prædiktion af aktiekurser og visuel opmåling af tre-dimensional menneskelig bevægelse.

1.1 Stokastiske modeller og Markov kæder

En Markov kæde er en stokastisk model for sekventielle fænomener beskrevet ved en sekvens af stokastiske variable. De stokastiske variable i sekvensen er indekseret på samme vis som en liste i Python, dvs. at første element har index $n = 0$ og næste element $n = 1$, også videre. For et vilkårligt element i sekvensen med index n er modellens tilstand beskrevet ved den stokastiske variabel $X_n \in \Omega$, som tager værdier fra en mængde Ω. Til vores diskrete stokastiske variabel X_n hører der en sandsynlighedsfordeling beskrevet ved sandsynlighedsmassefunktionen $p(X_n)$, således at for et konkret element fra mængden $\omega \in \Omega$, så angiver $p(X_n = \omega)$ sandsynligheden for at X_n antager værdien ω. Ligeledes gælder der at $p(X_n)$ er normaliseret således at

$$\sum_{i=1}^{K} p(X_n = \omega^{(i)}) = 1 \ , \tag{1}$$

hvor $K = |\Omega|$ er antal elementer i mængden Ω og $\Omega = \{\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(K)}\}$. Der gælder også at $p(X_n = \omega^{(i)}) \geq 0$ for alle $\omega^{(i)} \in \Omega$.

Udover at kende sandsynligheden for de enkelte elementer i sekvensen er vi også interesseret i at kende sandynligheden for forekomsten af en bestemt sekvens af elementer. Vi ønsker altså at modellere sandsynlighedsmassefunktionen $p(X_0, X_1, \ldots, X_N)$, hvor N angiver antal elementer i sekvensen. Hvis Ω eksempelvis angiver mængden af bogstaver så angiver $p(X_0 = \omega_0, X_1 = \omega_1, \ldots, X_N = \omega_N)$ sandsynligheden for en bestemt sekvens af elementer.

En Markov kæde er en konkret model for $p(X_0, X_1, \ldots, X_N)$, hvor vi antager at tegnet på indeks n kun afhænger af tegnet på indeks $n-1$. Denne afhængighed beskrives ved den betinget sandsynligheden for X_n givet at vi kender tegnet på indeks $n-1$, $X_{n-1} = \omega_{n-1}$, og vi opskriver denne som $p(X_n | X_{n-1} = \omega_{n-1})$. Der gælder at $p(X_n | X_{n-1} = \omega_{n-1})$ er normaliseret således at

$$\sum_{i=1}^{K} p(X_n = \omega^{(i)} | X_{n-1} = \omega_{n-1}) = 1 \ , \tag{2}$$

givet at $X_{n-1} = \omega_{n-1}$. Den betingede sandsynlighed $p(X_n | X_{n-1} = \omega_{n-1})$ kaldes i denne sammenhæng for overgangssandsynligheden. Hvis vi samtidig specificere sandsynlighedsmassefunktionen

$q(X_0)$ for det første index $n = 0$ - sandsynligheden for det initielle element, så kan vi opskrive sandsynlighedsmassefunktionen for en sekvens baseret på en Markov kæde model som

$$p(X_0, X_1, \ldots, X_N) = p(X_N|X_{N-1}) \cdots p(X_2|X_1)p(X_1|X_0)q(X_0) \tag{3}$$

eller skrevet lidt mere kompakt ved hjælp af produkt symbolet

$$p(X_0, X_1, \ldots, X_N) = q(X_0) \prod_{n=1}^{N} p(X_n|X_{n-1}) \ . \tag{4}$$

Bemærk at overgangssyndlighedsmassefunktionen $p(X_n|X_{n-1})$ er den samme for alle indeks $n > 0$ (denne egenskab kaldes for homogenitet og stationaritet).

Vi kan opskrive overgangssandsynlighederne som en $K \times K$ matrix $\mathbf{P} = \{P_{ij}\}$ hvor elementet på i'te række og j'te søjle er givet ved

$$P_{ij} = p(X = \omega^{(j)}|X = \omega^{(i)}) \ . \tag{5}$$

Vi dropper indekset n, da overgangsandsynlighederne jo ikke afhænger af n. Den i'te række i $\mathbf{P}$ svarer til sandsynlighedsmassefunktionen $p(X|X = \omega^{(i)})$ for det konkrete tegn $\omega^{(i)}$ og er normaliseret som angivet i (2). Vi kan ligeledes skrive den initielle sandsynlighedsmassefunktion $q(X_0)$ som en K- dimensional søjlevektor $\mathbf{v}_0 = (q(X_0 = \omega^{(1)}), \ldots, q(X_0 = \omega^{(K)}))$.

Vi kan ved hjælp af denne vektor-matrix formulering beregne sandsynlighedsmassefunktionen $\mathbf{v}_n = p(X_n)$ for tegnene til indeks n ved hjælp af matrix multiplikation

$$\mathbf{v}_n^T = \mathbf{v}_0^T \mathbf{P}^n \ , \tag{6}$$

hvor $\mathbf{P}^n$ angiver matrix multiplikation af $\mathbf{P}$ med sig selv n gange og hvor ^{T} betyder transponering. $\mathbf{v}_n = p(X_n)$ kaldes for marginal fordelingen af $p(X_0, X_1, \ldots, X_N)$ for indeks n.

En af de ting man er interesseret i når man arbejder med Markov kæder er hvad der sker med $\mathbf{v}_n$ når vi lader indekset n vokse. Hvis Markov kæden beskrevet ved $\mathbf{P}$ er en ergodisk homogen Markov kæde, en bestemt matematiske egenskaber som vi ikke skal komme nærmere ind på her (for yderligere information om Markov kæder anbefaler vi [1]), så gælder der at Markov kæden konvergere mod en stationær fordeling $\pi = (\pi(X = \omega^{(1)}), \ldots, \pi(X = \omega^{(K)}))$ (dvs. en fordeling uafhængig af indekset n)

$$\lim_{n \to \infty} \mathbf{v}_0^T \mathbf{P}^n = \pi^T \ . \tag{7}$$

Dvs. at efter et vist antal skridt n i kæden så forbliver fordelingen over tegnene i Ω den samme. Vi kan finde π ved enten at simulerer kæden i n skridt, men vi kan også finde π analytisk ved at beregne venstre egenvektoren $\mathbf{e}_1$ af $\mathbf{P}$ der hører til den største egenværdi λ_1 som vil have værdien $\lambda_1 = 1$. Venstre egenvektoren af $\mathbf{P}$ er defineret som $\mathbf{e}_1^T \mathbf{P} = \lambda_1 \mathbf{e}_1^T$. Vi kan beregne venstre egenvektorer ved at udnytte at $\mathbf{P}^T \mathbf{e}_1 = \lambda_1 \mathbf{e}_1$ og dvs. at vi blot skal finde egenværdier og -vektorer for den transponeret matrix $\mathbf{P}^T$ - eksempelvis ved at anvende `numpy.linalg.eig` i Python. Hvis vi normalisere $\mathbf{e}_1$ som (1) så gælder der at $\pi = \mathbf{e}_1$. Som en sidebemærkning kan nævnes at den næst største egenværdi af $\mathbf{P}$ giver et mål for konvergenshastigheden, dvs. hvor mange skridt n vi skal tage før kæden er konvergeret til π (for yderligere information om dette se [1]).

Dette resultat er under antagelse af at vores Markov kæde model opfylder de matematiske betingelser vi ikke vil nævne her. Vi kan i stedet undersøge om en konkret model konvergere mod $\pi = \mathbf{e}_1$ ved at simulerere kæden i n skridt og derpå estimere sandsynlighedsmassefunktionen $\mathbf{v}_n$ og se om den er tæt på $\mathbf{e}_1$.

1.1.1 Transformationsmetoden til sampling

For at kunne simulerer en Markov kæde model har vi behov for at kunne sample tilfældige elementer fra mængden Ω udfra sandsynlighedsmassefunktionerne $q(X_0)$ og $p(X_n|X_{n-1})$. Til det formål kan vi anvende den såkaldte transformationsmetode, som kan anvendes på diskrete mængder og sandsynlighedsmassefunktioner. I det efterfølgende vil vi for at forenkle forklaringen betragte en vilkårlig sandsynlighedsmassefunktion $p(X)$ på $X \in \Omega$. Husk på at for et fastlagt $X_{n-1} = \omega$ kan vi

betragte $p(X_n|X_{n-1} = \omega)$ som en sandsynlighedsmasse på linje med $q(X_0)$ og en vilkårlig anden $p(X)$. Vi kan derfor også anvende nedenstående metode til at sample fra $p(X_n|X_{n-1} = \omega)$.

Metoden går ud på at sample et uniformt fordelt tilfældig tal z mellem $[0, 1]$ og derpå benytte dette til et omvendt opslag i den akkumulerede sandsynlighedsfordeling for at finde det tilhørende element fra mængden Ω. Først skal vi beregne den akkumulerede sandsynlighedsfordeling $h(X)$ for sandsynlighedsmassefunktionen $p(X)$ og det gøres ved

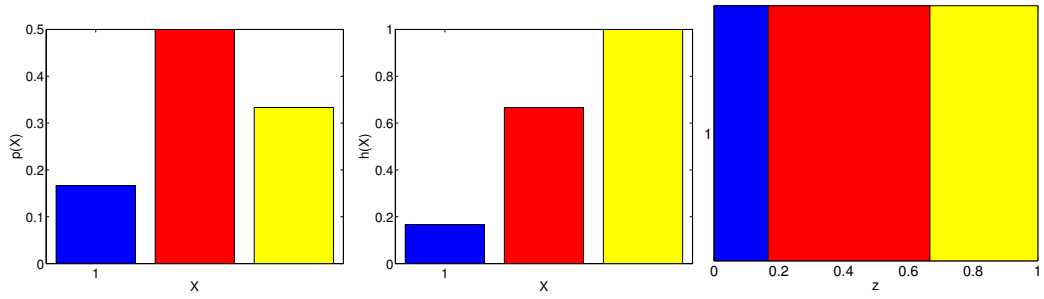
$$h(X = \omega^{(i)}) = \sum_{j=1}^{i} p(X = \omega^{(j)}) \ . \tag{8}$$

Dvs. at $h(X = \omega^{(i)})$ er summen af sandsynlighederne for alle elementer i Ω op til element i. Vi antager iøvrigt at elementerne i Ω har en orden således at ovenstående summation giver mening. [Implementationshint: Husk på at vi kan betragte $p(X)$ som en vektor $\mathbf{v}$ og vi kan derfor beregne $h(X)$ ved hjælp af funktionen `numpy.cumsum`.]

Algoritmen for at sample et tilfældigt udvalgt element $\omega \in \Omega$ udfra sandsynlighedsmassefunktionen $p(X)$ er som følger:

1. Udtræk et uniformt fordelt tilfældig tal $z \in [0, 1]$. [Implementationshint: Her kan du anvende Pythons indbyggede eller `numpy.random.rand` tilfældig tal generator.]
2. Find ved opslag i vektoren $H = (0, h(X = \omega^{(1)}), h(X = \omega^{(2)}), \ldots, , h(X = \omega^{(K-1)}))$ det mindste element indeks i således at $z >= H_i$.
3. Returner nu det samplet symbol givet ved indeks i, dvs. $\omega^{(i)}$.

Figur 1 viser en illustration af transformationsmetoden. Vi kan tænke på den akkumulerede fordeling $h(X)$ som en inddeling af intervallet $[0, 1]$ i delintervaller som hver svarer til et af elementerne i Ω. Længden af delintervallet er bestemt ved elementets sandsynlighed i følge $p(X)$. Opslaget i $h(X = \omega^{(i)})$ kan derfor tænkes som at finde det delinterval af $[0, 1]$ som z ligger indenfor og dette delinterval svarer til et konkret element indeks i.



Figur 1: Denne figur illustrere transformationsmetoden med et simpelt eksempel. Her har vores mængde $\Omega = \{\texttt{blå}, \texttt{rød}, \texttt{gul}\}$ tre elementer og vi har $p(X) = (1/6, 3/6, 2/6)$. Figuren illustrere både $p(X)$ og $h(X)$ som histogrammer. Sidste graf viser at vi kan benytte $h(X)$ til at inddele intervallet $[0, 1]$ i tre delintervaller, således at de hver har længden svarende til sandsynligheden for det tilhørende element. Algoritmen finder det delinterval hvor det tilfældige tal z falder indenfor og det samplede element svarer nu til det til delintervallet hørende element. Hvis $z = 0.4$, så lander vi i det røde interval og vi skal derfor returnere det røde element fra Ω.

Til tirsdag:

Læs om deterministiske endelige automata i uddrag fra Kapitel 3 i Stuart "Understanding computation", O'Reilly 2013 (spring afsnittet med titlen Simulation over).
Læs om grammatikker og syntakstræer i Kapitel 1.5, 2.2, 2.3 fra Sethi "Programming Languages"som findes på Absalon under `Undervisningsmateriale`.
Til forelæsningen gennemgår vi deterministiske tilstandsmaskiner og endelige automata, samt kontekstfri grammatikker og syntakstræer for beregningsudtryk.

Til torsdag:

Læs om regulæreudtryk på `https://docs.python.org/2/howto/regex.html#regex-howto` og dokumentationen for `re` modulet på `https://docs.python.org/2/library/re.html`.
Læs om Markov kæder på ugesedlen.
Til forelæsningen gennemgår vi regulæreudtryk og stokastiske tilstandsmaskiner med fokus på Markov kæder.

Bemærk: Der er ingen obligatorisk afleveringsopgave i denne uge.

Litteratur

[1] Pierre Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, 1999.

1.2 Tirsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden torsdag.

15ti1 Hvis du mangler at lave nogle af øvelsesopgaverne fra ugeseddel 14, så lav disse nu. Opgaverne 14ti1, 14ti2, 14to1, 14to2 og 14to3 er særlig relevante for den obligatoriske opgave fra ugeseddel 14.

15ti2 Skriv et program som indlæser en streng fra tastaturet og derpå optæller forekomsten (dvs. hyppigheden) af forskellige tegn i strengen. Udskriv tegnhyppighederne på tabelform til skærmen.

15ti3 Udvid dit program fra opgave 15ti2, således at tegnhyppighederne også bliver visualiseret som et grafisk histogram.

15ti4 Skriv et program som kan indlæse og udskrive til skærmen indholdet af tekstfilen som du finder i Absalon under `Undervisningsmateriale/Datamateriale/tekst-utf8.txt`. Teksten er gemt i unicode UTF-8 format og skal indlæses og repræsenteres i unicode format. Python 2.x indeholder en variant af datatypen `str` som er en unicode repræsentation af strenge kaldet `unicode`. Husk på at vi i Python kan lave unicode strengværdier ved at vedhæfte `u` foran strengen, eksempelvis som `u"Dette er en unicode strengværdi"`. Python 2.x indeholder et standard modul kaldet `codecs` som indeholder funktionalitet til at læse og skrive unicode tekstfiler (se `https://docs.python.org/2/library/codecs.html`). Du kan åbne tekstfilen med `fh = codecs.open("tekst-utf8.txt", encoding="utf-8")` hvorpå du kan læse fra file handle `fh` som du plejer. Bemærk at i `unicode` typen repræsenteres ikke-amerikanske tegn med en byte hexadecimal kode således at `\xe6` refererer til tegnet æ etc. Du kan konvertere en `unicode` streng til en standard `str` streng ved at anvende enten `codecs.encode` funktionen eller `unicode.encode` metoden. Eksempelvis således `ustr.encode('latin_1')` (under antagelse af at `ustr` indeholder en streng af typen `unicode`).

1.3 Torsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst ved udløbet af denne uge.

15to1 Hvis du ikke blev færdig med øvelsesopgaverne i tirsdags, så lav de resterende nu.

15to2 Skriv et program som simulerer et trafiklys. Repræsentér de tre tilstande Rød, Gul, Grøn som en deterministisk tilstandsmaskine. Programmet kan skifte tilstande ved input fra tastatureret.

15to3 Skriv et program som simulerer en deterministisk endelig automaton maskine. Maskinenen skal kunne indlæse en streng og skifte tilstande efter hvilket tegn den læser. Prøv eksempelvis at lade maskinen håndtere tegnene `a`, `b` og `c`. Du bestemmer selv hvorledes maskinen er

opbygget, dvs. hvor mange tilstande og hvilke tilstandsovergange maskinen har. Afprøv din automaton med forskellige strenge bestående af tegnene `a`, `b` og `c` og se hvordan maskinen skifter tilstande. Kan du opbygge en automaton som tæller (eller acceptere) forekomster af delstrenge `abc`?

15to4 Skriv et program som ved hjælp af regulæreudtryk og Python modulet `re` indlæser en streng bestående af ord og derpå opdeler strengen i ord ved at finde mellemrumstegn og dele strengen ved disse.

15to5 Skriv et program som indlæser en streng bestående af vilkårlige tegn og ved hjælp af regulæreudtryk og Python modulet `re` finder alle forekomster af tegn fra en delmængde af tegn, eksempelvis {`a,b,c`}, i strengen og ignorere alle andre tegn.

15to6 Angiv en kontekstfri grammatik, som beskriver syntaksen for en sekvens af cifre og bogstaver, der starter med et bogstav.

15to7 Tegn syntakstræer for følgende udtryk med hjælp af grammatikken for udtryk, som beskrevet i Sethi eller forelæsningsslides. Oversæt udtrykkene til omvendt polsk (postfix) notation, og beskriv hvordan det sidste af dem kan evalueres ved hjælp af en stak.

$$x + y * 2 \tag{9}$$

$$x * (y - z) + x * x \tag{10}$$

$$2 * 3 + 4 * (5 + 7)/2 \tag{11}$$

15to8 Skriv et program som implementere en stak baseret lommeregner som kan evaluere regneudtryk i omvendt polsk (postfix) notation som beskrevet i Sethi afsnit 2.3 eller ved forelæsningerne. Du kan anvende Python lister til at implementere stakken og de to operationer `push` og `pop`.

15to9 Skriv et program som kan simulerer en Markov kæde model med to tilstande repræsenteret ved tegnene {`0,1`}. Kædens initielle sandsynlighedsmassefunktion over tilstandene er $q(X_0 = 0) = 0.5$ og $q(X_0 = 1) = 0.5$, og overgangssandsynlighederne kan på tabelform opskrives som

	0	1
0	0.25	0.75
1	0.75	0.25

og på matrixform

$$\mathbf{P} = \{p(X = \omega^{(j)}|X = \omega^{(i)})\} = \begin{pmatrix} 0.25 & 0.75 \\ 0.75 & 0.25 \end{pmatrix} . \tag{12}$$

Tegn tilstandsdiagrammet på papir. Dit program skal kunne simulerere Markov kæden ved først at sample et tilfældig tegn fra mængden {`0,1`} udfra sandsynlighedsmassefunktionen $q(X_0)$ ved at anvende transformationsmetoden beskrevet i afsnit 1.1.1. De efterfølgende tegn skal samples ud fra $p(X = \omega^{(j)}|X = \omega^{(i)})$ givet det forrige tegn $\omega^{(i)}$.