Hidden Markov Models
Viterbi State Estimation

M. Sami Fadali
UNR

Outline

- Markov processes and Markov chains.
- Hidden Markov models.
- Application to state estimation.
- State estimation as minimum path problem.
- Viterbi algorithm.

Graph G

- G=(V,E)
- V= set of vertices (nodes, states) \{a,b,c, \ldots\}
- E= set of edges (branches) \{(a,b),(a,c),\ldots\}
- Path from node a to node b: list of adjacent branches connecting node a to node b.
- Directed graph: travel along edges is one-directional
- Weighted graph: weight (cost) assigned to each edge.

Markov Process

- Markov property: Future value independent of the past values given the present value.
- Continuous time or discrete time.

\[ P(x(t+h) = x_i \mid x(s) = x_j, s \leq t) = P(x(t+h) = x_i \mid x(t) = x_j), \forall h > 0 \]
Markov Chain

- Finite or countably infinite set of discrete states.
- Random transition between states at discrete instants.
- Probability of transition at time $k$ depends on state at time $k$ only (not on history of states)
- **Notation**: backwards in many books ($i$ to $j$).

$$ P(x(k)=i|x(k-1)=j,\ldots,x(m)=j_m)=P(x(k)=i|x(k-1)=j) = p_{ij} $$

Transition Probabilities

- Satisfy the following properties
  $$ p_{ij} = P(x(k)=i|x(k-1)=j) $$
  $$ p_{ij} \in [0,1] $$
  $$ \sum_{i=0}^{\infty} p_{ij} = 1 $$

Example
$$ 0.1 + 0.9 = 0.7 + 0.3 = 0.6 + 0.4 = 1 $$

Markov Chain Example

- Represent Markov Chain as a graph.
- States: nodes numbered $1$–$4$.
- Edges: weights are the transition probabilities.

Transition Matrix

- Matrix of transition probabilities
  $$ A = [P(x(k)=i|x(k-1)=j)] $$

Example
$$ A = \begin{bmatrix}
0.1 & 0.7 & 1 & 0 \\
0.3 & 0 & 0 & 0.4 \\
0.6 & 0 & 0 & 0.6 \\
0 & 0.3 & 0 & 0 \\
\end{bmatrix} $$
Recursion

\[
P(x(k) = i) = \sum_{j=1}^{n} P(x(k) = i \mid x(k-1) = j)P(x(k-1) = j)
\]

\[
= [p_{i1} \ p_{i2} \ \cdots \ p_{in}]
\]

- Initial probabilities for each state as a vector

\[
b = p_x(0) = \begin{bmatrix} P(x(0) = 1) \\ P(x(0) = 2) \\ \vdots \\ P(x(0) = n) \end{bmatrix}
\]

State–space Markov Model

- State equation governing probability evolution.
- State vector = vector of probabilities not states.

\[
p_{ij} = P(x(k) = i \mid x(k-1) = j)
\]

\[
P(x(k) = i) = \sum_{j=1}^{n} P(x(k) = i \mid x(k-1) = j)P(x(k-1) = j)
\]

\[
p_x(k) = Ap_x(k-1) + b\delta(k)
\]

\[
b = p_x(0) \quad A = [P(x(k) = i \mid x(k-1) = j)]
\]

Hidden Markov Model

- Markov model with state hidden.
- State indirectly observed through measurement \(y(k)\)

\[
P(y(k) = i) = \sum_{j=1}^{S} P(y(k) = i \mid x(k) = j)P(x(k) = j)
\]

\[
p_y(k) = C p_x(k)
\]

\[
C = [P(y(k) = i \mid x(k) = j)]
\]

State–space HMM

- State equation governing probability evolution.
- State vector = vector of probabilities not states.

\[
p_x(k) = Ap_x(k-1) + b\delta(k)
\]

\[
b = p_x(0) \quad A = [P(x(k) = i \mid x(k-1) = j)]
\]

\[
p_y(k) = C p_x(k)
\]

\[
C = [P(y(k) = i \mid x(k) = j)]
\]
Hidden Markov Modeling

- Use observations to determine the most likely sequence of states.
- Recast problem as a “minimum path” problem.

HMM Analysis

- Determine the most likely state sequence.
- Given HMM $M$ and measurements

\[ y = \{ y(k), k = 1, \ldots, T \} \]

\[ P(y | x, M) = \prod_{i=1}^{T} P(y(i) | x(i)) \]

\[ P(x | M) = P(x(1)) \prod_{i=2}^{T} P(x(i) | x(i-1)) \]

\[ P(y, x | M) = P(y | x, M) P(x | M) \]

Minimum Path Problem

- Logs often simplify problem.
- Branch cost is a log-likelihood.

\[ P(y, x | M) = P(y | x, M) P(x | M) \]

\[ x^* = \arg \max_x P(y, x | M) \]

\[ = \arg \min_x \{- \ln[P(y, x | M)]\} \]

HMM Training

- Use experimental or simulation data to determine transition probabilities.

\[ a(i, j) = \frac{\text{no. of transitions from state } j \text{ to state } i}{\text{no. of transitions from state } j} \]

\[ = P(x = i | x = j) \]

\[ c(i, j) = \frac{\text{no. of outputs } y \text{ in state } j}{\text{no. of outputs from state } j} \]

\[ = P(y = y, | x = j) \]
Several optimization problems can be recast as finding “shortest path” through a graph.

- **Dynamic programming**: compute backwards to find the optimum (shortest) path.
- **Reverse Dynamic Programming**: work forward to find the shortest path.

### Viterbi Algorithm

1. For each vertex $v_i$ at time $k$
   a) Find the path of each path = cost of the survivor path to each vertex $v_j$ at time $k-1$ + cost of branch $(v_i, v_j)$.
   b) Find the survivor path = path of least cost.
   c) Store the survivor path and the cost of each vertex $v_i$ at time $k$.
2. Increment $k$ and repeat.

### Viterbi Algorithm

- Store: time index ($k$), survivor terminating in state $x_k (x_{0,k})$, survivor length $|x_{0,k}|$, for all states $x_k$, $k = 1, \ldots, M$.
- Initialization: $k = 0$, $x_{0,k} = x_0$, $|x_{0,k}| = 0$.
- Recursion: Compute the cost and find the survivor and its length then increment time.

\[
|x_{0-(k+1)}| = \min_{x_k}\left\{|x_{0-k}| + |x_{k-(k+1)}|\right\}
\]

\[
x_{0-(k+1)} = \arg\min_{x_k}\left\{|x_{0-k}| + |x_{k-(k+1)}|\right\}
\]

\[k = k + 1\]
Optimization

\[ k = 0 \quad k = 1 \quad k = 2 \quad k = 3 \quad k = 4 \]

\[
\begin{array}{c}
0 \\
4 \\
3 \\
10 \\
3 \\
0 \\
x = x_1 \\
x = x_2 \\
x = x_3
\end{array}
\]

Cost(3 at 2) = min(8, 10) = 8

References