Analysis of different methods to approach and solve Hidden Markov Models.

By: Prayag Jain (170001037)
Yasasvi V Peruvemba (170002061)

Contents

Hidden Markov Model

Definition

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states.

Hidden Markov models are especially known for their application in reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

A Motivating Example

Given :

	U_1	U_2	U_3
U_1	0.1	0.4	0.5
U_2	0.6	0.2	0.2
U_3	0.3	0.4	0.3

and

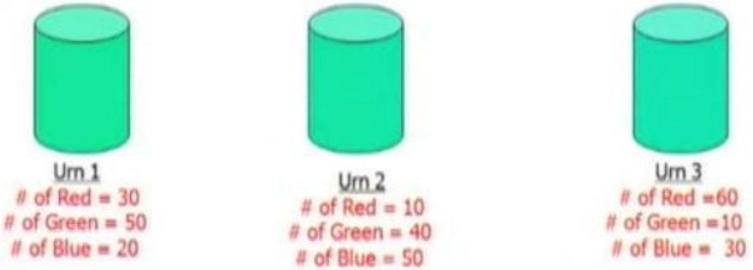
	R	G	B
U_1	0.3	0.5	0.2
U_2	0.1	0.4	0.5
U_3	0.6	0.1	0.3

Observation : RRGGBRGR

State Sequence : ??

Not so Easily Computable.

Colored Ball choosing



Probability of transition to another Urn after picking a ball:

	U_1	U_2	U_3
U_1	0.1	0.4	0.5
U_2	0.6	0.2	0.2
U_3	0.3	0.4	0.3

Forward & Backward Algorithm

Definition

The **forward–backward algorithm** is an inference algorithm for hidden Markov models which computes the posterior marginals of all hidden state variables given a sequence of observations/emissions $o_{1:T} := o_1, \dots, o_T$, i.e. it computes, for all hidden state variables $X_t \in \{X_1, \dots, X_T\}$, the distribution $P(X_t \mid o_{1:T})$. This inference task is usually called *smoothing*. The algorithm makes use of the principle of dynamic programming to efficiently compute the values that are required to obtain the posterior marginal distributions in two passes. The first pass goes forward in time while the second goes backward in time; hence the name *forward–backward algorithm*.

Pseudocode

```
Backward(guessState, sequenceIndex):
    if sequenceIndex is past the end of the sequence, return 1
    if (guessState, sequenceIndex) has been seen before, return saved result
    result = 0
    for each neighboring state n:
        result = result + (transition probability from guessState to
                           n given observation element at sequenceIndex)
                        * Backward(n, sequenceIndex+1)
    save result for (guessState, sequenceIndex)
    return result
```

[Forward Backward C++ Implementation]()

Time Complexity

The brute-force procedure for the solution of this problem is the generation of all possible N^T state sequences and calculating the joint probability of each state sequence with the observed series of events. This approach has time complexity $O(T \cdot N^T)$, where T is the length of sequences and N is the number of symbols in the state alphabet. This is intractable for realistic problems, as the number of possible hidden node sequences typically is extremely high. However, the forward–backward algorithm has time complexity $O(N^2 T)$.

Viterbi Algorithm

Definition

The **Viterbi algorithm** is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the **Viterbi path**—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.

Pseudocode

```
function VITERBI(O, S, Π, Y, A, B) : X
    for each state j ∈ {1, 2, ..., K} do
        T1[j,1] ← π_j·B_{jy_1}
        T2[j,1] ← 0
    end for
    for each observation i = 2, 3, ..., T do
        for each state j ∈ {1, 2, ..., K} do
            T1[j,i] ← max_k (T1[k,i-1] · A_kj · B_{jy_i})
            T2[j,i] ← argmax_k (T1[k,i-1] · A_kj)
        end for
    end for
    z_T ← argmax_k (T1[k,T])
    x_T ← s_{z_T}
    for i ← T,T-1,...,2 do
        z_{i-1} ← T2[z_i,i]
        x_{i-1} ← s_{z_{i-1}}
    end for
    return X
end function
```

[Viterbi C++ Implementation]

Time Complexity

The complexity of this implementation is $O(T \times |S|^2)$. A better estimation exists if the maximum in the internal loop is instead found by iterating only over states that directly link to the current state (i.e. there is an edge from k to j). Then using amortized analysis one can show that the complexity is $O(T \times (|S| + |E|))$, where E is the number of edges in the graph.

CarpeDiem Algorithm

Definition

In order to determine the end point of the best path to a given layer, one can avoid inspecting all vertices in that layer. In particular, after sorting the vertices in layer t according to their vertical weight, the search can be stopped when the difference in vertical weight of the best node so far and the next vertex in the ordering is big enough to counterbalance any advantage that can be possibly derived from exploiting a better transition and/or a better ancestor

Lets Define

S^{1*} : *an upper bound to the maximal transition weight in the current graph*

$$S^{1*} \geq \max_{y_t, y_{t-1}} S^1_{y_t, y_{t-1}}; \tag{3}$$

γ_t^* : *the reward of the best path to any vertex in layer t (including the vertical weight of the ending vertex)*

$$\gamma_t^* = \max_{y_t} \gamma(y_t);$$

β_t : *an upper bound to the reward that can be obtained in reaching layer t* $(2 \leq t \leq T)$

$$\beta_t = \gamma_{t-1}^* + S^{1*}; \tag{4}$$

$\sqsupseteq_t$: *a total ordering—based on vertical weights—of vertices at layer t.*

$$\sqsupseteq_t \equiv \{(y_t, y'_t) | S^0_{y_t} \geqslant S^0_{y'_t}\}. \tag{5}$$

Also, we say that vertex y_t is more promising *than vertex* y'_t *iff* $y_t \sqsupseteq_t y'_t$.

Pseudocode

begin
  foreach y_1 { Initialization Step } **do**
2   $\mathbb{G}(y_1) \leftarrow S^0_{y_1}$; { Opens vertex y_1 }
  end
3 $y^*_1 \leftarrow \arg\max_{y_1}(\mathbb{G}(y_1))$;
  $\mathbb{B}_2 \leftarrow \mathbb{G}(y^*_1) + S^{1*}$;
  foreach *layer* $t \in 2 \ldots T$ **do**
    $y^*_t \leftarrow$ result of Algorithm 3 on layer t;
  end
  return path to y^*_T;
end

Algorithm 2: `CarpeDiem`.

begin
  $y^*_t \leftarrow$ most promising vertex;
  $y'_t \leftarrow$ next vertex in the $\sqsupseteq_t$ ordering;
  Open vertex y^*_t {call Algorithm 4};
  while $\mathbb{G}(y^*_t) < \mathbb{B}_t + S^0_{y'_t}$ **do**
4   Open vertex y'_t {call Algorithm 4};
5   $y^*_t \leftarrow \arg\max_{y'' \in \{y^*_t, y'_t\}} [\mathbb{G}(y'')]$;
    $y'_t \leftarrow$ next vertex in the $\sqsupseteq_t$ ordering;
  end
6 $\mathbb{B}_{t+1} \leftarrow \mathbb{G}(y^*_t) + S^{1*}$;
  return y^*_t;
end

Algorithm 3: Forward search strate

Data: A vertex y_t to be opened
begin
  $y^*_{t-1} \leftarrow$ most promising vertex;
  $y'_{t-1} \leftarrow$ next vertex in the $\sqsupseteq_{t-1}$ ordering;
  while y'_{t-1} *is open* **do**
    $y^*_{t-1} \leftarrow \arg\max_{y'' \in \{y'_{t-1}, y^*_{t-1}\}} \left[\mathbb{G}(y'') + S^1_{y_t, y''}\right]$;
    $y'_{t-1} \leftarrow$ next vertex in the $\sqsupseteq_{t-1}$ ordering;
  end
  while $\left(\mathbb{G}(y^*_{t-1}) + S^1_{y_t, y^*_{t-1}} < \mathbb{B}_{t-1} + S^0_{y'_{t-1}} + S^{1*}\right)$ **do**
    Open y'_{t-1} {call Algorithm 4};
    $y^*_{t-1} \leftarrow \arg\max_{y'' \in \{y'_{t-1}, y^*_{t-1}\}} \left[\mathbb{G}(y'') + S^1_{y_t, y''}\right]$;
    $y'_{t-1} \leftarrow$ next vertex in the $\sqsupseteq_{t-1}$ ordering;
  end
7 $\mathbb{G}(y_t) \leftarrow \mathbb{G}(y^*_{t-1}) + S^1_{y_t, y^*_{t-1}} + S^0_{y_t}$;
end

Algorithm 4: Backward search strategy to open y_t.

Time Complexity

The best case happens when horizontal rewards, being equal for each transition, do not provide any discriminative power. In such a case the right hand side of the inequality in Formula 6 is zero and the inequality is guaranteed to be satisfied immediately. Moreover, being the backward strategy based on a bound similar to the one that leads to Formula 6, it will never open any other vertex. Then, a single vertex per layer is opened and `CarpeDiem` has order of $O(T + TK\log(K)) = O(TK\log(K))$ time complexity. A more formal argument about `CarpeDiem` complexity is stated by Theorem 3 and proved in Appendix B.

`CarpeDiem` *has* $O(TK^2)$ *worst case time complexity and* $O(TK\log K)$ *best case time complexity.*

Thank You!!