Project Report

On

Analysis of Different Methods to Approach and Solve Hidden Markov Models

Submitted by

Prayag Jain
Yasasvi V Peruvemba

Computer Science and Engineering

2nd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2019

Introduction

To understand, implement and analyze the algorithms used to effectively solve Hidden Markov Models. 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

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.

Algorithm Analysis

Viterbi Algorithm : 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.

Complexity Analysis :

The Time complexity of this algorithm is

$O(NT^2)$

And the space complexity is

$O(T^2 + kT + NT)$

K is the total number of unique observable states.

T is the total number of unique hidden states.

N is the length of the observed sequence.

Forward-Backward Algorithm : The forward–backward algorithm is an algorithm for hidden Markov models which computes the posterior marginal probabilities of all hidden state variables for a set of observed sequence of events.

Complexity Analysis :

z $\in$ Hidden States in a layer

m is the set of Hidden States

n is the length of observed sequence

- $\alpha_1(z_1)=p(z_1,x_1)=p(z_1)p(x_1|z_1)$
- If each z has m states then computational complexity is
  - $\Theta(m)$ for each z_k for one k
  - $\Theta(m^2)$ for each k
  - $\Theta(nm^2)$ in total

CarpeDiem Algorithm : 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 a layer 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 from a better ancestor.

Complexity Analysis :

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 + T K log(K)) = O(T K log(K))
time complexity.
A more formal argument about CarpeDiem complexity is stated by
CarpeDiem has **O(T K^2)** worst case time complexity and
O(T K log(K)) best case time complexity.

Algorithm Design

Viterbi Algorithm Pseudo Code

```
function VITERBI (O, S, Π, Y, A, B) : X
    for each state j ∈ {1, 2, ..., K} do
        T1[j,1] ← πj·Bjy1
        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] · Akj · Bjyi)
            T2[j, i] ← arg max_k (T1[k, i − 1] · Akj)
        end for
    end for
    zT ← arg max_k (T1[k, T])
    xT ← s_zT
    for i ← T,T-1,...,2 do
        zi-1 ← T2[zi,i]
        xi-1 ← s_zi-1
    end for
    return X
end function
```

Forward Algorithm Pseudo Code

init $t = 0$, transition probabilities $p(x_t|x_{t-1})$, emission probabilities, $p(y_j|x_i)$, observed sequence, $y(1:t)$

```
for t = t + 1
    α_t(x_t) = p(y_t|x_t) Σ_{x_{t-1}} p(x_t|x_{t-1}) α_{t-1}(x_{t-1}).
until t=T
```

return $p(y(1:t)) = \alpha_T$

Backward Algorithm Pseudo Code

```
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
```

CarpeDiem Algorithm Pseudo Code

begin

    foreach y_1 { Initialization Step } **do**

        $\mathbb{G}(y_1) \leftarrow S^0_{y_1}$; { Opens vertex y_1 }

    end

    $y_1^* \leftarrow \arg\max_{y_1}(\mathbb{G}(y_1))$;

    $\mathbb{B}_2 \leftarrow \mathbb{G}(y_1^*) + S^{1*}$;

    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 strategy.

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.

Implementation

Viterbi C++ Implementation

Forward Backward C++ Implementation

Github Link

Conclusion :

This project has been instrumental in understanding Hidden Markov Models and their immense contributions towards the advancements in technologies like Speech Synthesis, Speech tagging, Bioinformatics and Cryptanalysis to name a few. We have also analysed different approaches to solve these class of Hidden Markov Model problems via various algorithms and finally a slightly optimised algorithm.

The Viterbi algorithm uses a dynamic programming approach to determine the hidden state sequence from a given set of observed sequence.

The Forward-Backward Algorithm also uses a dynamic approach, but to calculate the state probabilities, for further operations.

The CarpeDiem algorithm is an optimization on the Viterbi algorithm which works on a dynamic approach, but eliminating a few corner case calculation making it significantly more efficient than the pre-existing Viterbi algorithm.

Hence, we have analysed these three algorithms as the most prominent way to solve Hidden Markov Models.

References

1. https://en.wikipedia.org/wiki/Viterbi_algorithm

2. https://en.wikipedia.org/wiki/Forward%E2%80%93backward_algorithm

3. www.jmlr.org/papers/volume10/esposito09a/esposito09a.pdf

4. https://en.wikipedia.org/wiki/Hidden_Markov_model