viterbi

December 19, 2024

1 Julia Implementation of Viterbi Algorithm

[1]:
```julia
function Viterbi(states, init, trans, emit, obs)
    # Input:
    # states: Array of hidden states (e.g., [1, 2, ..., S])
    # init: Array of initial probabilities for each state
    # trans: S × S transition matrix (transition probabilities between states)
    # emit: S × O emission matrix (probabilities of observations given states)
    # obs: Array of T observations (sequence of observations)

    T = length(obs)  # Number of observations
    S = length(states)  # Number of states

    # Initialize matrices
    prob = zeros(T, S)  # T × S matrix for probabilities
    prev = fill(NaN, T, S)  # T × S matrix for storing previous states

    # Initialization step (t = 0)
    for s in 1:S
        prob[1, s] = init[s] * emit[s, obs[1]]  # Calculate initial␣
↪probabilities
    end

    # Recursion step (t = 1 to T-1)
    for t in 2:T
        for s in 1:S
            for r in 1:S
                new_prob = prob[t - 1, r] * trans[r, s] * emit[s, obs[t]]
                if new_prob > prob[t, s]
                    prob[t, s] = new_prob
                    prev[t, s] = r  # Store the state r that maximized the␣
↪probability
                end
            end
        end
    end
```

```
    # Backtracking to find the most probable path
    path = Vector{Int}(undef, T)  # Array to store the path
    path[T] = argmax(prob[T, :])  # Find the state with the highest probability␣
↪at the last time step

    for t in (T - 1):-1:1
        path[t] = prev[t + 1, path[t + 1]]  # Follow the back-pointers to␣
↪reconstruct the path
    end

    return path
end
```

[1]: Viterbi (generic function with 1 method)

1.1 Explanation

1. **Inputs**:
   - `states`: Hidden states (e.g., $S_1, S_2, \ldots, S_S$).
   - `init`: Initial probabilities $P(S_i)$ for each state.
   - `trans`: Transition probabilities $P(S_j|S_i)$ between states.
   - `emit`: Emission probabilities $P(O_k|S_i)$ of observations O_k given states.
   - `obs`: Sequence of observations.
2. **Initialization**:
   - At $t = 0$, calculate the initial probabilities using $init[s] \cdot emit[s, obs[1]]$.
3. **Recursion**:
   - For $t = 1$ to $T - 1$, compute probabilities for each state s by considering all possible previous states r.
   - Track the state r that maximized the probability.
4. **Backtracking**:
   - Start from the state with the maximum probability at the final time step.
   - Follow the `prev` matrix to reconstruct the most probable sequence of states.
5. **Output**:
   - Returns the most probable sequence of hidden states.

[2]:
```
states = [1, 2, 3]  # Hidden states
init = [0.5, 0.2, 0.3]  # Initial probabilities
trans = [0.7 0.2 0.1; 0.1 0.6 0.3; 0.3 0.3 0.4]  # Transition probabilities
emit = [0.9 0.1; 0.2 0.8; 0.1 0.9]  # Emission probabilities
obs = [1, 2, 1]  # Observation sequence (1-based indexing)

# Run the Viterbi algorithm
most_probable_path = Viterbi(states, init, trans, emit, obs)
println("Most probable path: ", most_probable_path)
```

```
Most probable path: [1, 1, 1]
```

1.2 Output

For the example above, the algorithm will return the most probable sequence of states corresponding to the observation sequence.

2 References

Viterbi_algorithm

[ ]:
```

```