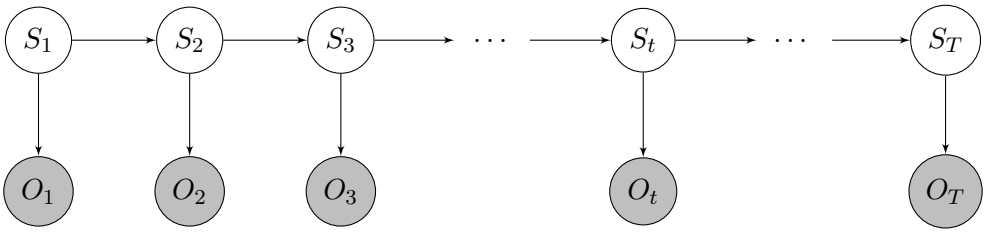
Viterbi Algorithm

Eric Rice
ericjrice95@gmail.com

Context

In this problem, you will decode an English sentence from a long sequence of non-text observations. To do so, you will implement the same basic algorithm used in most engines for automatic speech recognition. In a speech recognizer, these observations would be derived from real-valued measurements of acoustic waveforms. Here, for simplicity, the observations only take on binary values, but the high-level concepts are the same.

Consider a discrete Hidden Markov Model with $n = 27$ hidden states $S_t \in \{1, 2, \ldots, 27\}$ and binary observations $O_t \in \{0, 1\}$, where $t = 1, 2, \ldots, T$.



You will be given ASCII data files which contain parameter values for the initial state distribution $\pi_i = P(S_1 = i)$, the transition matrix $a_{ij} = P(S_{t+1} = j | S_t = i)$ (for any t), and the emission matrix $b_i(k) = P(O_t = k | S_t = i)$ (for any t), as well as a long bit sequence of $T = 325000$ observations. Use the Viterbi algorithm to compute the most probable sequence of hidden states conditioned on this particular sequence of observations.

To check your answer: suppose that the hidden states $\{1, 2, \ldots, 26\}$ represent letters $\{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$ of the English alphabet, and suppose that hidden state 27 encodes a space between words. If you have implemented the Viterbi algorithm correctly, the most probable sequence of hidden states (ignoring repeated elements) will reveal a highly recognizable message, as well as an interesting commentary on our times.

Theory

The most probable sequence of hidden states given our observations is

$$S^* = \{S_1^*, S_2^*, \ldots, S_T^*\} = \arg\max_{S_1,\ldots,S_T} P(S_1, \ldots, S_T | O_1, \ldots, O_T).$$

Via a simple manipulation using the product rule, we can write the above in terms of the joint probability of our random variables:

$$\begin{aligned} S^* &= \arg\max_{S_1,\ldots,S_T} P(S_1, \ldots, S_T | O_1, \ldots, O_T) \\ &= \arg\max_{S_1,\ldots,S_T} \left[\frac{P(S_1, \ldots, S_T, O_1, \ldots, O_T)}{P(O_1, \ldots, O_T)} \right] \\ &= \arg\max_{S_1,\ldots,S_T} P(S_1, \ldots, S_T, O_1, \ldots, O_T) \end{aligned}$$

With this in mind, we define auxiliary values ℓ_{it}^* as follows. Here, a logarithm is taken for its nice properties but also to avoid underflow.

$$\begin{aligned} \ell_{i1}^* &= \log\left[P(S_1 = i, O_1)\right] = \log\left[\pi_i b_i(O_1)\right] && (t = 1) \\ \ell_{it}^* &= \max_{S_1,\ldots,S_{t-1}} \log\left[P(S_1, \ldots, S_{t-1}, S_t = i, O_1, \ldots, O_t)\right] && (t > 1) \end{aligned}$$

We now use the product rule as well as conditional independence implied by our belief network in order to derive a recurrence relation which we may use to compute ℓ_{it}.

$$\begin{aligned} \ell_{j,t+1}^* &= \max_{S_1,\ldots,S_{t-1},i} \log\left[P(S_1, \ldots, S_{t-1}, S_t = i, S_{t+1} = j, O_1, \ldots, O_{t+1})\right] \\ &= \max_{S_1,\ldots,S_{t-1},i} \big\{\log\left[P(S_1, \ldots, S_{t-1}, S_t = i, O_1, \ldots, O_t)\right] \\ &\qquad + \log\left[P(S_{t+1} = j | S_1, \ldots, S_{t-1}, S_t = i, O_1, \ldots, O_t)\right] \\ &\qquad + \log\left[P(O_{t+1} | S_1, \ldots, S_{t-1}, S_t = i, S_{t+1} = j, O_1, \ldots, O_t)\right]\big\} \\ &= \max_{S_1,\ldots,S_{t-1},i} \big\{\log\left[P(S_1, \ldots, S_{t-1}, S_t = i, O_1, \ldots, O_t)\right] \\ &\qquad + \log\left[P(S_{t+1} = j | S_t = i)\right] + \log\left[P(O_{t+1} | S_{t+1} = j\right]\big\} \\ &= \max_i \left[\ell_{it}^* + \log(a_{ij})\right] + \log(b_j(O_{t+1})) \end{aligned}$$

Notice that $S_T^* = \arg\max_i(\ell_{iT}^*)$. During computation of these values, it is useful to record the most likely state transitions $\Phi_{t+1}(j) = \arg\max_i \left[\ell_{it}^* + \log(a_{ij})\right]$. From here, we can recover S^* via backtracking:

$$S_t^* = \Phi_{t+1}(S_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1$$

Solution

(a) **Source code:** *(Written in Python)*

```
import numpy as np
import string
from matplotlib import pyplot as plt
from numba import jit


@jit(nopython=True)
def viterbi(o, a, b, pi):
    """Computes the most likely sequence of hidden states S given
    data o and parameters a, b, pi.  S = S* in the theory."""
    T = len(o)
    n = len(a)

    ell = np.zeros(shape=(n, T))
    for i in range(0, n):
        ell[i, 0] = np.log(pi[i] * b[i, o[0]])

    phi = np.zeros(shape=(T, n), dtype=np.int32)
    for t in range(0, T - 1):
        for j in range(0, n):
            to_be_maxed = np.zeros(n)
            for i in range(0, n):
                to_be_maxed[i] = ell[i, t] + np.log(a[i, j])
            phi[t + 1, j] = np.argmax(to_be_maxed)
            ell[j, t + 1] = to_be_maxed[phi[t + 1, j]]\
                + np.log(b[j, o[t + 1]])

    S = np.zeros(T, dtype=np.int32)
    S[-1] = np.argmax(ell[:, -1])
    for t in range(2, T + 1):
        S[-t] = phi[-t + 1, S[-t + 1]]

    return S


if __name__ == "__main__":
    o = np.loadtxt('observations.txt', dtype=np.int32)
    a = np.loadtxt('transitionMatrix.txt', dtype=np.float64)
    b = np.loadtxt('emissionMatrix.txt', dtype=np.float64)
    pi = np.loadtxt('initialStateDistribution.txt', dtype=np.float64)
```

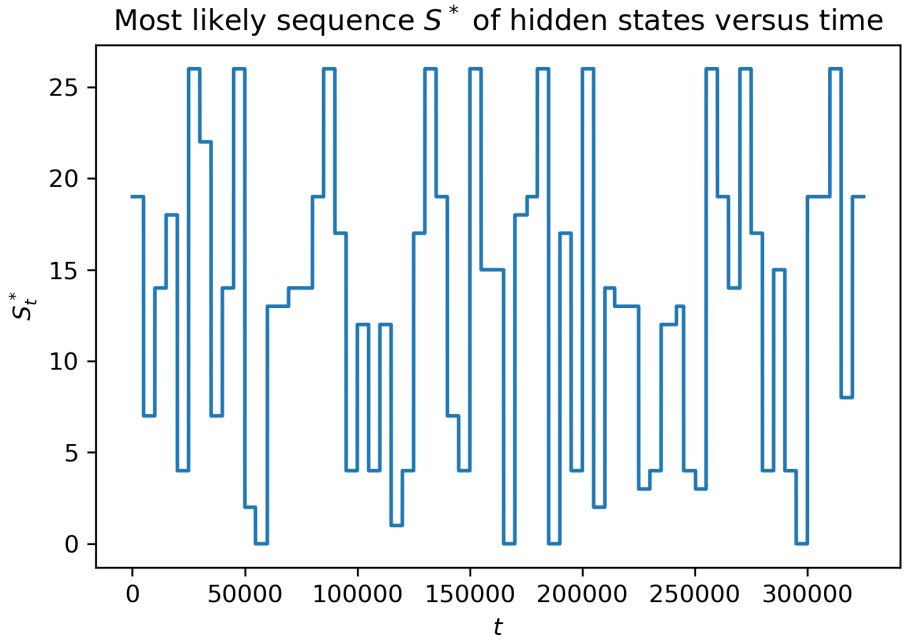
```
# Viterbi path
S = viterbi(o, a, b, pi)

# Checking answer
alphabet = string.ascii_uppercase + ' '
message = ''
for t in range(0, len(S)):
    if t == 0:
        message += alphabet[S[t]]
    elif S[t] != S[t - 1]:
        message += alphabet[S[t]]

print(message)

# Plotting S versus t
t = np.array(range(0, len(S)))
plt.plot(t, S)
plt.xlabel('$t$')
plt.ylabel('$S_t$')
plt.title('Most likely sequence $S^*$ of hidden states versus time')
plt.show()
```

(b) **Results and the hidden message**



The hidden message is: "THOSE WHO CANOT REMEMBER THE PAST ARE CONDEMNED TO REPEAT IT."