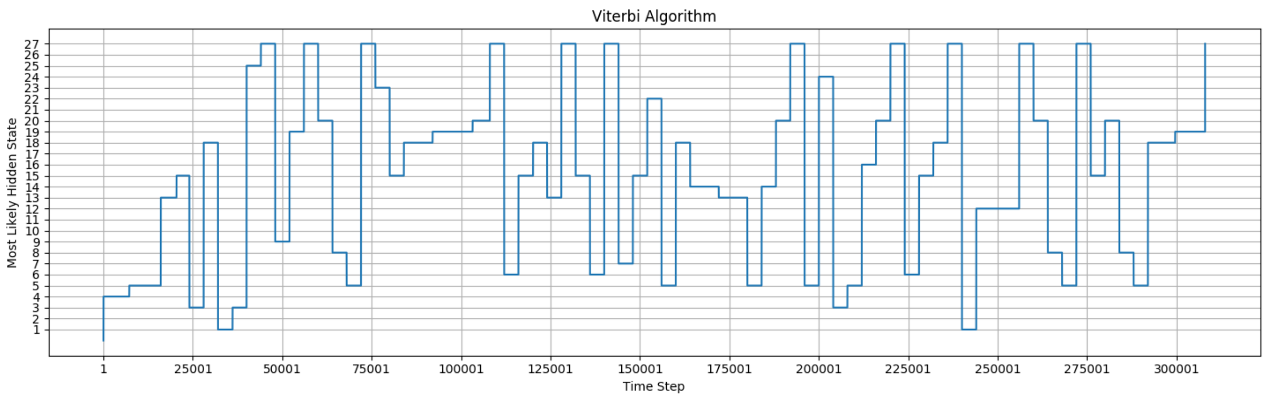
Elliot Pourmand
A14731789

6.1 Viterbi Algorithm

a) Source code attached at end

b)

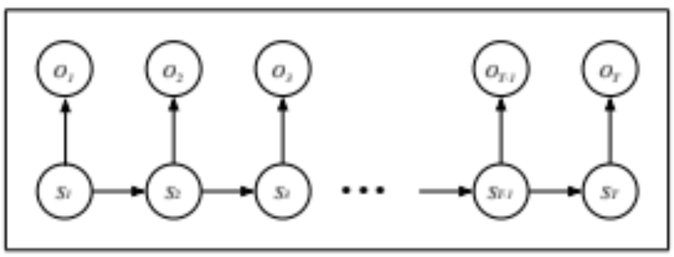


decompressed: democracy is the worst form of government except for all the others

6.2 Conditional independence

Consider the hidden Markov model (HMM) shown below, with hidden states S_t and observations O_t for times $t \in \{1, 2, \ldots, T\}$. Indicate whether the following statements are true or false.

false	$P(S_t \mid S_{t-1}) = P(S_t \mid S_{t-1}, O_t)$
true	$P(S_t \mid S_{t-1}) = P(S_t \mid S_{t-1}, O_{t-1})$
false	$P(S_t \mid S_{t-1}) = P(S_t \mid S_{t-1}, S_{t+1})$
false	$P(S_t \mid O_{t-1}) = P(S_t \mid O_1, O_2, \ldots, O_{t-1})$
true	$P(O_t \mid S_{t-1}) = P(O_t \mid S_{t-1}, O_{t-1})$
false	$P(O_t \mid O_{t-1}) = P(O_t \mid O_1, O_2, \ldots, O_{t-1})$
false	$P(O_1, O_2, \ldots, O_T) = \prod_{t=1}^{T} P(O_t \mid O_1, \ldots, O_{t-1})$
true	$P(S_2, S_3, \ldots, S_T \mid S_1) = \prod_{t=2}^{T} P(S_t \mid S_{t-1})$
true	$P(S_1, S_2, \ldots, S_{T-1} \mid S_T) = \prod_{t=1}^{T-1} P(S_t \mid S_{t+1})$
false	$P(S_1, S_2, \ldots, S_T \mid O_1, O_2, \ldots, O_T) = \prod_{t=1}^{T} P(S_t \mid O_t)$
false	$P(S_1, S_2, \ldots, S_T, O_1, O_2, \ldots, O_T) = \prod_{t=1}^{T} P(S_t, O_t)$
true	$P(O_1, O_2, \ldots, O_T \mid S_1, S_2, \ldots, S_T) = \prod_{t=1}^{T} P(O_t \mid S_t)$



6.3 More conditional independence

Indicate the **smallest** subset of evidence nodes that must be considered to compute each conditional probability shown below. The first two problems are done as examples. (You may assume everywhere that $2 < t < T - 1$, i.e., do not worry about special boundary cases.)

(a) (Optional)

$$P(S_t|S_1, S_2, \ldots, S_{t-1}) = P(S_t|S_{t-1})$$

$$P(O_t|S_1, S_2, \ldots, S_T) = P(O_t|S_t)$$

$$P(S_t|S_{t+1}, S_{t+2}, \ldots, S_T) = P(S_t|S_{t+1})$$

$$P(S_t|O_t, O_{t-1}, O_{t+1}) = P(S_t|O_t, O_{t-1}, O_{t+1})$$

$$P(S_t|O_t, O_{t+1}, \ldots, O_T) = P(S_t|O_t, O_{t+1}, \ldots, O_T)$$

$$P(O_t|O_1, O_2, \ldots, O_{t-1}) = P(O_t|O_1, O_2, \ldots, O_{t-1})$$

$$P(O_t|S_{t-2}, S_{t-1}, S_{t+1}, S_{t+2}) = P(O_t|S_{t-1}, S_{t+1})$$

$$P(O_t|O_{t-1}, O_{t+1}, S_1, S_T) = P(O_t|O_{t-1}, O_{t+1}, S_1, S_T)$$

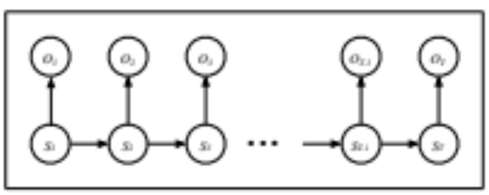
(b) **(NOT optional – will be graded)**

$$P(S_t|O_t, O_{t-1}, O_{t+1}, S_{t-1}, S_{t+1}) = P(S_t|O_t, S_{t-1}, S_{t+1})$$

$$P(S_t|S_1, S_T, O_1, O_t, O_T) = P(S_t|O_t, S_1, S_T)$$

$$P(O_t|O_1, O_2, \ldots, O_{t-1}, S_{t-1}) = P(O_t|S_{t-1})$$

$$P(O_t|O_1, O_2, \ldots, O_{t-1}, S_{t-2}) = P(O_t|O_{t-1}, S_{t-2})$$



6.4

a) $$P(Y_1 | X_1) = \frac{P(Y_1, X_1)}{P(X_1)}$$

$$= \frac{\sum_{x} P(Y_1, X_1, X_0 = x_0)}{P(X_1)}$$

$$= \frac{\sum_{x_0} P(Y_1 | X_1, X_0 = x_0) P(X_1, X_0 = x_0)}{P(X_1)}$$

$$\frac{\sum_{x_0} P(Y_1 | X_1, X_0 = x_0) \cancel{P(X_1)} P(X_0 = x_0)}{\cancel{P(X_1)}}$$

$$\boxed{\sum_{x_0} P(Y_1 | X_1, X_0 = x_0) P(X_0 = x_0)}$$

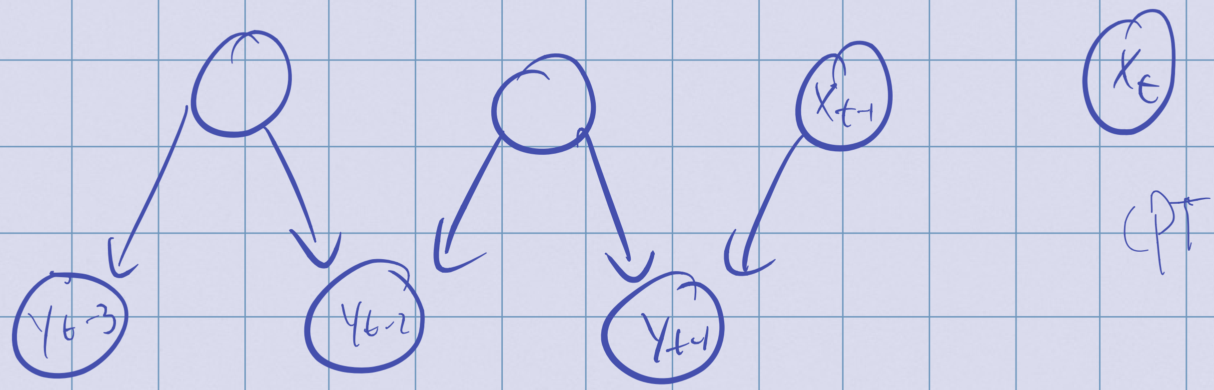
b) $$P(Y_1) = \sum_{x_0, x_1} P(Y_1, X_0 = x_0, X_1 = x_1)$$

$$= \sum_{x_0, x_1} P(Y_1 | X_0 = x_0, X_1 = x_1) P(X_0 = x_0, X_1 = x_1)$$

$$\boxed{= \sum_{x_0, x_1} P(Y_1 | X_0 = x_0, X_1 = x_1) P(X_0 = x_0) P(X_1 = x_1)}$$

c) $P(X_t \mid Y_1, Y_2, \ldots, Y_{t-1})$

Notice X_t is d-seperated from $Y_1, Y_2 \ldots Y_{t-1}$



thus $P(X_t \mid Y_1, Y_2 \ldots Y_{t-1}) \boxed{= P(X_t)}$

d) $P(Y_t \mid X_t, Y_1, \ldots, Y_{t-1})$

$$X_0 \to Y_1 \leftarrow X_1 \to Y_2 \leftarrow X_2$$

marginalize $\sum_{x_{t-1}} P(Y_t, X_{t-1} = x_{t-1} \mid X_t, Y_1, \ldots Y_{t-1})$

*product Rule

*recursion

$$\sum_{x_{t-1}} P(Y_t \mid X_{t-1} = x_{t-1}, X_t, Y_1, \ldots Y_{t-1}) P(X_{t-1} = x_{t-1} \mid X_t, \ldots Y_1, \ldots Y_{t-1})$$

*cond ind.

$$\boxed{\sum_{x_{t-1}} P(Y_t \mid X_{t-1} = x_{t-1}, X_t) P(X_{t-1} = x_{t-1} \mid Y_1, \ldots Y_{t-1})}$$

e) $P(Y_t \mid Y_1, Y_2, \ldots, Y_{t-1})$

$$\sum_{x_t, x_{t-1}} P(Y_t, X_t = x_t, X_{t-1} = x_{t-1} \mid Y_1, Y_2, \ldots, Y_{t-1})$$ * marginalize

* product rule

$$\sum_{x_t, x_{t-1}} P(Y_t, X_t = x_t \mid X_{t-1} = x_{t-1}, Y_1, Y_2 \ldots Y_{t-1}) P(X_{t-1} = x_{t-1} \mid Y_1, Y_2 \ldots Y_{t-1})$$

* product rule

$$\sum_{x_t, x_{t-1}} P(Y_t \mid X_t = x_t, X_{t-1} = x_{t-1}, \cancel{Y_1, \ldots, Y_{t-1}}) P(X_t = x_t \mid Y_1, \ldots Y_{t-1}) P(X_{t-1} = x_{t-1} \mid Y_1, Y_2, \ldots Y_{t-1})$$

* cond. ind. | part c) | * recursion

$$X_0 \to Y_1 \leftarrow X_1 \to Y_2 \leftarrow X_2$$

$$= \boxed{\sum_{x_t, x_{t-1}} P(Y_t \mid X_t = x_t, X_{t-1} = x_{t-1}) P(X_t = x_t) P(X_{t-1} = x_{t-1} \mid Y_1, Y_2 \ldots Y_{t-1})}$$

```
import numpy


def fetch_data():
  f = open('observations.txt')
  observations = f.readlines()
  f.close()

  f = open('initialStateDistribution.txt')
  initial_state_distributions = f.readlines()
  f.close()

  f = open('transitionMatrix.txt')
  transition_matrix = f.readlines()
  f.close()

  f = open('emissionMatrix.txt')
  emission_matrix = f.readlines()
  f.close()

  observations = list(map(lambda obs: int(obs), observations[0].split()))
  initial_state_distributions = list(map(lambda i: float(i), initial_state_distributions))
  for i in range(len(transition_matrix)):
     row = transition_matrix[i].split()
     row = list(map(lambda r: float(r), row))
     transition_matrix[i] = row
  emission_matrix = list(map(lambda e: list([float(e.split('\t')[0]),
float(e.split('\t')[1])]), emission_matrix))
  return initial_state_distributions, transition_matrix, emission_matrix,
observations


def viterbi():
  import math
  # Create two n x T tables
```

```
    initial_state_distributions, transition_matrix, emission_matrix, observations = fetch_data()
    print(len(emission_matrix), len(emission_matrix[0]))
    N_STATES = len(initial_state_distributions)
    N_TIME_STEPS = int(len(observations)/1)
    value_table = [[0 for j in range(N_TIME_STEPS)] for i in range(N_STATES)]
    phi_table = [[0 for j in range(N_TIME_STEPS)] for i in range(N_STATES)]
    print(len(value_table[0]))

    # Set initial t = 0 state
    for i in range(len(initial_state_distributions)):
        value_table[i][0] = math.log(initial_state_distributions[i] * emission_matrix[i][observations[0]])

    # Mark phi table at T = 0 to -1
    for i in range(N_STATES):
        phi_table[i][0] = -1



    # Recursively compute value table
    for t in range(1, N_TIME_STEPS):
        for j in range(N_STATES):
            max_e = -9999999
            arg_max = -1
            for i in range(N_STATES):
                if value_table[i][t-1] + math.log(transition_matrix[i][j]) > max_e:
                    max_e = value_table[i][t-1] + math.log(transition_matrix[i][j])
                    arg_max = i
            value_table[j][t] = max_e + math.log(emission_matrix[j][observations[t]])
            phi_table[j][t] = arg_max

    max_pi_T = -999
    arg_max = -1
    for i in range(N_STATES):
        if value_table[i][N_TIME_STEPS-1] > max_pi_T:
            max_pi_T = value_table[i][N_TIME_STEPS-1]
            arg_max = i
```

```python
    message = []
    for t in range(N_TIME_STEPS-2, -1, -1):
        print(f"path at t: {t} ", phi_table[arg_max][t])
        arg_max = phi_table[arg_max][t]
        message.append(arg_max)

    message.reverse()

    import matplotlib.pyplot as plt
    message = message[:-1]
#  for i in range(len(message)):
#      message[i] += 1
    plt.plot(list(range(N_TIME_STEPS-2)), message)
    plt.grid(True)
    plt.ylabel('Most Likely Hidden State')
    plt.xlabel('Time Step')
    plt.title('Viterbi Algorithm')
    plt.yticks(numpy.arange(1, 28, 1))
    plt.xticks(numpy.arange(1, N_TIME_STEPS-2, 25000))
    plt.show()
    compressed = []
    current_char = message[0]
    for i in range(1, len(message)):
        if message[i] != current_char:
            compressed.append(message[i])
            current_char = message[i]
    compressed = compressed[:-1]
    from string import ascii_lowercase
    ascii_lower = dict(zip(range(26), ascii_lowercase))
    for i in compressed:
        if i == 26:
            print(' ')
        else:
            print(ascii_lower[i])

viterbi()
```