Sequence Learning

- British Left Waffles on Falkland Islands
  - (N, N, V, P, N, N)
  - (N, V, N, P, N, N)

- Segmentation
  - (b, i, b, i, b, i, b, i, b, i, b, i, b, i, b, i, b, i)

  China’s 14 open border cities marked economic achievements
Sequence Learning

3 states: N, V, P
Observation sequence: \( o_1, \ldots, o_n \)
State sequence (6+1): \( \text{Start, N, N, V, P, N, N} \)

Finite State Machines

Mealy Machine
Finite State Machines

Moore Machine

Probabilistic FSMs

- Start at a state $i$ with a start state probability: $\pi_i$
- Transition from state $i$ to state $j$ is associated with a transition probability: $a_{ij}$
- Emission of symbol $o$ from state $i$ is associated with an emission probability: $b_i(o)$
- Two conditions:
  - All outgoing transition arcs from a state must sum to 1
  - All symbol emissions from a state must sum to 1
Probabilistic FSMs

\[ \sum_i \pi_i = 1 \]
\[ \pi_A = \frac{1}{2} \]
\[ \pi_N = \frac{1}{2} \]

Emission

\[ b_A(\text{killer}) = 0 \]
\[ b_A(\text{crazy}) = 1 \]
\[ b_A(\text{clown}) = 0 \]
\[ b_A(\text{problem}) = 0 \]

\[ b_N(\text{killer}) = \frac{1}{3} \]
\[ b_N(\text{crazy}) = 0 \]
\[ b_N(\text{clown}) = \frac{1}{3} \]
\[ b_N(\text{problem}) = \frac{1}{3} \]

Transition

\[ a_{AA} = \frac{1}{3} \]
\[ a_{AN} = \frac{2}{3} \]
\[ a_{NN} = \frac{9}{10} \]
\[ a_{NA} = \frac{1}{10} \]

3/19/12
Hidden Markov Models

- There are \( n \) states \( s_1, \ldots, s_i, \ldots, s_n \)
- The emissions are observed (input data)
- Observation sequence \( O = (o_1, \ldots, o_t, \ldots, o_T) \)
- The states are not directly observed (hidden)
- Data does not directly tell us which state \( X_t \) is linked with observation \( o_t \)
  \( X_t \in \{s_1, \ldots, s_n\} \)

Markov Chains vs. HMMs

- For observation sequence \( babaa \)
  \( i.e: \ o_1=b, \ o_2=a, \ldots, \ o_5=a \)
- Compute \( P(babaa) \) using a bigram model
  \( P(b)*P(alb)*P(bla)*P(alb)*P(ala) \)
- Equivalent Markov chain:
Markov Chains vs. HMMs

• For observation sequence babaa
  \( i.e: o_1=b, o_2=a, \ldots, o_5=a \)

• Compute \( P(babaa) \) using a trigram model
  \[ P(ba)*P(b|ba)*P(a|ab)*P(a|ba) \]

• Equivalent Markov chain:
Markov Chains vs. HMMs

- Given an observation sequence
  \( O = (o_1, \ldots, o_t, \ldots, o_T) \)
- An \( n \)th order Markov Chain or \( n \)-gram model computes the probability
  \( P(o_1, \ldots, o_t, \ldots, o_T) \)
- An HMM computes the probability
  \( P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T) \) where the state sequence is *hidden*

Properties of HMMs

- Markov assumption
  \[ P(X_t = s_i | \ldots, X_{t-1} = s_j) \]
- Stationary distribution
  \[ P(X_t = s_i | X_{t-1} = s_j) = P(X_{t+l} = s_i | X_{t+l-1} = s_j) \]
HMM Algorithms

- HMM as language model: compute probability of given observation sequence
- HMM as parser: compute the best sequence of states for a given observation sequence
- HMM as learner: given a set of observation sequences, learn its distribution, i.e. learn the transition and emission probabilities

\[ P(o_1, \ldots, o_T) = \sum_{X_1, \ldots, X_{T+1}} P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T) \]
HMM Algorithms

• HMM as parser: compute the best sequence of states for a given observation sequence
• Compute best path $X_1, \ldots, X_{T+1}$ from the probability $P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T)$

Best state sequence $X^*_1, \ldots, X^*_{T+1}$

$$= \arg \max_{X_1, \ldots, X_{T+1}} P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T)$$

Best Path (Viterbi) Algorithm

• Key Idea 1: storing just the best path doesn’t work
• Key Idea 2: store the best path upto each state
Viterbi Algorithm

function viterbi (edges, input, obs): returns best path
edges = transition probability
input = emission probability
T = length of obs, the observation sequence
num-states = number of states in the HMM
Create a path-matrix: viterbi[num-states+1, T+1] # init to all 0s
for each state s: viterbi[s, 0] = π[s]
for each time step t from 0 to T:
  for each state s from 0 to num-states:
    for each s’ where edges[s,s’] is a transition probability:
      new-score = viterbi[s,t] * edges[s,s’] * input[s’,obs[t]]
      if (viterbi[s’,t+1] == 0) or (new-score > viterbi[s’, t+1]):
        viterbi[s’, t+1] = new-score
        back-pointer[s’,t+1] = s

# finding the best path
best-final-score = best-final-state = 0
for each state s from 0 to num-states:
  if (viterbi[s,T+1] > best-final-score):
    best-final-state = s
    best-final-score = viterbi[s,T+1]
# start with the last state in the sequence
x = best-final-state
state-sequence.push(x)
for t from T+1 downto 0:
  state-sequence.push(back-pointer[x,t])
  x = back-pointer[x,t]
return state-sequence