Introduction to Algorithms and Data Structures

Lecture 20: Probabilistic FSMs and the Viterbi Algorithm

Mary Cryan

School of Informatics
University of Edinburgh

Today's lecture

Two aims for today:

- To remind ourselves about Finite State Machines, and to introduce Probabilistic FSMs.
- To give a dynamic programming algorithm (the Viterbi algorithm) which computes the most likely route through a probabilistic FSM/HMM, for a given output string.

We will be working with transducer-like FSMs, where the FSM outputs a character or signal at every state (according to some probability).

Finite State Machines

In Inf1 we covered (deterministic) Finite State Machines, where we have an alphabet Σ, a set of states Q, a distinguished start state q_0, and subset $F \subseteq Q$ of *accepting states*, and a *transition function* $\delta : Q \times \Sigma \to Q$. For any (deterministic) FSM $M = \langle Q, \Sigma, q_0, F, \delta \rangle$, we can test a string $s \in \Sigma^*$ against the FSM:

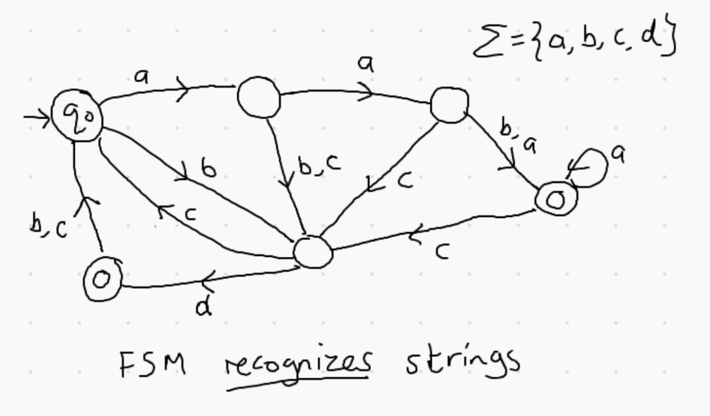
- Each string $s \in \Sigma^*$ has at most one computation path.

  The computation is *accepting* if and only if it ends in a state from F.
- If the computation for s gets stuck at some intermediate state on a deterministic FSM (no outgoing transition to consume the next character), then s is rejected.
- The language accepted by M is denoted $L(M)$.

  A language L that has some FSM that recognises (exactly) L is called a *regular* language.

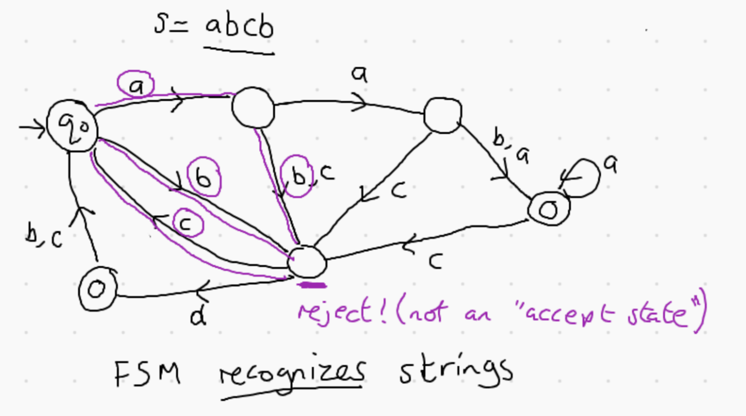
A non-deterministic FSM is a FSM where we have a *transition relation* $\Delta : Q \times \Sigma \times Q$ (so "$\Delta(q, a)$" may be a *subset* of Q rather than a single state).

Example FSM



The letters on the transitions/arrows are used to direct the path of a test string *s* ("next character" read from *s* gets matched to arrow with that label).

Example FSM



Testing the string $s = abcb$ has a computation path ending at a reject state (the accept states are the double-circles). So *abcb* does not belong to the language of this FSM.

Probabilistic Finite State Machines

A Probabilistic FSM is a finite-state machine of the form $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ with $\Delta \subseteq (Q \times \Sigma \times Q)$, together with a *probability label* $p_{q,a,q'} \in [0,1]$ for every $(q, a, q') \in \Delta$ such that for every $q \in Q, a \in \Sigma$, we have

$$\sum_{q' \in Q, (q,a,q') \in \Delta} p_{q,a,q'} = 1.$$

We are no longer "free" to choose our path like in non-deterministic FSMs, now there are probabilities of taking particular paths.

Strings are no longer in/out of the language, a string $s \in \Sigma^*$ has a specific *probability* of being accepted.

"between deterministic and non-deterministic?"

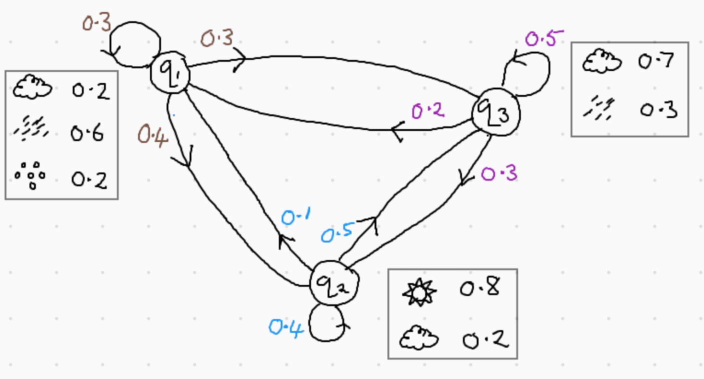
(in terms of choosing a computational path)

Hidden Markov Models (HMMs)

More general again is the Hidden Markov Model (HMM)

- We are in the "transducer" world - our job is no longer *test/accept* strings; instead we generate strings, outputting character/observations as we move round the HMM.
- The output of characters happens at the states.
- The transitions are now simple (q, q') pairs (no longer character-specific).
  For every $q \in Q$, every $(q, q') \in \Delta$ we have $p_{q,q'}$ such that for every $q \in Q$, $\sum_{q',(q,q')\in\Delta} p_{q,q'} = 1$.
- Every state q of the HMM will generate a character/observation from Σ according to some probability distribution on Σ (distribution is specific to the state).

Example: "Weather" HMM



This is a Hidden Markov Model for modelling weather sequences (rain one day, cloud the next, ...). This generates sequences of weather observations.

Hidden Markov Models (HMMs)

Definition

A *Hidden Markov Model (HMM)* is a graph/state-machine $M = \langle Q, \Sigma, \Delta, P, \{b_q : q \in Q\}, \pi \rangle$ with $\Delta \subseteq (Q \times Q)$, together with

- A *transition matrix* $P \in \mathbb{R}_+$ defining a *probability label* $p_{q,q'} \in [0, 1]$ for every $(q, q') \in \Delta$ such that for every $q \in Q$, our "next state" distribution satisfies

  $$\sum_{q' \in Q} p_{q,q'} = 1,$$

  and together with

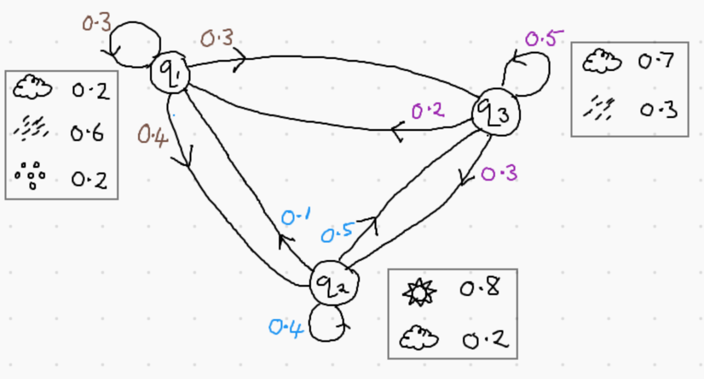
- A probability distribution b_q on Σ for every $q \in Q$, b_q defining the distribution of "emissions" from Σ associated with state q.

  We will require $\sum_{a \in \Sigma} b_q(a) = 1$ for every $q \in Q$.

  (we must emit one character every time we visit q)

- We sometimes have an extra probability distribution π to describe the "start state" distribution on states of Q (often *uniform*).

Our "weather" HMM

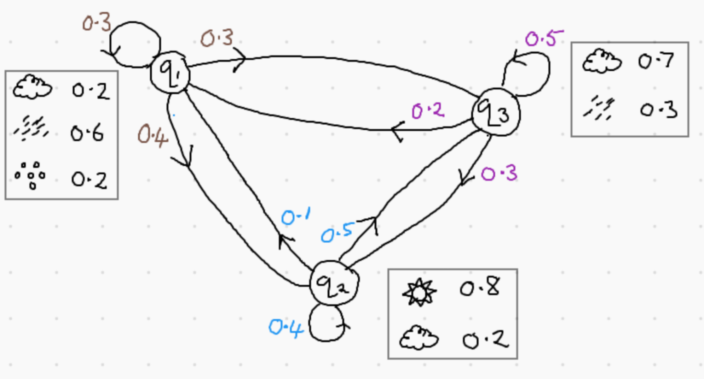


State set Q is $\{q1, q2, q3\}$.

The $p_{q,q'}$ values are the probabilities on the transitions (eg $p_{q1,q2}$ is 0.4).

The distribution b_{q1} has $b_{q1}(cloud) = 0.2$ $b_{q1}(rain) = 0.6$ and $b_{q1}(hail) = 0.2$.

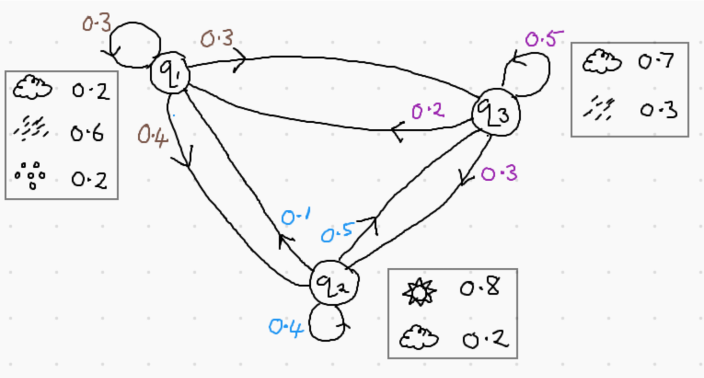
Our "weather" HMM



The diagram doesn't show details of the "start state" distribution π.
We will assume all states are equally likely start states (1/3 each).

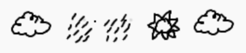
HMMs generate sequences of observations (different to FSMs, which test them)

The "max likelihood" question

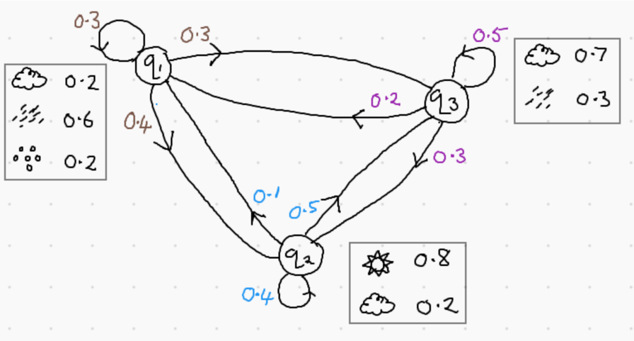


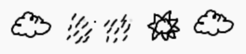
Our meteorology team have drawn-up this model to capture the patterns of weather in Scotland.

How well does it "fit" a sequence of observations? Say, for?



The "max likelihood" question





Path q1, q3, q2, q3, q2? . . . 0 (q3 has 0-probability for "sun")
Path q1, q1, q1, q2, q3? . . . $\frac{1}{3} \cdot 0.2 \cdot \mathbf{0.3} \cdot 0.6 \cdot \mathbf{0.3} \cdot 0.6 \cdot \mathbf{0.4} \cdot 0.8 \cdot \mathbf{0.5} \cdot 0.7$
Path q3, q3, q3, q2, q3? . . . $\frac{1}{3} \cdot 0.7 \cdot \mathbf{0.5} \cdot 0.3 \cdot \mathbf{0.5} \cdot 0.3 \cdot \mathbf{0.3} \cdot 0.8 \cdot \mathbf{0.5} \cdot 0.7$

3rd option best of the three shown (examining details).

Computing the "max likelihood" path

In general, we may have a HMM of arbitrary size, with arbitrary transition relation/matrix and arbitrary b_q distributions at the nodes/states:

> *Given a HMM defined by* $M = \langle Q, \Sigma, \Delta, P, \{b_q : q \in Q\}, \pi\rangle$*, and a sequence* $s \in \Sigma^*$*, what is the most likely path through* M *to have generated* s*?*

We gave three examples on the prior slide, but there are many more potential "routes through the HMM" we can try, for a sequence of 5 observations.

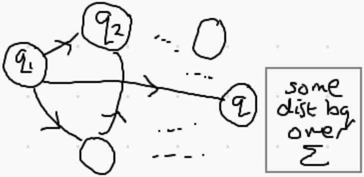
As the length of the sequence increases, the number of routes increases exponentially.

How can we get the route with highest probability (subject to the various parameter values of our HMM)?

USE DYNAMIC PROGRAMMING

Computing the "max likelihood" path

So we have some HMM M over alphabet Σ



We are given a sequence $s = s_1 \ldots s_m$ over Σ.

Want path through M most likely to have generated s.

(*) That path must have ended at some q of M.

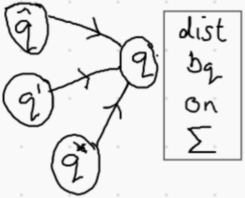
WHICH state?

So we are going to solve for the "most likely" path, for this given s, ending at state q ..... for every $q \in Q$.

(parametrising with q)

Computing the "max likelihood" path

We want the most likely path through M, ending at q, to have generated s.



process of generating s ends at q
$\Rightarrow s_m$ got "generated" by q.

So s_m was generated with conditional probability b_{q,s_m}

* $s_1 \dots s_{m-1}$ will have been generated first, and have ended at some q^* that has a transition $q^* \to q$

But WHICH ?

Computing the "max likelihood" path

In considering options for q^*, the "most likely" overall is the max-value of

$P_{q^* \to q} \times$ "value of most likely path for $s_1 \ldots s_{m-1}$ ending at q^*"

(max is taken over all q^* such that $(q^* \to q) \in \Delta$)

Our generalisation –

we solve "most-likely path (value) for every prefix $s_1 \ldots s_i$ of s, ending at state q (for every $q \in Q$)"

$mlp[i, q]$ = "this max"

Computing the "max likelihood" path

Dynamic programming view:

- Let $s = s_1 \ldots s_n$. The optimum path ended at *some* state $q \in Q$.
- Considering final state q, we must have arrived there via some incoming transition $(q* \to q)$ into q.
- When considering a hypothetical "final transition" $q* \to q$,
  - The cost of the *final step, then emission* is $p_{q*,q} \cdot b_{q,s_n}$.
  - The most likely path for string $s_1 \ldots s_{n-1}$ ending in state $q*$ is *another* "maximum likelihood" calculation (for a slightly shorter string).
- Our collection of subproblems is the "most likely path" question for $s_1 \ldots s_i$ ending at state q, for every $1 \leq i \leq n$, and for every $q \in Q$.

Our recurrence

We will write $mlp[i, q]$ to denote the cost of the most likely path of M to generate $s_1, \ldots, s_i$ which ends in $q \in Q$.

$$mlp[i, q] \;=\; \begin{cases} \pi_q \cdot b_{q,s_1} & i = 1 \\ \max_{q* \in Q}\{mlp[i-1, q*] \cdot p_{q*,q} \cdot b_{q,s_i}\} & i > 1 \end{cases}$$

(we don't explicitly check whether $(q*, q) \in \Delta$, however, we can assume that we have $p_{q*,q} = 0$ if this transition is not available)

We could also add an extra table called *prev* such that $prev[i, q]$ is the state $q*$ which optimizes the $mlp[i, q]$ value (with $prev[1, q] =' -'$ for all q).

Implementation:

- We will need tables `mlp`, `prev` of dimensions $n \times |Q|$ each, where n is the length of the given string/sequence.

  (exactly n, as we don't compute anything for an empty string)

Dynamic programming implementation

Algorithm Viterbi$(M = \langle Q, \Sigma, P, \{b_q : q \in Q\}, \pi\rangle, s = s_1 \ldots s_n)$

for $q \in Q$
    `mlp[1,q]` $\leftarrow \pi_q \cdot b_{q,s_1}$; `prev[1,q]` $\leftarrow$'-';
for $i = 2$ **to** n
    for $q \in Q$
        `mlp[i,q]` $\leftarrow 0$; `prev[1,q]` $\leftarrow$'-';
        for $q* \in Q$
            `trans` $\leftarrow p_{q*,q} \cdot b_{q,s_i}$
            if (`trans`$\times$`mlp[i-1,q*]`) $>$ `mlp[i,q]`
                `mlp[i,q]` $\leftarrow$(`trans`$\times$`mlp[i-1,q*]`)
                `prev[i,q]` $\leftarrow$`q*`
`max` $\leftarrow 0$
for $q \in Q$
    if `mlp[n,q]` $>$`max`
        `max` $\leftarrow$ `mlp[n,q]`
return `max`

Some Observations

Technical Observations:

- It's not hard to see from the algorithm (and the three nested **for** loops) that the running-time will be $\Theta(n \cdot |Q|^2)$ and the space used is of order $\Theta(n \cdot |Q|)$.
  - This is efficient/"polynomial-time" even if the Model is not "Finite State" (can be a general directed graph).
- If carrying out repeated multiplications of small probabilities, as this raises concerns about accuracy as the values get smaller.
  - In practice, many implementations instead will seek to maximize the $\log$ of the probability (of generating that sequence).
  - This has the advantage of also switching the multiplications into additions!
  - Values (the $\log$s of probabilities) will end up negative but "max" will still be the "max".

Reading and Working

Reading:

- The Viterbi algorithm is not included in the collection of Dynamic Programming problems studied in [CLRS]. However, it appears as problem 15.5 at the end of Chapter 15.
- HMMs are used to model many natural phenomena (for example, speech production, natural language modelling), and you will see them discussed in many courses within Informatics.