Hidden Markov Models

- Generative, rather than descriptive model.
  - Objects produced by random process.
- Dependencies in process, some random events influence others.
  - Time is most natural metaphor here.
- Simplest, most tractable model of dependencies is Markov.
- Lecture based on: **Rabiner**, "*A **Tutorial** on Hidden Markov Models and Selected Applications in Speech Recognition.*"

Markov Chain

- States: S_1, … S_N
- Discrete time steps, 1, 2, …
- State at time t is q_t.
- Initial state, q_1. $pi_i = P(q_1 = S_i)$.
- $P(q_t = S_j \mid q_{t-1} = S_i, q_{t-2} = S_k, \ldots)$

  $= P(q_t = S_j \mid q_{t-1} = S_i)$.

  This is what makes it *Markov.*
- Time independence:

  $a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i)$.

Examples

- 1D random walk in finite space.
- 1D curve generated by random walk in orientation.

States of Markov Chain

- Represent state at time t as vector:

$w(t) = (P(q_t=S_1), P(q_t=S_2), \dots P(q_t=S_N))$

- Put transitions, a_{ij} into matrix A.
  - A is *Stochastic,* meaning columns sum to 1.
- Then $w(t) = A^T * w(t-1)$.

Asymptotic behavior of Markov Chain

- $w(n) = A^T (A^T(\ldots(A^T(w(1))))) = A^{Tn}(w(1))$.
  - $w(n)$ will be leading eigenvector of A^T.
    - This means asymptotic behavior independent of initial conditions
    - Some special conditions required:
      - Reach every state from every state (ergodic).
      - Markov chain may not converge (periodic)

Hidden Markov Model

- Observations, $v_1, \ldots v_M$
  - We never know the state, but at each time step a state produces an observation.
- Observation distribution:

  $b_j(k) = P(v_k \text{ at } t | q_t = S_j)$.

  Note this is also taken to be time independent.
- Example, HMM that generates contours varying from smooth to rough.

Three problems

- Probability of observations given model.
  - Use to select model given observations (eg, speech recognition).
  - To refine estimate of HMM.
- Given model and observations, what were likely states?
  - States may have semantics (rough/smooth contour).
  - May provide intuitions about model.
  - However, this is often least useful problem.
- Find model to optimize probability of observations.
  - Learning the model.

Probability of Observations

- Solved with dynamic programming.
- *Whiteboard (see Rabiner for notes).*

Problem 1: Probability of observations given model: P(O | lambda).

Basic idea: we can do this with dynamic programming. This is basically inductive. Suppose we know the probability of producing the first t symbols and winding up in state i at time t, for all values of i. Then we want to use that to compute the same thing for t+1. The key thing is that to figure this out for time t+1 we just need to know if for time t. In particular, it won't matter what states we were in for time < t, just what states we were in at t.

Specifically, (abbreviate \alpha = \a) we define:

\a_t(i) = P(O_1, … O_t, q_t = S_i | \lambda)

1) Initialize: \a_1(i) = \pi_i b_i(O_1)
2) Recurse: \a_{t+1}(j) = [sum_{i=1}^N \a_t(i) a_{ij}] b_j(O_{t+1})
3) Termination: P(O|lambda) = sum_{i=1}^N \a_T(i)

Problem 2: Maximum likelihood sequence of internal states given model and observations. This is the same as (1), except we use maximum instead of sum, and keep backward pointers.

Problem 3: Estimating a model, given a sequence (or many of them). Formally: argmax_lambda P(O | lambda). Note that we are assuming we know the number of states (more states would always allow a better fit).

We approach this with an iterative algorithm. We assume some starting point, then improve it. Given a model, we estimate the sample probability of every parameter. Then we adjust the model to use each sample probability as the true one. For example, we estimate the probability that we begin in state 1, given the model AND the observations. Then we use this as the new prior probability that we will start in state 1. We do this for every aspect of the model.

The key to computing these sample probabilities is to figure out the probability that we will be in state i at time t, and the probability that we will move from state i at time t to state j at time t+1. It is important to note that we can't just use \a to determine this. This is because the probability that we will be in state i at time t doesn't just depend on the probability of emitting the first t symbols and winding up in state i after time t. It also depends on the chances that we will continue on from state i to emit all the rest of the symbols. For example, it is possible that from state i you almost never go to a state that will emit the next symbol you need.

So we define a similar quantity that says how likely we are to continue on a produce the rest of the symbols (using \B for \Beta):

\B_t(i) = P(O_{t+1}, … O_T | q_t = S_i, lambda)

This can be computed using dynamic programming in a way similar to \a.

1) Initialize: \B_T(i) = 1. This is because there are no further observations to account for.

2) Induction: \B_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1})\B_{t+1}(j)

That is, the probability that we'll produce all future observations if we're in state i at time t is the sum over all j's of the probability that we'll go to that j, produce the right next observation, and then produce all subsequent observations.

Using this, we can determine \g_t(i), the probability we're in state i at time t.

\g_t(i) = \a_t(i)\b_t(i)/P(O|lambda). Numerator is the probability we generate the observations while passing through state i at time t.

The denominator can be written by summing the expression in the numerator over all i.

Then we can determine the probability that we transition from one state to another at a particular time, given the observations:

\T_t(i,j) = \a_t(i) a_{ij} b_j(O_{t+1})\B_{t+1}(j)/P(O|lambda).

Using these, we can compute what we need. Take the sample value of a_{ij}. This is the sample probability we go from state i to state j. To do this, we can sum over all times, to find the expected number of times we go from state i to state j, and divide this by the expected number of times we are in state i. The initial distribution is just the expected number of times we are in state i at time 1. The sample distribution of b_j(k) is the expected number of times we're in state j in those times at which symbol k was observed, divided by the expected number of times we were in state k.

Note that we haven't proven that this iteration really improves the model, and that it converges, but these things are true, and kind of intuitive.

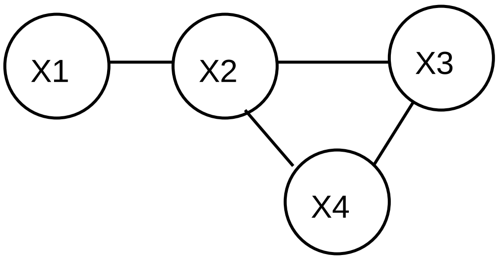
Practical Applications

- Continuous to discrete
  - Discretize observations with codebook.

Graphical Models

- Represent conditional dependencies with a graph.
- Each variable is a node
- Two nodes are connected if they are directly dependent.
- Two variables, X1 and X2, are conditionally independent, given knowledge of other variables, iff removing the nodes of the other variables disconnects X1 and X2.

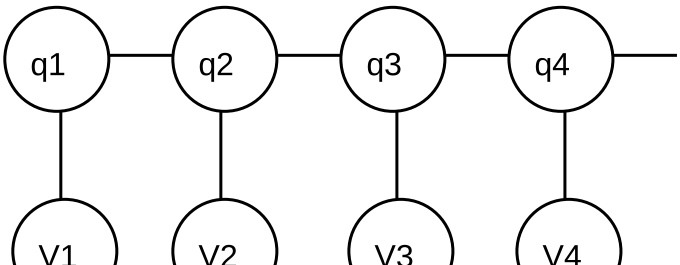
Example



This graph represents the relationship that given V2, V1 is conditionally independent of V3 and V4.

E.g., P(V1|V2,V3,V4) = P(V1|V2)

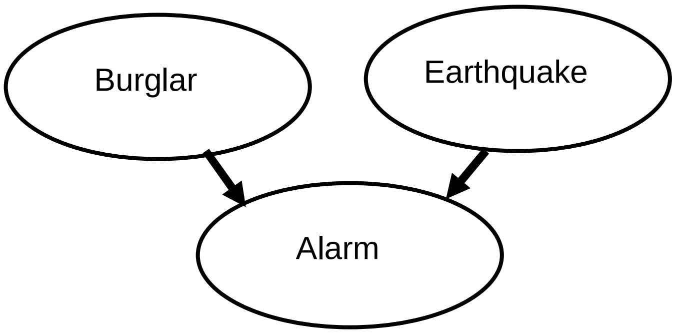
HMM



We perform inference with knowledge of conditional probabilities (the model) and of some variables (V1, V2…)

Belief Nets

- We add directions to edges, indicating causation.



In this classic example, the alarm might be set off by a burglar or an earthquake.

Knowing whether the alarm went off doesn’t make earthquake and burglar conditionally independent.

(e.g., if the alarm goes off, learning that there was an earthquake makes me much less worried about burglars).

Belief Propagation

- Like inference in an HMM, but can handle directed edges, and general DAGs.
- Still the basic idea is the same, to combine information from forward and backward directions (or maybe more than two independent directions).
- In general, lack of cycles allows us to use dynamic programming.

General Graphical models

- Model must include joint distribution of all variables that form cliques.
- We can compress a clique into a single variable, with states that give the cross-product of all states of individual variables.
  - If doing this produces a DAG, we can perform inference using dynamic programming.
  - Kfan is a graphical model with a single clique, and all other variables directly connected to this.
- Inference in general is NP-hard, but there is much work on effective algorithms for this problem.