Combining examples and prior knowledge in learning: summary

Without prior knowledge, typically do a kind of greedy search in hypothesis space:

- decide on a notion of goodness.
- decide on a notion of "nearby" (doesn't have to be symmetric).
- decide on where to start.

→ Keep moving to best neighbor until you get to local optimum.

For example:

Neural nets/backprop: Goodness = low squared error, "nearby" = nearby in weight space.

Decision trees/ID3: Goodness = information gain, "nearby" = add new node. Start at 0-size tree.

Disjunction learning (outer loop of FOIL): Goodness = how many positives you cover without covering negatives, "nearby" = add new term.

Different ways to use prior knowledge/domain theory

- Use to decide on starting point (KBANN).

- Use to define additional neighbors (FOCL).

- Use to change goodness criterion (EBNN).

New topic: learning a model of some process/environment having state

- Robot wandering around: model the environment.
- Speech recognition: model phoneme / word / sentence generation.
- Experimenting with some object like Rubik's cube: model underlying structure.

Issues:

- passive vs. active
- deterministic vs. probabilistic (or worse)
- all state info visible vs. hidden state
- goal is to make model (prediction) vs. goal is to do something.

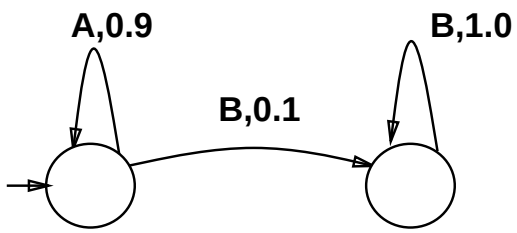
HMM: passive / probabilistic / hidden / model&prediction

Q-learning: active / probabilistic / visible / do something

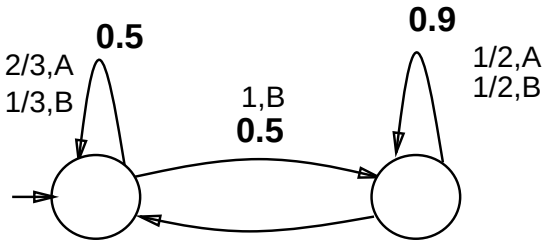
DFAs: active / deterministic / hidden / model&prediction

Hidden Markov Models

Process for generating string of characters.



(here, each edge has unique output)



- Set S of states, one marked as initial. (Sometimes also one marked as final)

- Transition probabilities: a_{ij} = prob go to state j given that in state i.

  (For each i, $\sum_j a_{ij} = 1$.)

- Output probabilities: $b_{ij}(X)$ = prob of outputting X given that on arc from i to j.

Typical uses:

- model a phoneme, word, or sentence.

- model a time process with hidden 'variables'.

- model the result of taking an action in an environment.

Typical HMM questions

- "Evaluation Problem": Find P(string | model).

  E.g., say you have HMMs for each of 50 words. You hear some sounds and want the most likely word.

  P(word | sounds) $\propto$ P(sounds | word)$\times$P(word).

- "Decoding problem": Find most likely path given string, model.

- "Learning Problem": Given a model without the probabilities and an observation, what is the best setting of the probabilities to maximize the likelihood of the observation: P(observation | model).

Evaluation problem: how to calculate P(observation | model)?

Idea: Use dynamic programming approach.

- Calculate P(at time t we're in state i *and* have produced first t characters of observation | model).

  - called $\alpha_i(t)$.

- $\alpha_i(0)$ is easy.

- $\alpha_j(t) = \sum_i \alpha_i(t-1) a_{ij} b_{ij}(obs_t)$.

- Example: BBA

- Final prob $= \sum_i \alpha_i(T)$ (T is # chars in observation) or $\alpha_{final}(T)$ if have final state.

- Practical issue: normalize to prevent underflow.

Learning problem (parameter estimation)

Goal: Try to find probabilities that maximize likelihood of observation.

Method 1: Gradient descent. (Not used much for HMMs)

- Start with some guess and compute P(obs | model).
- Try tweaking values to see if increases or decreases, and hill-climb.
- Problem: slow convergence. Hard to get exact derivatives.

Learning problem contd.

Goal: Try to find probabilities that maximize likelihood of observation.

Method 2: Baum-Welsh/Forward-backward/EM

- Start with some guess of model.
- Use model & observation to calculate expected number of traversals across each arc.

  (and expected # traversals across each arc while producing a given character)
- Use calculation to update model to model most likely to produce these ratios.

Analogy to calculating $(x + 0.3)^4 = x + 7$ by doing:

$$\begin{aligned} x_0 &= 1 \\ x_i + 0.3 &= \sqrt[4]{x_{i-1} + 7} \end{aligned}$$

(RHS less sensitive to error in guess than LHS)

Baum-Welsh, more detail

- Want to calculate for given model:
  - E_{ij} = E[# times cross $i \to j$ | obs].
  - $E_{ij}(X)$ = E[# times cross $i \to j$ while producing letter X | obs].

  So, calculate P[cross $i \to j$ at time t | obs] and add. Some details:
  - Rewrite as: $\frac{P[\text{cross } i \to j \text{ at time } t \text{ \& produce obs}]}{P[\text{produce obs}]}$
  - Numerator = P[at state i at time t and output first t chars] $\times a_{ij} \times b_{ij}(obs_t) \times$ P[output remaining chars | at state j at time $t+1$].

- Find what a_{ij}, $b_{ij}(X)$ are most likely to produce these expectations.
  - $\bar{a}_{ij} = E_{ij} / \sum_k E_{ik}$.
  - $\bar{b}_{ij}(X) = E_{ij}(X) / E_{ij}$.

Baum-Welsh, contd.

Theorem: This algorithm will converge to local optimum.

Easier to see:

- If observation determines state sequence, will converge in 1 step. (errors in numerator and denominator cancel)

General idea: Expectations less than linearly sensitive to errors in probabilities.