CSE 250A:
Probabilistic Reasoning and Decision-Making

Lecturer: Lawrence Saul

Notes by Owen Jow

1 Probability and Commonsense Reasoning 10/06/20

1.1 Probability Review

1.1.1 (Marginal) Independence

$$P(X|Y) = P(X)$$

1.1.2 Product Rule

$$P(X, Y) = P(X)P(Y|X)$$

1.1.3 Marginalization

$$P(X) = \sum_y P(X, Y = y)$$

1.1.4 Bayes' Rule

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

1.1.5 Conditionalized Versions of Everything

If you take each probability in the above expressions and add the random variable E on the right-hand side of a conditioning bar, you'll get a valid conditionalized version of the same idea or rule.

$$P(X|Y, E) = P(X|E) \quad (X \text{ is conditionally independent of } Y \text{ given } E)$$

$$P(X, Y|E) = P(X|E)P(Y|X, E) \quad \text{(conditionalized product rule)}$$

$$P(X|E) = \sum_y P(X, Y = y|E) \quad \text{(conditionalized marginalization)}$$

$$P(X|Y, E) = \frac{P(Y|X, E)P(X|E)}{P(Y|E)} \quad \text{(conditionalized Bayes' rule)}$$

1.1.6 Using These Ideas (E.g.)

- You can use independence to drop random variables from probabilities.
- You can use the product rule to break a joint probability into simpler probabilities.
- You can use marginalization to introduce new random variables into the joint. Make sure you introduce the variable(s) on the left-hand side of the conditioning bar.
- You can use Bayes' rule to make your probabilities reflect typical causal relationships, i.e. probabilities of symptoms given causes rather than the other way around.

In general, try to use these ideas to express unknown probabilities in terms of known probabilities.

1.2 Commonsense Reasoning

When you apply the rules of probability, you should end up with models that match common sense.

2 Belief Networks 10/08/20

2.1 Motivation

Goal: represent joint distributions over many random variables in a more compact way, and make inference w.r.t these distributions (i.e. evaluating certain probabilities) more efficient.

One way to perform inference for a conditional probability is to

(a) express the probability in terms of unconditional probabilities,[1]

(b) use marginalization to introduce all variables into each probability term, and

(c) evaluate all of these joint probabilities.[2]

But this often involves a marginalization sum over many terms. Not ideal.

2.2 Belief Networks

A **belief network** contains

(a) a directed acyclic graph (DAG) in which each node represents a random variable, and each edge represents a dependency between random variables, and

(b) a conditional probability table (CPT) that describes how each node depends on its parents, and ultimately allows you to compute concrete values for probabilities.

It is a compact representation of the joint distribution that encodes *independence assumptions* (via the DAG) and "*numerical influences of some variables on others*" (via the CPT).

You can exploit the structure of a belief network in order to simplify calculations during inference.

[1] Perhaps using Bayes' rule and the product rule.

[2] To evaluate a joint probability, it might help to expand the probability using the product rule and then drop variables from the constituent probabilities according to marginal/conditional independence assumptions.

2.3 To Construct a Belief Network,

1. Choose random variables of interest.
2. Choose an ordering of the random variables (preferably with causes before effects).[3]
3. For each random variable X_i,
   (a) Add node X_i to the network.
   (b) Set the parents of X_i to be the minimal subset satisfying

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|\text{parents}(X_i))$$

   (c) Define the conditional probability table $P(X_i|\text{parents}(X_i))$.

2.4 Interpreting Belief Networks

- If there is **no** path from node A to node B, then the random variables A and B are conditionally independent given all parents of A. [If $B \in$ parents(A), I don't think this tells you anything.]
- By extension, A and B are marginally independent if A has no parents and no path to B.
- If there *is* a path or an edge between nodes, it doesn't necessarily guarantee dependence. The graph might just have been constructed suboptimally, with unnecessary edges.

3 CPTs, D-separation 10/13/20

3.1 Representing CPTs

A conditional probability table is supposed to encode $P(Y|\text{parents}(Y))$, or the probability of Y taking on any of its values given all possible values of its parents.

Options for representation include tabular, logical/deterministic, noisy-OR, sigmoid...

For the time being, we will assume that all of the random variables are binary. In such a case, the CPT just needs to specify $P(Y = 1|\text{parents}(Y))$.

3.1.1 Tabular CPT

A tabular CPT is an exhaustive enumeration of probabilities given every configuration of parents. When all of the random variables are binary, this table will have 2^k rows.

[3] The better the ordering, the smaller the minimal subsets, i.e. you can use assumptions about independence to drop more variables from the product rule expansion of the joint distribution $P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|X_1, ..., X_{i-1})$. Then the graph will be more compact and encode more information (re: independence), and the CPTs will be smaller.

3.1.2 Logical/Deterministic CPT

A logical/deterministic CPT mimics a logical circuit (e.g. an AND or an OR gate), modeling the probability of Y equaling 1 as a deterministic function of its parents' values.

An AND gate corresponds to the product of each of the parents' values, and an OR gate corresponds to the product of one minus each of the parents' values.

Now, a simple formula suffices to represent the CPT, but there is no model of uncertainty. All probabilities are either 0 or 1.

3.1.3 Noisy-OR CPT

For each of the k parents $X_1, ..., X_k$, define a "probability" $p_i \in [0, 1]$. Use these numbers as part of the "noisy-OR" formula, which will represent the CPT:

$$P(Y = 0|X_1, ..., X_k) = \prod_{i=1}^{k} (1 - p_i)^{X_i}$$
$$P(Y = 1|X_1, ..., X_k) = 1 - P(Y = 0|X_1, ..., X_k)$$

Each term is only active if the corresponding $X_i = 1$.

- If every parent is equal to 0, $P(Y = 1|X_1, ..., X_k) = 0$.
- If exactly one parent X_j is equal to 1, $P(Y = 1|X_1, ..., X_k) = p_j$.

With a noisy-OR CPT, you can have non-binary probabilities.

Also, when more parents are active, $P(Y = 1|X_1, ..., X_k)$ will be higher (or at least not lower). This fits with the canonical application of the noisy-OR CPT, where the parents are diseases, Y is a symptom, and the more diseases that are active, the more likely the symptom is to appear.

3.1.4 Sigmoid CPT

Again, define k numbers (θ_i for $i = 1, ..., k$), this time $\in \mathbb{R}$. Then

$$P(Y = 1|X_1, ..., X_k) = \sigma\left(\sum_{i=1}^{k} \theta_i X_i\right)$$

where σ denotes the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. The sigmoid function squashes its argument into the range $[0, 1]$; very large arguments go to 1, and very small (negative) arguments go to 0.

- If θ_i is very large, $X_i = 1$ will push $P(Y = 1)$ closer to 1.
- If θ_i is very small (strongly negative), $X_i = 1$ will push $P(Y = 1)$ closer to 0.

In a sigmoid CPT, "some of the θ_i's can be positive, and some of the θ_i's can be negative," and so the above effects can be mixed. (By contrast, in a noisy-OR CPT there's a monotonicity – more parents coming on *always* means that the resulting probability will be equal or higher).

3.2 D-separation

If X, Y, and E are disjoint sets of nodes in a belief network, d-separation (*direction-dependent separation*) will tell us whether X and Y are conditionally independent given E.

Theorem. X and Y are conditionally independent given E if and only if every path from a node in X to a node in Y (ignoring directionalities of edges) is blocked by E.

"A path π is **blocked** by a set of nodes E if there exists a node $Z \in \pi$ for which one of the three following conditions hold."

1. $Z \in E$, and the adjacent edges in the path flow through the node (one enters Z, one exits Z).
2. $Z \in E$, and the adjacent edges in the path diverge from the node (both exit Z).
3. $Z \notin E$, all descendants of Z are also $\notin E$, and the adjacent edges in the path both enter Z.

Note: you have to check *all* paths, but you only have to find *one* node in each path!

3.2.1 Intuition

Any path from X to Y is a way that information from X *could* be propagated to Y. Each path thus represents a means by which information about Y *could* tell us something about X.

So if a path is blocked by the evidence set E, then Y cannot give us any information that E is not already giving us. In other words, if we know E, Y won't give us any extra information about X.

1. "Z is an intervening event in a causal chain."
2. "Z is a common explanation or cause."
3. "Z and its descendants represent an unobserved common effect."
    - Think of X and Y as causes. We would need to observe their common effect in order to get any information about one from the other (on this path), but we don't.

4 Inference in Polytrees 10/15/20

4.1 Inference

Goal: given a belief network, a set Q of query nodes, and a set E of evidence nodes, compute the posterior distribution $P(Q|E)$ for certain values of the query and evidence nodes.

You will need to express $P(Q|E)$ in terms of the CPTs $P(X_i|\text{parents}(X_i))$ of the belief network, since those are the only probabilities for which numbers initially exist. To do so:

1. Use **Bayes' rule** to express $P(Q|E)$ in terms of conditional probabilities that respect the order of the DAG. Predict descendants given their ancestors.
2. Use **marginalization** to add parent nodes to the probabilities.
3. Use the **product rule** to break down the joint probabilities.
4. Use **independence** to remove nodes from the right-hand sides of the conditioning bars.

4.2 Polytrees

A **polytree** is a belief network for which there is at most one path (again ignoring directionalities of edges) between any two nodes. Equivalently, a polytree is a belief network without any undirected cycles. Apparently, the name comes from the idea that polytrees are like trees that allow multiple parents per node. (All trees are polytrees but not all polytrees are trees.)

If X is a node in a polytree, then there is a distinct polytree subgraph for each parent and child of X (just the section of the graph branching out of the parent or child away from X[4]). No two of these subgraphs have common nodes.[5]

4.3 Inference in Polytrees

With a polytree, exact inference is tractable using a recursive algorithm.

- By employing Bayes' rule, marginalization, the product rule, and independence (proven from the DAG using d-separation), you can derive an inference algorithm for $P(X = x|E)$ that recurses on the parent and child subgraphs (again, also polytrees) of X.
- Since there are no loops in a polytree, the recursion is guaranteed to terminate. Eventually each recursion will reach an X node for which computation is trivial (a root node, a leaf node, or a node in the evidence set).

5 Inference in Loopy Belief Networks 10/20/20

5.1 Exact Inference in Loopy Belief Networks

When there are only a few loops, exact inference is still feasible. **Main idea:** transform the loopy belief network into a polytree and run the polytree algorithm from before.

5.1.1 Node Clustering

Cluster and merge nodes to remove loops, so that the graph becomes a polytree.

- Also merge the CPTs associated with the nodes, s.t. there is one big CPT for the merged node. Note: the CPT for the merged node grows exponentially with the number of constituent nodes.[6]
- The complexity of the polytree algorithm scales linearly with the sizes of the CPTs. So: keep the CPTs as small as possible.
  - **But** when you cluster nodes, the CPTs grow exponentially! So: cluster the minimal set of nodes that turn the belief network into a polytree.
    - **But** there is no efficient way to find such a clustering.

[4] More specifically, the parent/child along with all nodes connected to it **not** via X.

[5] If there were any common nodes, there would have to be a connection between the subgraphs not going through X, and since there is already a connection between the subgraphs going through X, this would imply a loop.

[6] For example, if you merge three binary random variables, then the resulting super-node has eight possible values.

5.1.2 Cutset Conditioning

Remove nodes by instantiating them as evidence.[7] After removing these nodes, the graph should be a polytree. Call the polytree algorithm for each instantiation, and combine the results.

- **Cutset:** the set of instantiated nodes.[8]
- For example, to compute $P(V = 1)$ from a belief network where removing D [a binary random variable that has no parents, s.t. the CPT $P(D = 1)$ is known] creates a polytree:
  1. Compute $P(V = 1|D = 0)$ and $P(V = 1|D = 1)$ using the polytree algorithm.
  2. Compute $P(V = 1)$ as $P(D = 0)P(V = 1|D = 0) + P(D = 1)P(V = 1|D = 1)$.
- The number of instantiations[9] grows exponentially with the size of the cutset.
  - **But** there is no efficient algorithm to find the minimal cutset.

5.2 Approximate Inference in Loopy Belief Networks

Exact inference in general belief networks is intractable (#P-hard). So: use **stochastic simulation methods** ("run lots of simulations") to approximate the probabilities you want to compute.

5.2.1 Sampling from the Joint Distribution in a Belief Network

If $X_1, ..., X_n$ are discrete random variables in a belief network, then the joint distribution is

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|\text{parents}(X_i))$$

according to the generative model defined by the belief network. (A *generative model* makes it easy to generate samples from the joint distribution.)

To sample from the joint distribution (i.e. sample a value for each X_i), simply perform a "forward pass" through the belief network, sampling each variable one by one according to $P(X_i|\text{parents}(X_i))$.

To sample a value for each X_i, simply generate a random number between 0 and 1 and map that number into a particular discrete value according to $P(X_i|\text{parents}(X_i))$.[10]

5.2.2 Approximate Inference

Goal: given a belief network, a set Q of query nodes, and a set E of evidence nodes, **estimate** the posterior distribution $P(Q|E)$ for certain values of the query and evidence nodes.

[7] If you do this for one binary random variable, you end up with one belief network where you assume that the random variable equals 0 and one belief network where you assume that the random variable equals 1.

[8] Why the name? It's the *set* you *cut* off from the rest of the network.

[9] Read: *the number of times that you need to run the polytree algorithm.*

[10] Each possible value of X_i occupies a partition of the range $[0, 1]$.

5.2.3 Rejection Sampling

To estimate $P(Q = q|E = e)$,

1. Generate N samples from the belief network's joint distribution.
2. Reject all samples where $E \neq e$ (where the evidence variables don't have the desired values).
3. Count the number of samples $N(q, e)$ for which $Q = q$ and $E = e$.
4. Count the number of samples $N(e)$ for which $E = e$.
5. Estimate $P(Q = q|E = e)$ as $\frac{N(q,e)}{N(e)}$.

In the limit, the rejection sampling estimate will converge to the correct value. But you will probably have to throw away a lot of samples (all samples without the relevant evidence). So the process is extremely inefficient, and converges *very slowly* for evidence with a low probability of occurring.

Note:

$$\frac{N(q,e)}{N(e)} = \frac{\sum_{i=1}^{N} I(q, q_i)I(e, e_i)}{\sum_{i=1}^{N} I(e, e_i)}$$

where $I(x, y)$ is 1 if $x = y$ and 0 otherwise.

5.2.4 Likelihood Weighting

Instead of sampling evidence nodes, instantiate them![11] Fix them to have the desired values. Then weight each sample by its likelihood (based on the evidence nodes' CPTs).

$$P(Q = e|E = e) \approx \frac{\sum_{i=1}^{N} I(q, q_i)P(E = e|\text{parents}(E) \text{ having observed values})}{\sum_{i=1}^{N} P(E = e|\text{parents}(E) \text{ having observed values})}$$

No more discarding! No more wasted computation! Faster convergence!

But it can still converge very slowly (if rare evidence is descended from the query nodes).[12]

6 Inference and Learning in Belief Networks 10/22/20

6.1 Markov Chain Monte Carlo (MCMC)

The most *generally* efficient of the approximate inference methods [that are covered in this class].

Goal: have evidence nodes affect how other nodes are sampled.

6.1.1 Markov Blanket

The **Markov blanket** B_X of a node X consists of its parents, children, and spouses[13].

- X is conditionally independent of the nodes outside of B_X given the nodes inside of B_X.

[11] Note: continue to sample non-evidence nodes based on the values of their parents.

[12] Likelihood weighting will converge more quickly if evidence nodes are ancestors of the query nodes. This is because the (fixed) evidence in such a case can affect how the query nodes are sampled.

[13] **Spouse:** any other parent of its child.

6.1.2 Simulation Process

1. **Initialization:**
   (a) Fix evidence nodes to have observed values.
   (b) Initialize non-evidence nodes to random values.
2. **Repeat N times:**
   (a) Pick a non-evidence node X at random.
   (b) Use **Bayes' rule** to compute $P(X|B_X)$.
   (c) Resample x from the posterior distribution $P(X|B_X)$.
   (d) Take a snapshot of all the nodes in the belief network.
3. **Estimate:**
   (a) Count the number of snapshots where the query nodes have the desired values.
   (b) Compute $\frac{\text{number of matched snapshots}}{\text{number of total snapshots } (N)}$. That ratio is the estimate for $P(Q = q|E = e)$.

6.1.3 Properties

- This sampling procedure "defines an ergodic Markov chain over the non-evidence nodes of the belief network" [if you sample enough, the procedure gives rise to a stationary distribution over the non-evidence nodes which is exactly the posterior distribution $P(\text{non-evidence nodes}|E)$].
- The MCMC estimates converge in the limit to the correct $P(Q|E)$.
- Before each sample, MCMC must compute $P(X|B_X)$ – leading to extra cost per step.
- MCMC can be much faster than likelihood weighting when rare evidence is in leaf nodes.
- To account for a possible poor initialization, you might want to add a "burn-in phase" in which statistics are not tracked for the first 1000-or-so samples.

6.2 Learning in Belief Networks

Learn from data how to estimate belief networks which can serve as useful models.

6.2.1 Maximum Likelihood Estimation

Model data by the belief network that assigns it the highest probability.

Choose the DAG and CPTs to maximize the likelihood

$$P(\text{observed data} \mid \text{DAG and CPTs})$$

6.2.2 Learning with Complete Data and Tabular CPTs

Assume:

- The DAG is known and fixed over a finite set of discrete random variables $\{X_1, ..., X_n\}$.
- The data consists of T complete instantiations of all the nodes in the belief network.
  - In notation, the dataset is $\left\{\left(x_1^{(t)}, ..., x_n^{(t)}\right)\right\}_{t=1}^{T}$.
  - The examples are assumed to be i.i.d. from the joint distribution of the belief network.
- The CPTs enumerate $P(X_i = x|\text{parents}(X_i) = \pi)$ and must be estimated for all x and π.

Since the examples are i.i.d., the probability of the dataset is

$$P(\text{data}) = \prod_{t=1}^{T} P\left(X_1 = x_1^{(t)}, ..., X_n = x_n^{(t)}\right)$$

The probability of the t^{th} example is

$$P\left(X_1 = x_1^{(t)}, ..., X_n = x_n^{(t)}\right) = \prod_{i=1}^{n} P\left(X_i = x_i^{(t)}|\text{parents}(X_i) = \text{parents}_i^{(t)}\right)$$

The log-likelihood is

$$\begin{aligned}
\mathcal{L} &= \log P(\text{data}) \\
&= \log \prod_{t=1}^{T} P\left(x_1^{(t)}, ..., x_n^{(t)}\right) \\
&= \log \prod_{t=1}^{T} \prod_{i=1}^{n} P\left(x_i^{(t)}|\text{parents}_i^{(t)}\right) \\
&= \sum_{i=1}^{n} \sum_{t=1}^{T} \log P\left(x_i^{(t)}|\text{parents}_i^{(t)}\right)
\end{aligned}$$

Goal: find the CPTs that make the log-likelihood as high as possible.

Let $\text{count}(X_i = x, \text{parents}(X_i) = \pi)$ denote the number of examples where $X_i = x$ and $\text{parents}(X_i) = \pi$. Now, when computing the log-likelihood, (1) weight each log-probability by the number of times it appears in the dataset and (2) sum over *possible values of x and π* instead of *examples*.

$$\mathcal{L} = \sum_{i=1}^{n} \sum_{x} \sum_{\pi} \underbrace{\text{count}(X_i = x, \text{parents}(X_i) = \pi)}_{\text{constants of the data}} \underbrace{\log P\left(X_i = x|\text{parents}(X_i) = \pi\right)}_{\text{CPTs to optimize}}$$

Intuitively, the larger the $\text{count}(X_i = x, \text{parents}(X_i) = \pi)$, the larger you should try to make $P\left(X_i = x|\text{parents}(X_i) = \pi\right)$. Note that you can't just make all of the probabilities large, because they're all parts of subsets which have to be normalized and sum to 1.

$$P_{\text{MLE}}\left(X_i = x|\text{parents}(X_i) = \pi\right) \propto \text{count}(X_i = x, \text{parents}(X_i) = \pi)$$

For a node with parents,

$$P_{\text{MLE}}\left(X_i = x|\text{parents}(X_i) = \pi\right) = \frac{\text{count}(X_i = x, \text{parents}(X_i) = \pi)}{\text{count}(\text{parents}(X_i) = \pi)}$$

For a root node,

$$P_{\text{MLE}}(X_i = x) = \frac{\text{count}(X_i = x)}{T}$$

7 Learning from Complete Data 10/27/20

7.1 Markov Models

7.1.1 Statistical Language Modeling

Goal: model the joint probability of a text $P(w_1, w_2, ..., w_L)$, where w_l is the l^{th} word in the text and L is the number of words in the text.

Assume:

- To predict the l^{th} word, you only need to consider the $n-1$ words that precede it:

$$P(w_l|w_1, w_2, ..., w_{l-1}) = P(w_l|w_{l-(n-1)}, ..., w_{l-1})$$

- Predictions don't depend on where the context appears in the text:

$$P(W_l = w'|w_{l-(n-1)}, ..., w_{l-1}) = P(W_{s+l} = w'|w_{s+l-(n-1)}, ..., w_{s+l-1})$$

Thus,

$$\begin{aligned} P(w_1, w_2, ..., w_L) &= \prod_l P(w_l|w_1, w_2, ..., w_{l-1}) && \text{(product rule)} \\ &= \prod_l P(w_l|w_{l-(n-1)}, ..., w_{l-1}) && \text{(conditional independence)} \end{aligned}$$

When $n = 1$ (**unigram** case), none of the words depend on any context and the belief network [over word nodes] has no edges. When $n = 2$ (**bigram** case), each word depends only on the word before it, and the belief network has an edge going from each word to the next word. In general (**n-gram** case), each word is connected to the $n-1$ previous words in the document.

According to the second assumption (position invariance), each non-root node has the same CPT.

To learn the CPT $P(W_l = w'|W_{l-1} = w)$ for non-root nodes in a bigram model,

- Collect a bunch of textual data over a well-defined vocabulary.
- Count how many times the word w is followed by w'.
- Count how many times the word w is followed by any word.
- Estimate

$$\begin{aligned} P_{\text{MLE}}(W_l = w'|W_{l-1} = w) &= \frac{\text{count}(w \to w')}{\text{count}(w \to *)} \\ &= \frac{\text{count}(w \to w')}{\sum_{w''} \text{count}(w \to w'')} \end{aligned}$$

Note: this estimate will assign zero probability to bigrams that do not appear in the training data.

7.2 Naive Bayes

7.2.1 Document Classification

Goal: label an arbitrary document by one of m topics, where each document consists of words from a finite vocabulary. Let $Y \in \{1, 2, ..., m\}$ denote the label, and let $X_i \in \{0, 1\}$ denote whether word i appears (mapping every document to a "sparse binary vector of fixed length"[14]).

The belief network features Y as a root node with all of the X_i's as its children. Thus there is the Naive Bayes assumption of conditional independence (which is probably not actually true) that

$$P(X_1, X_2, ..., X_n|Y) = \prod_{i=1}^{n} P(X_i|Y)$$

Again, it will be necessary to estimate the CPTs $P(Y = y)$ and $P(X_i = 1|Y)$ for the belief network.

- Collect a dataset of documents.
- Label each document by one of the m topics.
- Estimate the CPTs by maximizing the likelihood of the data.

$$P_{\text{MLE}}(Y = y) = \frac{\text{number of documents with label } y}{\text{total number of documents}}$$

$$P_{\text{MLE}}(X_i = 1|Y = y) = \frac{\text{number of documents with label } y \text{ that contain word } i}{\text{number of documents with label } y}$$

8 Linear Regression and Least-Squares 10/29/20

8.1 Linear Regression

Goal: predict the real-valued random variable $Y \in \mathbb{R}$ from real-valued parents $\mathbf{X} \in \mathbb{R}^d$.

Model the CPT relationship between the X_i's and Y using a **Gaussian conditional distribution**:

$$P(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\left(y - \mathbf{w}^\top \mathbf{x}\right)^2}{2\sigma^2}\right]$$

Learned parameters:

- The variance σ^2, which controls the level of uncertainty.
- The weight vector $\mathbf{w}$, which determines the mean of the Gaussian ($\mathbf{w}^\top \mathbf{x}$) based on $\mathbf{x}$.

To estimate the variance and the weights, use linear regression.

8.1.1 Problem Setup

Given

- a set of T complete training examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_T, y_T)\}$

[14] The length is equivalent to the number of words in the vocabulary.

- generated in an i.i.d. fashion [meaning $P(y_1, ..., y_T|\mathbf{x}_1, ..., \mathbf{x}_T) = \prod_{t=1}^{T} P(y_t|\mathbf{x}_t)$]
- under the assumption that $\mathbf{X}$ is always observed [meaning $P(\mathbf{x})$ need not be modeled],

estimate the σ^2 and $\mathbf{w}$ that maximize the conditional likelihood $P(y_1, ..., y_T|\mathbf{x}_1, ..., \mathbf{x}_T)$.

8.1.2 Interpreting the Log Conditional Likelihood

After some manipulation, the log conditional likelihood reduces to

$$\mathcal{L}(\mathbf{w}, \sigma^2) = -\frac{1}{2} \sum_{t=1}^{T} \left[\log(2\pi\sigma^2) + \frac{\left(y_t - \mathbf{w}^\top \mathbf{x}_t\right)^2}{\sigma^2} \right]$$

The weights $\mathbf{w}$ that maximize $\mathcal{L}(\mathbf{w}, \sigma^2)$ also minimize the MSE $\mathcal{E}(\mathbf{w})$:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{T} \sum_{t=1}^{T} \left(y_t - \mathbf{w}^\top \mathbf{x}_t\right)^2$$

Thus, MLE linear regression (for $\mathbf{w}$) is equivalent to the least-squares problem for a linear fit.

8.1.3 Maximizing the Log Conditional Likelihood

To maximize the log conditional likelihood $\mathcal{L}$, compute the partial derivatives $\frac{\partial \mathcal{L}}{\partial w_\alpha}$ (for $\alpha = 1, 2, ..., d$) and solve for the $w_1, w_2, ..., w_d$ where they vanish. You'll end up with d equations

$$\begin{aligned}
\sum_{t=1}^{T} y_t x_{\alpha t} &= \sum_{t=1}^{T} (\mathbf{w}^\top \mathbf{x}_t) x_{\alpha t} \\
&= \sum_{t=1}^{T} \left(\sum_{\beta=1}^{d} w_\beta x_{\beta t} \right) x_{\alpha t} \\
&= \sum_{\beta=1}^{d} \left(\sum_{t=1}^{T} x_{\alpha t} x_{\beta t} \right) w_\beta
\end{aligned}$$

Then define a $d \times d$ matrix $\mathbf{A}$ and a $d \times 1$ vector $\mathbf{b}$ s.t.

$$\begin{aligned}
A_{\alpha\beta} = \sum_{t=1}^{T} x_{\alpha t} x_{\beta t} \qquad &\implies \qquad \mathbf{A} = \sum_{t=1}^{T} \mathbf{x}_t \mathbf{x}_t^\top \\
b_\alpha = \sum_{t=1}^{T} y_t x_{\alpha t} \qquad &\implies \qquad \mathbf{b} = \sum_{t=1}^{T} y_t \mathbf{x}_t
\end{aligned}$$

Finally, to solve the system of linear equations, solve for $\mathbf{w}$ in $\mathbf{A}\mathbf{w} = \mathbf{b} \implies \mathbf{w}_{\text{MLE}} = \mathbf{A}^{-1}\mathbf{b}$.

As long as the inputs $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$ span $\mathbb{R}^d$, $\mathbf{A}$ will be invertible. But if $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$ don't span $\mathbb{R}^d$, you'll have to compute the minimum norm solution by minimizing $\|\mathbf{w}\|$ s.t. $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$.

8.2 Numerical Optimization

Goal: maximize/minimize a multivariable function $f(\vec{\theta})$ over $\vec{\theta} = (\theta_1, ..., \theta_d) \in \mathbb{R}^d$.

One way to do this is to compute the gradient and set it equal to 0. But in general this will be too difficult, and a numerical/algorithmic/iterative solution will be required.

- **Hillclimbing method:** perform iterative local search for a local/global maximum of $f(\vec{\theta})$.

8.2.1 Gradient Ascent/Descent

The gradient $\nabla f = \frac{\partial f}{\partial \vec{\theta}}$ points in the direction of fastest increase in $f(\vec{\theta})$.

Update the parameter vector according to

$$\vec{\theta} \leftarrow \vec{\theta} \pm \eta_t \left(\frac{\partial f}{\partial \vec{\theta}} \right)$$

where $\eta_t > 0$ is the step size (/learning rate).

8.2.2 Newton's Method

Gradient ascent/descent is based on a linear approximation,
whereas Newton's method is based on a quadratic approximation (and involves no learning rate).

Note: Newton's method requires expensive Hessian calculations (though it will converge quickly when it does converge) and can be unpredictable and/or unstable when the initial estimate sucks.[15]

9 Learning from Complete and Incomplete Data 11/03/20

9.1 Logistic Regression

Goal: predict the binary random variable $Y \in \{0, 1\}$ from real-valued parents $\mathbf{X} \in \mathbb{R}^d$.

Model the CPT relationship between the X_i's and Y using a **sigmoid conditional distribution**:

$$\begin{aligned} P(Y = 1|\mathbf{x}) &= \sigma\left(\mathbf{w}^\top \mathbf{x}\right) \\ &= \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} \end{aligned}$$

Logistic regression: learn the parameter $\mathbf{w} \in \mathbb{R}^d$ from complete data.

9.1.1 Problem Setup

Given: a complete dataset of i.i.d. examples $\{(\mathbf{x}_t, y_t)\}_{t=1}^T$ where $\mathbf{x}_t \in \mathbb{R}^d$ and $y_t \in \{0, 1\}$.

[15] e.g. when the initial estimate is almost equidistant from a local maximum and a local minimum

9.1.2 Maximizing the Log Conditional Likelihood

The log conditional likelihood is

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \log P(y_1, y_2, ..., y_T | x_1, x_2, ..., x_T) \\ &\vdots \\ &= \sum_{t=1}^{T} \left[y_t \log \sigma(\mathbf{w}^\top \mathbf{x}_t) + (1 - y_t) \log \sigma(-\mathbf{w}^\top \mathbf{x}_t) \right]\end{aligned}$$

The partial derivative of $\mathcal{L}(\mathbf{w})$ with respect to w_α is

$$\frac{\partial \mathcal{L}}{\partial w_\alpha} = \sum_{t=1}^{T} x_{\alpha t} \underbrace{\left[y_t - \sigma(\mathbf{w}^\top \mathbf{x}_t) \right]}_{\text{error signal}}$$

Unfortunately, you cannot solve for $\mathbf{w}$ analytically (by setting the gradient to $\mathbf{0}$) because the non-linearity of the sigmoid function creates an unsolvable system of equations. Therefore, you have to use a numerical process such as gradient ascent or Newton's method to solve this problem.

On the plus side, $\mathcal{L}(\mathbf{w})$ is a concave function of $\mathbf{w}$, meaning local maxima are also global maxima!

9.2 Learning from Incomplete Data

9.2.1 Learning from Incomplete Data with Tabular CPTs

Assume:

- The DAG is known and fixed over a finite set of discrete random variables $\{X_1, ..., X_n\}$.
- The CPTs enumerate $P(X_i = x | \text{parents}(X_i) = \pi)$ and must be estimated for all x and π.
- The i.i.d. data consists of T partially complete instantiations of the nodes in the belief network.

Let

- H_t be the set of hidden/latent variables for the t^{th} example.
- V_t be the set of visible/observed variables for the t^{th} example.

The log-likelihood of the data is

$$\begin{aligned}\mathcal{L} &= \log \prod_{t=1}^{T} P(V_t = v_t) \\ &\vdots \\ &= \sum_{t=1}^{T} \log \sum_{h} \prod_{i=1}^{n} P(X_i = x_i | \text{parents}(X_i) = \pi_i) \Big|_{\{H_t = h, V_t = v_t\}}\end{aligned}$$

Unfortunately, the CPTs are potentially all coupled due to the summation over hidden variables.

9.3 Auxiliary Functions

A function $Q(\vec{\theta}', \vec{\theta})$ is an **auxiliary function** for the objective function $f(\vec{\theta})$ if:

(i) $Q(\vec{\theta}, \vec{\theta}) = f(\vec{\theta})$ for all $\vec{\theta}$

(ii) $Q(\vec{\theta}', \vec{\theta}) \leq f(\vec{\theta}')$ for all $\vec{\theta}, \vec{\theta}'$

If $Q(\vec{\theta}', \vec{\theta})$ is an auxiliary function for the objective function $f(\vec{\theta})$, then the update rule

$$\vec{\theta}_{\text{new}} = \arg\max_{\vec{\theta}} Q(\vec{\theta}, \vec{\theta}_{\text{old}})$$

converges monotonically to a stationary point[16] with $f(\vec{\theta}_{\text{new}}) \geq f(\vec{\theta}_{\text{old}})$.

In other words, the update is guaranteed to monotonically increase the objective value.

Proof.

$$\begin{aligned} f(\vec{\theta}_{\text{new}}) &\geq Q(\vec{\theta}_{\text{new}}, \vec{\theta}_{\text{old}}) && \text{(property (ii))} \\ &\geq Q(\vec{\theta}_{\text{old}}, \vec{\theta}_{\text{old}}) && \text{(argmax update)} \\ &= f(\vec{\theta}_{\text{old}}) && \text{(property (i))} \end{aligned}$$

10 The EM Algorithm

11/05/20

10.1 In a Nutshell

Goal: estimate CPTs[17] from incomplete data (as described in the previous lecture).

- Randomly initialize the CPTs with nonzero elements.[18]
- Repeat until convergence:
  - **[E-Step]** Use the CPTs to "infer values for the missing data."[19]
  - **[M-Step]** Re-estimate the CPTs from the newly completed data.[20]

10.2 E-Step (Inference)

During the E-step, you compute posterior probabilities that can be used to fill in missing data.[21]

- At root nodes, you should compute $P(X_i = x | V_t = v_t)$.
- At nodes with parents, you should compute $P(X_i = x, \text{parents}(X_i) = \pi | V_t = v_t)$.

These probabilities must be computed in a quadruple loop over **every example** V_t, **every node** X_i, **every value of** $X_i = x$, and **every configuration of the parents**$(X_i) = \pi$.

[16] **Stationary point:** a point where the gradient vanishes (saddle point or local/global maximum/minimum).

[17] Note: you can use the EM algorithm to estimate many other types of probabilistic models as well.

[18] Assign random probabilities to all $P(X_i = x | \text{parents}(X_i) = \pi)$, and avoid zeros because they cannot be unlearned.

[19] [E-Step] Use the current CPTs to compute the posterior probabilities $P(H_t = h | V_t = v_t)$.

[20] [M-Step] Update the CPTs based on the computed posterior probabilities.

[21] The posterior probabilities provide a sense of how likely different values of the missing data are.

10.3 M-Step (Learning)

In the M-step, you use the posterior probabilities (which you just computed) to update the CPTs.

- At root nodes,

$$P(X_i = x) \leftarrow \frac{1}{T} \sum_{t=1}^{T} P(X_i = x | V_t = v_t)$$

- At nodes with parents,

$$P(X_i = x | \text{parents}(X_i) = \pi) \leftarrow \frac{\sum_{t=1}^{T} P(X_i = x, \text{parents}(X_i) = \pi | V_t = v_t)}{\sum_{t=1}^{T} P(\text{parents}(X_i) = \pi | V_t = v_t)}$$

10.4 Key Properties

- **No learning rate to tune.**
- **Monotonic convergence.**
  - The CPT updates never decrease the log-likelihood of the incomplete data.[22]
  - (The goal, of course, is to maximize the log-likelihood of the incomplete data.)

10.5 More Generally

- **E-Step / Expectation:**
  Compute the auxiliary function (sum of expected values given the current CPTs).[23]
- **M-Step / Maximization:**
  Choose the CPTs that maximize the auxiliary function.[24]

11 Latent Variable Models 11/10/20

If A and B are discrete random variables with m and n possible values (respectively), then you can think of $P(A|B)$ as an $m \times n$ matrix containing the probabilities for every (a, b) combination.

12 Latent Variable Models, cont. 11/12/20

12.1 Noisy-OR Models

Noisy-OR model: predict the value of a binary node Y conditioned on the values of a large number of binary parent nodes $X_1, X_2, ..., X_k$ according to

[22] $\mathcal{L} = \sum_t \log P(V_t = v_t)$

[23] The auxiliary function is the lower-bounding function that you move along (in the M-step) to get each update.

[24] Each argument to the auxiliary function represents a setting of all CPTs in the belief network.

$$P(Y = 1|x_1, x_2, ..., x_k) = 1 - \prod_{i=1}^{k}(1 - p_i)^{x_i}$$

You can use the EM algorithm to estimate parameters p_i that maximize the log-conditional likelihood

$$\mathcal{L} = \sum_t \log P(y_t|x_t)$$

To use the EM algorithm, you have to add a hidden node (representing the latent, binary random variable $Z_i \in \{0, 1\}$) between each X_i and Y. These Z_i's serve as the unobserved data.

$$P(Z_i = 1|x_i) = p_i x_i \qquad (\text{“}Z_i \text{ is a noisy copy of } X_i\text{”})$$
$$P(Y = 1|Z_1, Z_2, ..., Z_n) = \texttt{OR}(Z_1, Z_2, ..., Z_n) \qquad (\texttt{OR} \text{ is the logical-OR operator})$$

12.2 Hidden Markov Models

12.2.1 Overview

- At each time step t, there is a hidden state S_t and an observation O_t.
- Each O_t depends on S_t, and each S_t (with the exception of S_1) depends on S_{t-1}.
- An observation O_t is meant to be a “noisy, partial reflection” of the true, underlying state S_t.
- Frequently, the goal is to estimate $P(\text{some subset of } s_i\text{'s} \mid \text{some subset of } o_i\text{'s})$.
- Assume that each state has n possible values and that each observation has m possible values.

12.2.2 HMMs as Belief Networks

With an HMM, you'll make the following assumptions of conditional independence:

$$P(S_t|S_1, S_2, ..., S_{t-1}) = P(S_t|S_{t-1})$$
$$P(O_t|S_1, S_2, ..., S_T) = P(O_t|S_t)$$

Also, you'll assume that the CPTs are shared across time:

$$P(S_t = s'|S_{t-1} = s) = P(S_{t+1} = s'|S_t = s)$$
$$P(O_t = o|S_t = s) = P(O_{t+1} = o|S_{t+1} = s)$$

The joint distribution can be written as

$$P(S_1, S_2, ..., S_T, O_1, O_2, ..., O_T) = P(S_1)P(O_1|S_1)\prod_{t=2}^{T}[P(S_t|S_{t-1})P(O_t|S_t)]$$

12.2.3 Parameters

- $a_{ij} = P(S_{t+1} = j|S_t = i)$ [$n \times n$ transition matrix]
- $b_{ik} = P(O_t = k|S_t = i)$ [$n \times m$ emission matrix]
- $\pi_i = P(S_1 = i)$ [$n \times 1$ initial state distribution]

13 Hidden Markov Models 11/17/20

13.1 Forward Algorithm

Goal: compute the likelihood $P(o_1, o_2, ..., o_T)$ of a sequence of observations in an HMM.

Strategy: build up the quantity $P(o_1, o_2, ..., o_t, S_t = i)$ for each i over increasing values of t.

For a particular sequence of observations $\{o_1, o_2, ..., o_T\}$, define the $n \times T$ matrix with elements

$$\alpha_{it} = P(o_1, o_2, ..., o_t, S_t = i)$$

This is an inference problem, so you can assume that you have the three parameters of the HMM (the transition matrix, the emission matrix, and the initial state distribution). Given those,

$$\begin{aligned}\alpha_{i1} &= P(o_1, S_1 = i) \\ &= P(S_1 = i)P(o_1|S_1 = i) \\ &= \pi_i b_{io_1}\end{aligned}$$

The rest of the columns (going from left to right) can be filled in as follows:

$$\begin{aligned}\alpha_{j,t+1} &= P(o_1, ..., o_{t+1}, S_{t+1} = j) \\ &= \sum_{i=1}^{n} P(o_1, ..., o_{t+1}, S_t = i, S_{t+1} = j) \\ &= \sum_{i=1}^{n} [P(o_1, ..., o_t, S_t = i)P(S_{t+1} = j|o_1, ..., o_t, S_t = i)P(o_{t+1}|o_1, ..., o_t, S_t = i, S_{t+1} = j)] \\ &= \sum_{i=1}^{n} [P(o_1, ..., o_t, S_t = i)P(S_{t+1} = j|S_t = i)P(o_{t+1}|S_{t+1} = j)] \\ &= \sum_{i=1}^{n} \alpha_{it} a_{ij} b_{jo_{t+1}}\end{aligned}$$

After you've filled in all the columns, you will have the probability $P(o_1, o_2, ..., o_T, S_T = i)$ for every value of i, and you can add up those probabilities (i.e. marginalize out S_T) to get $P(o_1, o_2, ..., o_T)$.

This algorithm, where you fill in the matrix of α_{it} elements one column at a time (from left to right, moving forward in time), is called the **forward algorithm**.

13.2 Viterbi Algorithm

Goal: compute the most likely sequence of states $[\arg\max_{s1,...,s_T} P(s_1, ..., s_T|o_1, ..., o_T)]$ in an HMM.

$$\begin{aligned}\underset{s1,...,s_T}{\arg\max}\, P(s_1, ..., s_T|o_1, ..., o_T) &= \underset{s1,...,s_T}{\arg\max} \left[\frac{P(s_1, ..., s_T, o_1, ..., o_T)}{P(o_1, ..., o_T)}\right] \\ &= \underset{s1,...,s_T}{\arg\max}\, P(s_1, ..., s_T, o_1, ..., o_T) \\ &= \underset{s1,...,s_T}{\arg\max} \log P(s_1, ..., s_T, o_1, ..., o_T)\end{aligned}$$

For a particular sequence of observations $\{o_1, o_2, ..., o_T\}$, define the $n \times T$ matrix with elements

$$l^*_{it} = \max_{s_1,...,s_{t-1}} \log P(s_1, ..., s_{t-1}, S_t = i, o_1, ..., o_t)$$

l^*_{it} is the log-probability of the subsequence of t hidden states $s_1, ..., s_t$ that best explains the t observations $o_1, ..., o_t$ with the additional constraint that the latest hidden state $S_t = i$.

Again, this is an inference problem, so you have access to the transition matrix, the emission matrix, and the initial state distribution. The first column of the matrix ($t = 1$) can be computed as

$$\begin{aligned} l^*_{i1} &= \log P(S_1 = i, o_1) \\ &= \log \left[P(S_1 = i)P(o_1|S_1 = i)\right] \\ &= \log \pi_i + \log b_{io_1} \end{aligned}$$

The rest of the columns ($t > 1$) can be computed as

$$\begin{aligned} l^*_{j,t+1} &= \max_{s_1,...,s_t} \log P(s_1, ..., s_t, S_{t+1} = j, o_1, ..., o_{t+1}) \\ &= \max_i \max_{s_1,...,s_{t-1}} \log P(s_1, ..., s_{t-1}, S_t = i, S_{t+1} = j, o_1, ..., o_{t+1}) \\ &= \max_i \max_{s_1,...,s_{t-1}} \log[P(s_1, ..., s_{t-1}, S_t = i, o_1, ..., o_t) \\ &\qquad \cdot P(S_{t+1} = j|s_1, ..., s_{t-1}, S_t = i, o_1, ..., o_t) \\ &\qquad \cdot P(o_{t+1}|s_1, ..., s_{t-1}, S_t = i, S_{t+1} = j, o_1, ..., o_t)] \\ &= \max_i \max_{s_1,...,s_{t-1}} \log \left[P(s_1, ..., s_{t-1}, S_t = i, o_1, ..., o_t)P(S_{t+1} = j|S_t = i)P(o_{t+1}|S_{t+1} = j)\right] \\ &= \max_i \max_{s_1,...,s_{t-1}} \left[\log P(s_1, ..., s_{t-1}, S_t = i, o_1, ..., o_t) + \log a_{ij} + \log b_{jo_{t+1}}\right] \\ &= \max_i \left[l^*_{it} + \log a_{ij}\right] + \log b_{jo_{t+1}} \end{aligned}$$

13.2.1 Computing the Most Likely Sequence of States

To derive the most likely sequence of hidden states $[\{s^*_1, ..., s^*_T\} = \arg\max_{s1,...,s_T} P(s_1, ..., s_T|o_1, ..., o_T)]$ from the matrix of l^*_{it} values that you just computed, construct the $n \times T$ matrix with elements

$$\phi_{t+1}(j) = \arg\max_i \left[l^*_{it} + \log a_{ij}\right]$$

- You can record the maximizing i during your computation of $l^*_{j,t+1}$.
- Note that $\phi_{t+1}(j)$ is the element at the j^{th} row and $(t+1)^{\text{th}}$ column.
- "Given the observations $o_1, ..., o_{t+1}$, and given that you ended up in state j at time $t + 1$ ($S_{t+1} = j$), which state were you most likely to have been in at time t?"

Finally, you can compute the most likely states via *backtracking*.

First, compute the most likely state at time T:

$$s^*_T = \arg\max_i l^*_{iT}$$

- "The value of i that corresponds to the maximum element in the final column is just the state that you would end up in at time T to explain the observations as well as possible."

Then, for each time t from $T - 1$ to 1,

$$s^*_t = \phi_{t+1}(s^*_{t+1})$$

Note: this whole dynamic programming process is called the **Viterbi algorithm**, and the resulting sequence of states $\{s^*_1, ..., s^*_T\}$ is sometimes called the **Viterbi path**.

13.3 Backward Algorithm

Goal: given one or more sequences of observations $\{o_1, ..., o_T\}$, estimate the parameters of the HMM $\{\pi_i, a_{ij}, b_{ik}\}$ that maximize the likelihood of the observed data $P(o_1, ..., o_T)$.

Assume: there is only one sequence of observations (meaning an extra summation over sequences will be omitted), and we know the (fixed) cardinality n of the hidden state space.

Strategy: use the EM algorithm to estimate the CPTs π_i, a_{ij}, and b_{ik}.

In general, for the EM algorithm, you need to compute $P(X_i = x, \text{parents}(X_i) = \pi | V)$ in the E-step so that you can re-estimate $P(X_i = x | \text{parents}(X_i) = \pi)$ in the M-step.

The HMM parameters that you want to estimate are

$$\begin{aligned} \pi_i &= P(S_1 = i) \\ a_{ij} &= P(S_{t+1} = j | S_t = i) \\ b_{ik} &= P(O_t = k | S_t = i) \end{aligned}$$

So in the E-step, you need to compute

$$\begin{aligned} P(S_1 = i | o_1, ..., o_T) \\ P(S_{t+1} = j, S_t = i | o_1, ..., o_T) \\ P(O_t = k, S_t = i | o_1, ..., o_T) = I(o_t, k) P(S_t = i | o_1, ..., o_T) \end{aligned}$$

To compute these probabilities, you need to define one more $n \times T$ matrix.
Analogous to $\alpha_{it} = P(o_1, o_2, ..., o_t, S_t = i)$, which predicted everything up to time t,

$$\beta_{it} = P(o_{t+1}, o_{t+2}, ..., o_T | S_t = i)$$

predicts all of the observations **after** time t **given** that the hidden state at time t is equal to i.

13.3.1 Computing β_{it}

Start from the last column ($t = T$):

$$\begin{aligned} \beta_{iT} &= P(\text{unspecified future } | S_T = i) \\ &= 1 \end{aligned}$$

- β_{it} computes the probability of future observations given $S_t = i$.
- But observations after time T are unspecified. So we set β_{iT} to 1 by definition.

Then, moving from right to left over columns ($t < T$):

$$\begin{aligned}
\beta_{it} &= P(o_{t+1}, ..., o_T | S_t = i) \\
&= \sum_{j=1}^{n} P(S_{t+1} = j, o_{t+1}, ..., o_T | S_t = i) \\
&= \sum_{j=1}^{n} [P(S_{t+1} = j | S_t = i) P(o_{t+1} | S_t = i, S_{t+1} = j) P(o_{t+2}, ..., o_T | S_t = i, S_{t+1} = j, o_{t+1})] \\
&= \sum_{j=1}^{n} [P(S_{t+1} = j | S_t = i) P(o_{t+1} | S_{t+1} = j) P(o_{t+2}, ..., o_T | S_{t+1} = j)] \\
&= \sum_{j=1}^{n} a_{ij} b_{j o_{t+1}} \beta_{j,t+1}
\end{aligned}$$

This **backward algorithm** fills in the β matrix one column at a time from right to left.

14 HMMs and Clustering with GMMs 11/19/20

In the **forward-backward algorithm**, you compute the matrices

$$\begin{aligned}
\alpha_{it} &= P(o_1, ..., o_t, S_t = i) \\
\beta_{it} &= P(o_{t+1}, ..., o_T | S_t = i)
\end{aligned}$$

During the forward pass, you compute

$$\begin{aligned}
\alpha_{i1} &= \pi_i b_{i o_1} \\
\alpha_{j,t+1} &= \sum_{i=1}^{n} \alpha_{it} a_{ij} b_{j o_{t+1}}
\end{aligned}$$

During the backward pass, you compute

$$\begin{aligned}
\beta_{iT} &= 1 \\
\beta_{it} &= \sum_{j=1}^{n} a_{ij} b_{j o_{t+1}} \beta_{j,t+1}
\end{aligned}$$

14.1 EM Algorithm for HMMs

14.1.1 E-Step

Given the α and β matrices, you can compute the posteriors mentioned in the previous lecture.

To compute $P(S_t = i|o_1, ..., o_T)$:

$$\begin{aligned} P(S_t = i|o_1, ..., o_T) &= \frac{P(S_t = i, o_1, ..., o_T)}{P(o_1, ..., o_T)} \\ &\vdots \\ &= \frac{\alpha_{it}\beta_{it}}{\sum_{i'} \alpha_{i't}\beta_{i't}} \end{aligned}$$

To compute $P(S_t = i, S_{t+1} = j|o_1, ..., o_T)$:

$$\begin{aligned} P(S_t = i, S_{t+1} = j|o_1, ..., o_T) &= \frac{P(S_t = i, S_{t+1} = j, o_1, ..., o_T)}{P(o_1, ..., o_T)} \\ &\vdots \\ &= \frac{\alpha_{it}a_{ij}b_{jo_{t+1}}\beta_{j,t+1}}{\sum_{i'} \alpha_{i't}\beta_{i't}} \end{aligned}$$

14.1.2 M-Step

For one sequence of observations:

$$\begin{aligned} \pi_i &\leftarrow P(S_1 = i|o_1, ..., o_T) \\ a_{ij} &\leftarrow \frac{\sum_t P(S_{t+1} = j, S_t = i|o_1, ..., o_T)}{\sum_t P(S_t = i|o_1, ..., o_T)} \\ b_{ik} &\leftarrow \frac{\sum_t I(o_t, k)P(S_t = i|o_1, ..., o_T)}{\sum_t P(S_t = i|o_1, ..., o_T)} \end{aligned}$$

14.2 Multivariate Gaussian Distributions

Motivation: introduce real-valued variables into belief networks.

The probability density function (PDF) over $\mathbb{R}^d$ for a multivariate Gaussian distribution is

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu)^\top \mathbf{\Sigma}^{-1}(\mathbf{x} - \mu)\right\}$$

where μ and $\mathbf{\Sigma}$ are (respectively) the mean and covariance matrix of the distribution.

$$\begin{aligned} \mu &= \mathbb{E}[\mathbf{x}] = \int_{\mathbb{R}^d} P(\mathbf{x})\mathbf{x} \\ \mathbf{\Sigma} &= \mathbb{E}[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^\top] = \int_{\mathbb{R}^d} P(\mathbf{x})(\mathbf{x} - \mu)(\mathbf{x} - \mu)^\top \end{aligned}$$

The distribution peaks at the value at μ and falls off exponentially as you move away from μ.

14.2.1 Properties

1. If $\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$, then
$$\mathbf{Ax} \sim \mathcal{N}(\mathbf{A}\mu, \mathbf{A\Sigma A}^\top)$$
for any linear transformation $\mathbf{A}$.

2. If $\mathbf{x}_i \sim \mathcal{N}(\mu_i, \mathbf{\Sigma}_i)$ are independent random variables in $\mathbb{R}^d$, then any linear combination $\sum_i \alpha_i \mathbf{x}_i$ is normally distributed in $\mathbb{R}^d$.
3. If $P(\mathbf{x})$ is multivariate Gaussian, then all of the marginal distributions $\{P(x_1), P(x_1, x_2), ...\}$ and conditional distributions $\{P(x_1|x_2), P(x_1|x_2, x_3), ...\}$ are also multivariate Gaussian.

14.2.2 Maximum Likelihood Estimation

If $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$ are i.i.d. examples in $\mathbb{R}^d$ and $\mathbf{x}$ is normally distributed, you can estimate $(\mu, \mathbf{\Sigma})$ using maximum likelihood estimation [choosing the $(\mu, \mathbf{\Sigma})$ that maximize the log-likelihood of the data $\mathcal{L} = \sum_{t=1}^T \log P(\mathbf{x}_t)$] as

$$\mu_{\text{MLE}} = \frac{1}{T} \sum_{t=1}^T \mathbf{x}_t$$
$$\mathbf{\Sigma}_{\text{MLE}} = \frac{1}{T} \sum_{t=1}^T (\mathbf{x}_t - \mu_{\text{MLE}}) (\mathbf{x}_t - \mu_{\text{MLE}})^\top$$

14.3 Clustering with GMMs

Goal: infer labels $z \in \{1, ..., k\}$ from inputs $\mathbf{x} \in \mathbb{R}^d$ without any labeled examples.

Strategy: model this via a belief network where the unobserved cluster label $z \in \{1, ..., k\}$ feeds into the observed input $\mathbf{x} \in \mathbb{R}^d$. The conditional probability "tables" are

- $P(Z = i)$ [fraction of data in i^{th} cluster]
- $P(\mathbf{x}|Z = i)$ [multivariate Gaussian $\mathbf{x} \sim \mathcal{N}(\mu_i, \mathbf{\Sigma}_i)$]

Notice that each cluster has its own mean and covariance matrix.

14.3.1 Learning the Parameters of the GMM

If you had complete data $\{(\mathbf{x}_t, z_t)\}_{t=1}^T$, you could estimate the parameters via MLE as

$$P(Z = i) = \frac{1}{T} \sum_{t=1}^T I(z_t, i) = \frac{T_i}{T}$$
$$\mu_i = \frac{\sum_{t=1}^T I(z_t, i) \mathbf{x}_t}{\sum_{t=1}^T I(z_t, i)} = \frac{1}{T_i} \sum_{t=1}^T I(z_t, i) \mathbf{x}_t$$
$$\mathbf{\Sigma}_i = \frac{\sum_{t=1}^T I(z_t, i)(\mathbf{x}_t - \mu_i)(\mathbf{x}_t - \mu_i)^\top}{\sum_{t=1}^T I(z_t, i)} = \frac{1}{T_i} \sum_{t=1}^T I(z_t, i)(\mathbf{x}_t - \mu_i)(\mathbf{x}_t - \mu_i)^\top$$

where $T_i = \sum_{t=1}^T I(z_t, i)$ represents the number of points in the i^{th} cluster.

Given that your data is incomplete, however, you'll have to use the EM algorithm. Let's assume that your data looks like $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$. The E-step will take the following form:

$$\begin{aligned} P(Z = i|\mathbf{x}_t) &= \frac{P(\mathbf{x}_t|Z = i)P(Z = i)}{P(\mathbf{x}_t)} \\ &= \frac{P(\mathbf{x}_t|Z = i)P(Z = i)}{\sum_j P(\mathbf{x}_t|Z = j)P(Z = j)} \end{aligned}$$

And the M-step will look like

$$\begin{aligned} P(Z = i) &\leftarrow \frac{1}{T}\sum_t P(Z = i|\mathbf{x}_t) \\ \mu_i &\leftarrow \frac{\sum_{t=1}^T P(Z = i|\mathbf{x}_t)\mathbf{x}_t}{\sum_{t=1}^T P(Z = i|\mathbf{x}_t)} \\ \mathbf{\Sigma}_i &\leftarrow \frac{\sum_{t=1}^T P(Z = i|\mathbf{x}_t)(\mathbf{x}_t - \mu_i)(\mathbf{x}_t - \mu_i)^\top}{\sum_{t=1}^T P(Z = i|\mathbf{x}_t)} \end{aligned}$$

After running the EM algorithm, you will have estimated the *mean of each cluster* (μ_i) along with *a description of how each cluster falls off around its mean* ($\mathbf{\Sigma}_i$).

15 Reinforcement Learning 11/24/20

15.1 GMMs, cont.

You can model a multimodal distribution as a weighted sum (a *mixture*) of Gaussian distributions, which will typically have as many peaks as it does Gaussians.

15.2 Linear Dynamical Systems

Goal: given sensor measurements, determine the location and bearing of a missile.

You can model the continuous dynamical system as a series of hidden states over time $\mathbf{s}_t \in \mathbb{R}^n$, where each hidden state feeds into the next hidden state and emits an observation $\mathbf{o}_t \in \mathbb{R}^m$.

- The graphical model is the same as that of an HMM.

Let the conditional probability distributions for this belief network be

$$\begin{aligned} \mathbf{s}_1 &\sim \mathcal{N}(\mu_0, \mathbf{\Sigma}_0) && [\mu_0 \in \mathbb{R}^n, \mathbf{\Sigma}_0 \in \mathbb{R}^{n\times n}] \\ \mathbf{s}_{t|t>1} &\sim \mathcal{N}(\mathbf{A}\mathbf{s}_{t-1}, \mathbf{\Sigma}_\mathrm{H}) && [\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{\Sigma}_\mathrm{H} \in \mathbb{R}^{n\times n}] \\ \mathbf{o}_t &\sim \mathcal{N}(\mathbf{B}\mathbf{s}_t, \mathbf{\Sigma}_\mathrm{O}) && [\mathbf{B} \in \mathbb{R}^{m\times n}, \mathbf{\Sigma}_\mathrm{O} \in \mathbb{R}^{m\times m}] \end{aligned}$$

This system is linear because $\mathbb{E}[\mathbf{s}_t|\mathbf{s}_{t-1}] = \mathbf{A}\mathbf{s}_{t-1}$ and $\mathbb{E}[\mathbf{o}_t|\mathbf{s}_t] = \mathbf{B}\mathbf{s}_t$.

To compute $P(\mathbf{s}_t|\mathbf{o}_1, ..., \mathbf{o}_t)$ in this linear dynamical system, you can perform Kalman filtering.

Note: $P(\mathbf{s}_1, \mathbf{o}_1, ..., \mathbf{s}_t, \mathbf{o}_t)$ is multivariate Gaussian, so its marginal and conditional distributions are Gaussian as well. Thus you only need to track the mean and covariance of $P(\mathbf{s}_t|\mathbf{o}_1, ..., \mathbf{o}_t)$:

$$\begin{aligned} \mu_t &= \mathbb{E}[\mathbf{s}_t|\mathbf{o}_1, ..., \mathbf{o}_t] \\ \mathbf{\Sigma}_t &= \mathbb{E}[(\mathbf{s}_t - \mu_t)(\mathbf{s}_t - \mu_t)^\top|\mathbf{o}_1, ..., \mathbf{o}_t] \end{aligned}$$

15.3 Reinforcement Learning

In reinforcement learning (RL), agents learn from experience in their environments. Agents can take actions and observe (perhaps indirectly) states and rewards. The goal is to maximize reward.

Formally, there'll be a Markov decision process (MDP) with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition probabilities $P(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a)$, and a reward function $R(s,s',a)$.

Markov assumptions:

$$P(S_{t+1} = s'|S_t = s, A_t = a) = P(S_{t+1} = s'|S_t = s, A_t = a, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, ...)$$
$$P(S_{t+1} = s'|S_t = s, A_t = a) = P(S_{t+1+\tau} = s'|S_{t+\tau} = s, A_{t+\tau} = a)$$

250A assumptions: $\mathcal{S}$ and $\mathcal{A}$ are discrete and finite, rewards only depend on the state $[R(s,s',a) = R(s)]$, rewards are bounded $[\max_s |R(s)| < \infty]$, and rewards/policies are deterministic.

The most common way to measure long-term return is in a discounted, infinite-horizon sense:

$$\sum_{t=0}^{\infty} \gamma^t r_t$$

where $\gamma \in [0,1)$ is the discount factor and r_t is the reward at time t.

Goal: establish a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected return

$$\mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| s_0 = s\right]$$

16 Markov Decision Processes 12/01/20

16.1 Value Functions

The value of a state under policy π is

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| s_0 = s\right]$$

(It's just the expected **long-term** return if you start in state s and follow policy π.)

16.1.1 Bellman Equation

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}^{\pi}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ... \middle| s_0 = s\right] \\
&= R(s) + \gamma \mathbb{E}^{\pi}\left[R(s_1) + \gamma R(s_2) + ... \middle| s_0 = s\right] \\
&= R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) \mathbb{E}^{\pi}\left[R(s_1) + \gamma R(s_2) + ... \middle| s_1 = s'\right] \\
&= R(s) + \gamma \sum_{s'} P(s'|s,\pi(s)) V^{\pi}(s')
\end{aligned}$$

16.1.2 Action Value Function

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| s_0 = s, a_0 = a \right] \\ &\vdots \\ &= R(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s') \end{aligned}$$

(It's the expected return if you start in state s, take action a, and **thereafter** follow policy π.)

16.1.3 Optimal Value Functions

Let π^* be an optimal policy s.t. $V^{\pi^*}(s) \geq V^\pi(s)$ for all π, s. Then the optimal value functions (which are shared by all optimal policies) are $V^*(s) = V^{\pi^*}(s)$ and $Q^*(s,a) = Q^{\pi^*}(s,a)$.

If you know the optimal action value function:

$$\begin{aligned} V^*(s) &= \max_a [Q^*(s,a)] \\ \pi^*(s) &= \arg\max_a [Q^*(s,a)] \end{aligned}$$

If you know the optimal state value function:

$$\begin{aligned} Q^*(s,a) &= R(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \\ \pi^*(s) &= \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right] \end{aligned}$$

16.2 Algorithms

16.2.1 Policy Evaluation

Goal: compute the state value function $V^\pi(s)$ for some fixed policy π.

The Bellman equation

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s')$$

is a system of n linear equations for n unknowns (because there is an equation for each of the n values of s, and the unknowns are $V^\pi(s)$ for each of the n values of s).

You can solve the linear system as $V^\pi = (I - \gamma P^\pi)^{-1} R$, where V^π is an (unknown) n-dimensional vector of state values, I is the $n \times n$ identity matrix, P^π is the (known) $n \times n$ transition matrix induced by the policy π, and R is a (known) n-dimensional vector of rewards for each state.

16.2.2 Policy Improvement

Goal: given a policy π and its state value function $V^{\pi}(s)$,
compute a policy π' such that $V^{\pi'}(s) \geq V^{\pi}(s)$ for all states s.

Given $Q^{\pi}(s,a)$ for policy π, define the "greedy policy" π' as

$$\begin{aligned}\pi'(s) &= \arg\max_a \left[Q^{\pi}(s,a)\right] \\ &= \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')\right] \\ &= \arg\max_a \left[\sum_{s'} P(s'|s,a) V^{\pi}(s')\right]\end{aligned}$$

Then the greedy policy is the policy we're looking for: $V^{\pi'}(s) \geq V^{\pi}(s)$ for all states s.

- "If it's better to choose action a in state s before following π,
  then it's always better to choose action a in state s."

16.2.3 Policy Iteration

Goal: compute an optimal policy $\pi^*(s)$.

1. Choose an initial policy $\pi : \mathcal{S} \to \mathcal{A}$.
2. Repeat until convergence: [25]
   - Compute the action value function $Q^{\pi}(s,a)$ (*by first computing the state value function*).
   - Compute the greedy policy $\pi'(s) = \arg\max_a \left[Q^{\pi}(s,a)\right]$, and use it as the new π.

17 Value Iteration 12/03/20

If you know the optimal value function $V^*(s)$, you can get an optimal policy π^* as

$$\pi^*(s) = \arg\max_a \left[\sum_{s'} P(s'|s,a) V^*(s')\right]$$

So: an alternative way to get an optimal policy is to first identify the optimal state value function, and then perform the above calculation.

17.1 Bellman Optimality Equation

$$\begin{aligned}V^*(s) &= \max_a \left[Q^*(s,a)\right] \\ &= \max_a \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s')\right]\end{aligned}$$

[25] *Convergence*: when the state value function doesn't change from one iteration to the next, i.e. $V^{\pi'}(s) = V^{\pi}(s)$ for all states s. At this point, $V^{\pi}(s) = V^*(s)$ for all states s. Note that policy iteration will always converge.

17.2 Value Iteration

Initialize $V_0(s) = 0$ for all states s. Then iterate until convergence:

$$V_{k+1}(s) = \max_a \left[R(s) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right] \text{ for all states } s$$

At this point, you will have $V^*(s)$, which you can use (as previously described) to compute π^*.

18 Temporal Differences 12/08/20

18.1 Model-Free Reinforcement Learning

Goal: learn an optimal policy. If you have a model of the environment (transition probabilities and reward function), you can run policy or value iteration. If you don't, you can either estimate the transition probabilities[26] (model-based approach) or go without them (model-free approach).

18.1.1 Stochastic Approximation Theory

You can estimate the mean μ of a random variable X from i.i.d. samples by initializing your estimate to 0 ($\mu_0 = 0$), then performing the following incremental update after receiving each sample x_t:

$$\begin{aligned} \mu_t &= (1 - \alpha_t)\mu_{t-1} + \alpha_t x_t \\ &= \mu_{t-1} + \alpha_t (x_t - \mu_{t-1}) \end{aligned}$$

where $\alpha_t \in (0, 1)$ is the step size, and $x_t - \mu_{t-1}$ is a "temporal difference." For convergence to the true mean ($\mu_t \to \mathbb{E}[X]$ as $t \to \infty$), it must be the case that $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$.

18.1.2 Model-Free Policy Evaluation

Goal: estimate $V^\pi(s)$ directly from experience, without knowing the transition probabilities $P(s'|s,a)$.

Temporal difference prediction:

1. Initialize $V_0(s) = 0$ for all states s.
2. Choose a starting state s_0 and observe its associated reward r_0.
3. Repeat (*explore the state space and update the $V^\pi(s)$ estimate*):
   (a) Use the fixed policy π to choose an action $\pi(s_t)$ to take from the current state s_t.
   (b) Note the state that you end up in (s_{t+1}) and its associated reward (r_{t+1}).
   (c) Update the V^π estimate for state s_t:

$$V_{t+1}(s_t) = \underbrace{V_t(s_t)}_{\text{previous estimate}} + \underbrace{\alpha(s_t)}_{\text{step size}} \Big[\underbrace{r_t + \gamma V_t(s_{t+1})}_{\text{"sample" of Bellman equation RHS}} - \underbrace{V_t(s_t)}_{\text{previous estimate}} \Big]$$

[26] Note: we will assume that we can obtain the reward function by visiting each of the states.

References

1. Lawrence Saul, Fall 2020 CSE 250A lectures.