6.824 Artificial Intelligence, Fall 1993
Midterm Solutions

Problem 1. Consider objects with n linear attributes where each object assigns each attribute x_i an integer value from 1 to 2^k (k is the number of bits used to represent the integer x_i). We consider conjunctive concepts defined by conditions of the form

$$x_1 \in [l_1,\ u_1] \wedge x_2 \in [l_2,\ u_2] \wedge \ldots \wedge x_n \in [l_n,\ u_n]$$

where each l_i and u_i is an integer in the range $[1,\ 2^k]$ and $l_i \leq u_i$.

part a. The cardinality of the concept space is $(2^k)^{2n}$ so to ensure that with probability $1-\delta$ all concepts consistent with the examples have error rate no larger than ϵ

$$m \geq \frac{1}{\epsilon}(\ln \frac{1}{\delta} + 2kn \ln 2)$$

examples suffice.

part b. These concepts defines hypercubes in n dimensional space, and we know from lecture that the VC-dimension of such concept spaces is $2n$. So

$$m \geq \frac{1}{\epsilon}(4 \log \frac{2}{\delta} + 16n \log \frac{13}{\epsilon})$$

are sufficient for PAC learning.

part c. We want

$$\begin{aligned} m_a &\leq m_b && (0.1) \\ \frac{1}{\epsilon}(\ln \frac{1}{\delta} + 2kn \ln 2) &\leq \frac{1}{\epsilon}(4 \log \frac{2}{\delta} + 16n \log \frac{13}{\epsilon}) && (0.2) \\ k &\leq \frac{4}{2n} - \frac{3 \log \delta}{2n \ln 2} - \frac{8 \log \frac{13}{\epsilon}}{\ln 2} && (0.3) \end{aligned}$$

As n gets large

$$k \leq \frac{8 \log \frac{13}{\epsilon}}{\ln 2}$$

Now is ϵ is $\frac{1}{32}$

$$\begin{aligned} k &\leq \frac{8 \log \frac{13}{2^{-5}}}{\ln 2} && (0.4) \\ &\leq \frac{8}{\ln 2}(\log 13 + 5 \log 2) && (0.5) \\ &\leq 100 && (0.6) \end{aligned}$$

In other words the bound derived from cardinality does better than the VC-dimension bound for problems represented with up to 100 bits. That's surprising...

Problem 2. Suppose that we represent a function as a mixture of Gaussians.

$$f(x) = \sum_{1 \leq j \leq k} a_j e^{\frac{-(x-m_j)^2}{2\sigma_j^2}}$$

We are given the parameters a_i, m_i, and σ_i that define the function f,and a set $\{\langle x_1,\, y_1\rangle,\, \ldots,\, \langle x_n,\, y_n\rangle\}$ of desired input/output pairs.

part a. We have initial values for a_i's, m_i's, and σ_i's, after seeing the example can modify these values according to the difference between the "correct" output y_i and the approximation $f(x_i)$ Gradient descent indicates that we should modify the a_i's, m_i's, and σ_i's according to
$a_i' = a_i + \eta\Delta a_i$ where $\Delta a_i = -\frac{\partial E}{\partial a_i}$
$m_i' = m_i + \eta\Delta m_i$ where $\Delta m_i = -\frac{\partial E}{\partial m_i}$
and $\sigma_i' = \sigma_i + \eta\Delta\sigma_i$ where $\Delta\sigma_i = -\frac{\partial E}{\partial \sigma_i}$

$$E = \sum_i (y_i - f(x_i))^2 = \sum_i (y_i - \sum_j a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})^2$$

let $\delta = y - f(x)$

We get

$$\frac{\partial E}{\partial a_j} = \sum_i 2(y_i - a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})(-e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}}) \tag{0.7}$$

$$= \sum_i -2\delta e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}} \tag{0.8}$$

$$\frac{\partial E}{\partial m_j} = \sum_i 2(y_i - a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})(-a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})(-\frac{2(x_i-m_j)}{2\sigma_j^2})(-1) \tag{0.9}$$

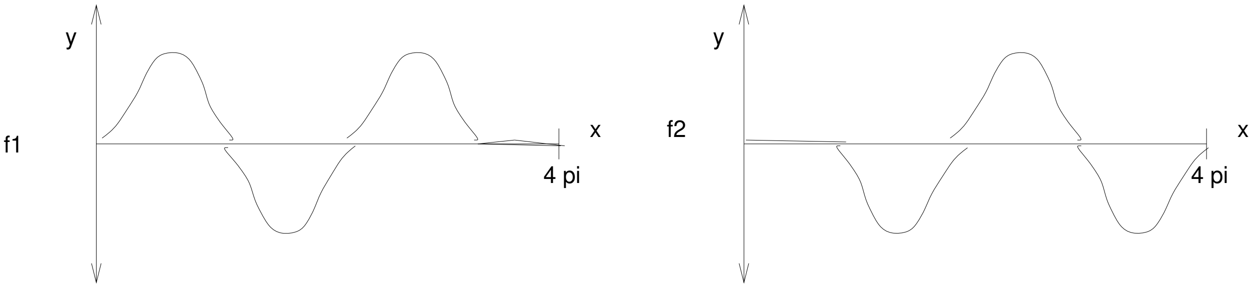
$$= \sum_i -2a_j\delta(\frac{2(x_i-m_j)}{\sigma_j^2})e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}} \tag{0.10}$$

and

$$\frac{\partial E}{\partial \sigma_j} = \sum_i 2(y_i - a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})(-a_j e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}})(-\frac{(x_i-m_j)^2}{2})(-2\sigma_j^{-3}) \quad (0.11)$$

$$= \sum_i -2a_j\delta(\frac{(x_i-m_j)^2}{\sigma_j^3})e^{-\frac{(x_i-m_j)^2}{2\sigma_j^2}} \quad (0.12)$$

part b. The following are two different functions which are a composition of 3 Gaussians that approximate a sin function.



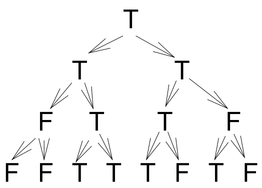
part c. Gradient descent is a form of least squares fitting of data in whic we minimize the error of the data. Say that the data given is the function value with Gaussian noise ϵ, then we want to maximize

$$Pr(D|f) = \prod_i e^{-\frac{\epsilon_i^2}{2\sigma^2}} = e^{-\frac{\sum_i \epsilon_i^2}{2\sigma^2}}$$

This corresponds to minimizing $\sum \epsilon_i^2$, which is exactly the error we tried to reduce in part a, and it is a maximum likelihood approach.

Problem 3. Consider a Bayesian network which is an n level binary tree with arcs directed from parents to children and ending in the 2^n leaves of the tree. Each node in this tree is a Boolean variable where the probability that a node has the same value as its parent is $(1-\epsilon)$ with $0 < \epsilon < \frac{1}{2}$. The prior probability that the root node is true is $\frac{1}{2}$. Suppose that we observe that all the leaf nodes have value T. We use P_n to represent a node that is n arcs above the leaf nodes.

part a. Consider the execution of a tree network shown below.



$$Pr(E) = \frac{1}{2}(1-\epsilon)^{10}\epsilon^4$$

and in general for any execution E

$$Pr(E) = \frac{1}{2}(1-\epsilon)^{nochanges}\epsilon^{changes}$$

part b. Recall that $\beta_{P_n}(T)$ is the probability of the oservations below the node P_n given that $P_n = T$. $\beta_{P_n}(F)$ is defined similarly. For each n these numbers can be written as a polynomials in ϵ. Since ϵ is in the interval $(0,\ \frac{1}{2})$ we take the order of a polynomial in ϵ to be *smallest* exponent of ϵ in a nonzero term. For example, the order of $1+\epsilon$ is 0 and the order of $3\epsilon^2+\epsilon^3$ is 2. The polynomial 0, i.e., the function that is always 0, should be considered to have infinite order $(0 = \epsilon^\infty)$.

$\beta_{P_n}(T)$ and $\beta_{P_n}(F)$ are computed from the following recurrences

$\beta_{leaf}(T) = 1$ this has polynomial order 0
$\beta_{P_n}(T) = ((1-\epsilon)\beta_{P_n}(T) + \epsilon\beta_{P_n}(F))^2$

Hypothesise that the polynomial order of $\beta_{P_n}(T)$ is 0. We can see from the first term above that a 0 order term remains after applying the recursion. So $\beta_{P_n}(T)$ is order 0.

$\beta_{leaf}(F) = 0$ this has polynomial order ∞
$\beta_{P_n}(F) = ((1-\epsilon)\beta_{P_n}(F) + \epsilon\beta_{P_n}(T))^2$

Now since we know that the polynomial order of $\beta_{P_n}(T)$ is 0 the second term of the recursion above gives an order of 2. We can then verify that the recursion preserves this polynomial order. So $\beta_{P_0}(F)$ is order ∞ and for $n > 0$ $\beta_{P_n}(F)$ is order 2.

part c. Now consider a set of observations O at the leaves of a tree with n levels of arcs and root node n. Consider all the possible ways that the network can be executed starting with root value $P_n = T$ and ending in observed values O (an arbitrary set of observations). A given execution corresponds to a particular set of values for all the internal nodes of the tree. For a given execution we consider the number of "changes" in the execution, i.e., the number of nodes whose values are different from the value of its parent.

We define $B_{P_n}(T) = min_{e \in executions}$number of changes of e.

The order of the polynomial for $\beta_{P_n}(T)$ is always equal to $B_{P_n}(T)$.

From the hint we know that

$$\beta_{P_n}(T) \quad = \quad Pr(O|P_n = T) \qquad (0.13)$$

$$= \sum_{e \in executions} Pr(E|P_n = T) \tag{0.14}$$

$$= \sum_{e \in executions} \frac{1}{2}(1-\epsilon)^{nochanges}\epsilon^{changes} \tag{0.15}$$

Note that the execution with minimum change is included in the summation above. Say this execution has m changes That means the the polynomial has a term $(1 - \epsilon)^{nochanges}\epsilon^m$, so the smallest polynomial term in $\beta_{P_n}(T)$ is ϵ^m. In other words, the polynomial order of $\beta_{P_n}(T)$ is m.

Problem 4. Suppose we are given a tree structured Bayesian network with observed values at some of the leaves. The tree may have nonuniform depth. We want to find the most likely execution of that network given the observations. We assume that the Bayesian network is a binary tree. Every node other than the leaves has exactly two children. We suppose that the Bayesian network and the observations O are represented by assertions in a data base of the following forms.

- The assertion `(LEFT-CHILD` n m`)` means that m is the left child of n.
- The assertion `(RIGHT-CHILD` n m`)` means that m is the right child of n.
- The assertion `(CONDITIONAL-COMPLEXITY` X x Y y w`)` means that Y is a child node of the node X and $w = -\log_2 P_r(Y = y \mid X = x)$.
- The assertion `(OBSERVED` Z z`)` means that Z is a leaf node which has been observed to have value z.

To find the most likely execution given the data we let E_X be the set of possible executions of the nodes below X, i.e., the set of all possible assignments of values to the nodes below X. For a given node X and possible value x for X we define $\delta_X(x)$ as follows.

$$\delta_X(x) = \max_{e \in E_x} P_r(e \wedge O | X = x)$$

In English, $\delta_X(x)$ is the probability of the most likely execution e of the nodes at or below X such that e is consistent with the observations O. If Z is a leaf node observed to have value z then $\delta_Z(z) = 1$. If Z is a leaf node and w is some possible value for Z other than the observed value then $\delta_Z(w) = 0$. We define $C_X(x)$ to be $-\log_2 \delta_X(x)$. Note that if w is a possible value for Z other than the observed value for z then $C_Z(w) = \infty$. In this case we can ignore the value w for Z. If z is the observed value for leaf X then $C_Z(z) = 0$.

We wish to infer assertions of the following form

- The assertion ($\leq$ (C X x) w) means that $C_X(x) \leq w$

part a. The inference rules inferring the complexity bounds are

$$\frac{\begin{array}{l}\texttt{(LEFT-CHILD } X\ Y\texttt{)}\\ \texttt{(RIGHT-CHILD } X\ Z\texttt{)}\\ \texttt{(}\leq\texttt{ (C } Z\ z\texttt{) } w_1\texttt{)}\\ \texttt{(CONDITIONAL-COMPLEXITY } X\ x\ Z\ z\ w_2\texttt{)}\\ \texttt{(}\leq\texttt{ (C } Y\ y\texttt{) } w_3\texttt{)}\\ \texttt{(CONDITIONAL-COMPLEXITY } X\ x\ Y\ y\ w_4\texttt{)}\end{array}}{\texttt{(}\leq\texttt{ (C } X\ x\texttt{) (+ } w_1\ w_2\ w_3\ w_4\texttt{))}}$$

$$\frac{\texttt{(OBSERVED } Z\ z\texttt{)}}{\texttt{(}\leq\texttt{ (C } Z\ z\texttt{) 0)}}$$

part b. Assume that there are n nodes in the network and that each node has d possible values.

The number of prefix firing of the above rules is $O(nd^2)$ (the order of the antecedants of the first rule is important in this analysis.) The number of bound assertions the Dijkstra tightest bounds algorithm will make is $O(nd)$. So the total running time is $O(nd^2 + nd \log nd)$.