CS472 Foundations of Artificial Intelligence
Fall 2000
Assignment 5 Solutions

1. **Decision Trees**

   (a) Let's use HS for "Hockey Skills", HC for "Hair Color" and FM for "Favourite Music". Following precisely the DESISION-TREE-LEARNING algorithm on p.537 and the instructions in chapter 18.4 in R&N, we procede as follows:

   i. Compute the information gain of all attributes in order to make the heuristic decision in the first call of CHOOSE-ATTRIBUTE:

$$\begin{aligned} Gain(HS) &= I(\tfrac{3}{8},\tfrac{5}{8}) - Remainder(HS) = \\ &= \underbrace{-\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}}_{I(\frac{3}{8},\frac{5}{8})} - \underbrace{(\frac{3}{8}I(\frac{1}{3},\frac{2}{3}) + \frac{5}{8}I(\frac{2}{5},\frac{3}{5}))}_{Remainder(HS)} = \\ &= 0.95443 - (0.34436 + 0.60684) = 0.00322 \end{aligned}$$

$$\begin{aligned} Gain(HC) &= I(\tfrac{3}{8},\tfrac{5}{8}) - Remainder(HC) = \\ &= 0.95443 - (\tfrac{4}{8}I(\tfrac{2}{4},\tfrac{2}{4}) + \tfrac{1}{8}I(1,0) + \tfrac{3}{8}I(0,1)) = 0.95443 - 0.50000 = 0.45443 \end{aligned}$$

$$\begin{aligned} Gain(FM) &= I(\tfrac{3}{8},\tfrac{5}{8}) - Remainder(FM) = \\ &= 0.95443 - (\tfrac{3}{8}I(0,1) + \tfrac{5}{8}I(\tfrac{3}{5},\tfrac{2}{5})) = 0.95443 - 0.60684 = 0.34759 \end{aligned}$$

   So we choose HC as our first attribute to split on, since it has the largest information gain.

   ii. The "red" branch is homogenous w.r.t. class, so we create a leaf there with class value "Pass". Similarly, we create a leaf node with class value "Fail" along the "brown" branch. Now we need to compute the information gain of HS and FM attributes in order to make the heuristic decision along the "blond" branch in the subsequent call of CHOOSE-ATTRIBUTE:

$$\begin{aligned} Gain(HS) &= I(\tfrac{2}{4},\tfrac{2}{4}) - Remainder(HS) = 1 - (\tfrac{2}{4}I(\tfrac{1}{2},\tfrac{1}{2}) + \tfrac{2}{4}I(\tfrac{1}{2},\tfrac{1}{2})) = 1 - (0.5 + 0.5) = 0 \\ Gain(FM) &= I(\tfrac{2}{4},\tfrac{2}{4}) - Remainder(FM) = 1 - (\tfrac{2}{4}I(0,1) + \tfrac{2}{4}I(1,0)) = 1 \end{aligned}$$

   The final tree is presented on figure 1.

   (b) The best answer here is to state that choosing a particular attribute with maximum gain can prevent finding a (possibly) small set of attributes, which are able to do the classification on their own, but have relatively low information gain separately. Considering this argument and a reasonably big set of examples, we can generalize the method to produce trees, an order of magnitude bigger than the minimum tree possible.

   Some other solutions which are accepted, but technically are not quite right, are (1) voting functions and (2) bunch of irrelevant attributes. Note that the primary reason for them not being right is that we are neither able to place functions (or random variables with specific possibilities) in the leafs of a decision tree, nor we can do complex checks over the edges.

Grader's comments: Minor mistakes were noted during the grading of (a). A more common one is assuming the $Gain(A) = 1 + ...$ where it is 0.9... as shown above (-3 points). Some students missed

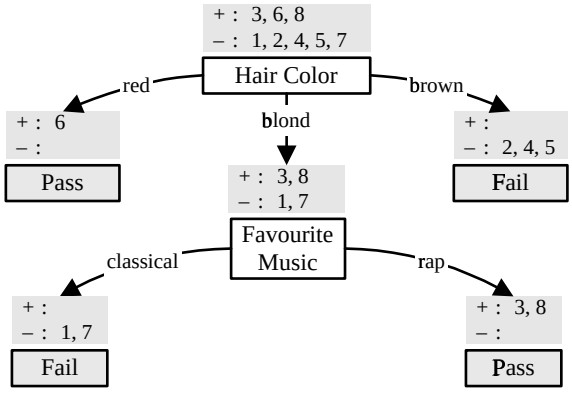

Figure 1: The final decision tree.

to calculate the gains for the second subdivision of the tree (following the "blond" branch) and were penalized 2 points. Some other minor issues were penalized 1 point. In the rare cases where nearly half of the solution was missing, 10 points were substracted.

The grading of (b) was very flexible, although there were some significant mistakes, namely: Assuming that a tree with a larger branching factor is bigger (-3 points); providing a specific solution to show that the algorithm can produce a tree *a bit* bigger than the minimum possible, without showing a way to scale out to much bigger trees (-2 points). Some very unclear answers were penalized 1 point.

2. **Computational Learning Theory**

(a) The first step is to show that k-CNF is learnable, i.e. that any function in this form can be approximated after seeing a reasonable number of examples. To do this we need to calculate the number of hypotheses in the language and use that to determine the sample complexity according to equation 18.2 in R&N.

Because each literal can be positive or negative in a clause, the number of clauses with k literals is $2^k\binom{n}{k} \leq 2^k n^k$. Each clause can either appear ot be absent in a formula, therefore $|k-\text{CNF}| = 2^{2^k\binom{n}{k}} \leq 2^{2^k n^k}$. Hence the sample complexity for k-CNF is $m \geq \frac{1}{\epsilon}(\ln\frac{1}{\delta} + \ln|k-\text{CNF}|) = O(n^k)$.

The second step is finding an efficient algorithm that returns a k-CNF formula that is consistent with the training example. The following algorithm does this and it can be easily seen to run in polynomial time:

H = conjunction of all k-length clauses
for each positive example e
  for each clause c in H
    if e falsify c
      delete c from H;
return$H$;

Alternate solution: Let $\Phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a k-CNF formula, where $C_i = t_{i1} \vee t_{i2} \vee \ldots \vee t_{ik}$ are the m disjunction clauses. We suppose that there are n different variables $x_1, x_2, \ldots, x_n$ and that each term $t \in \{t_{ij} : i \in [1,m], j \in [1,k]\}$ is equal to either x_l or $\bar{x}_l$ for some $l \in [1,n]$.

Now, if we take the negation of the i^{th} clause $\neg C_i$, apply DeMorgan's law and get rid of all double negations, we get a conjunction K_i of k literals of n attributes, which matches exactly

the definition of "test" in section 18.6 of R&N (Learning Decision Lists). Further, if the test K_i succeedes, it is clearly the case that the decision on Φ is "No". All this implies that it is appropriate to construct a "decision list" equivalent of our formula Φ in the form on figure 2.

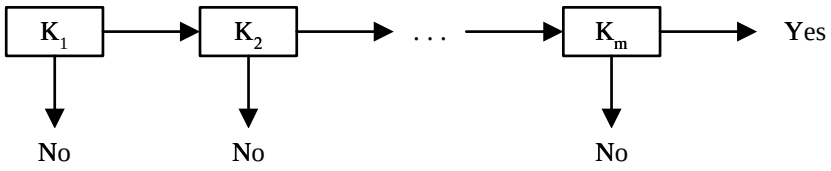

Figure 2: Decision list, representing the formula Φ.

Because we defined a translation from k-CNF to k-DL(n) (which is PAC-learnable), all formulas Φ in k-CNF are PAC-learnable.

(b) Suppose we draw m examples. Each example has n input features plus its classification, so there are 2^{n+1} distinct input/output examples to choose from. For each example, there is exactly one contradictory example, namely the example eith the same imput features but with the opposite classification. Thus, the probability of finding no contradiction is:

$$\frac{\text{number of sequences of } m \text{ non-contradicting examples}}{\text{number of sequences of } m \text{ examples}} = \frac{2^{n+1} \times (2^{n+1}-1) \times \ldots \times (2^{n+1}-m+1)}{2^{m(n+1)}} =$$

$$= \frac{2^{n+1}!}{(2^{n+1}-m)! \times 2^{m(n+1)}}$$

For $n = 10$ with 2048 possible examples, a contradiction becomes likely with probability $> \frac{1}{2}$ after 54 examples have been drawn.

Grader's comments: Many students had trouble with this question, in part because the problem description wasn't totally clear and in part because the book's treatment of decision lists wasn't very strong. For those of you who haven't seen the hints posted on the webpage, you should look at them - they should make this problem seem more understandable. Most people got 15/20 for showing that they understood what the PAC property means and providing a bound on $ln(|k - \text{CNF}|)$ of $O(n^k)$, even if the analysis for k-CNF learning wasn't complete. Very few people actually provided an algorithm for k-CNF learning, so I made that part extra credit: for an algorithm which more or less copied the decision-list learning algorithm from the book, I usually gave 5 extra points; for an algorithm like the one present here, I gave 10 extra points.

Part (b) proved to be too hard for almost everyone, so I also made it extra credit. (You should all consider taking a probability course!) Most people tried to use the formula in the book for the number of examples required for an algorithm to be PAC. This number has nothing to do with the number of examples required before the probabiliy of reaching a contradiction becomes 0.5 - it just tells us whether an *algorithm* is PAC, but says nothing whatsoever about the nature of the examples seen. Anyway, I gave 4 points to anyone who tried this approach. For people who tried to derive a probabilistic formula, I gave from 5 to 10 extra points, depending on the quality of the answer.

3. **Neural Network Learning**

(a) Two different networks, both computing the boolean XOR function are presented on figure 3. Input and output values are in standard notation, i.e. 1 for *True* and 0 for *False*. All units use the "Step" activation function but possibly with different thresholds.

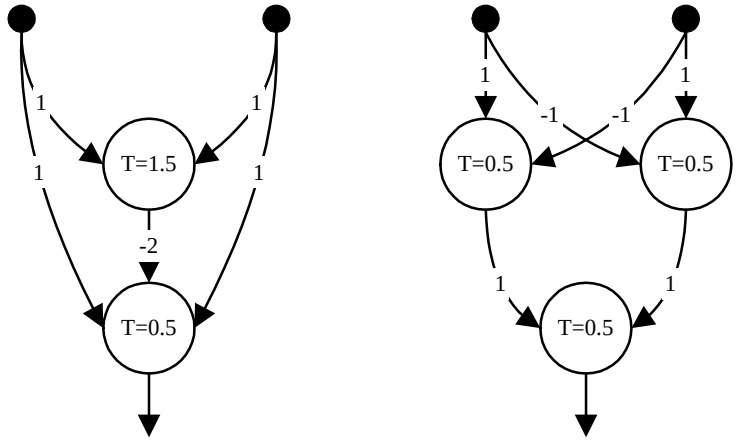

Figure 3: Two examples of XOR neural networks.

(b) Following precisely the BACK-PROP-UPDATE algorithm on p.581 of R&N (which uses stochastic back-propagation, updating the weights after each example) we obtain the following:

i. Example 1

- $O \leftarrow$ RUN-NETWORK$(network, I^e)$

$$\begin{aligned} a_1 &= 0 \\ a_2 &= 0 \\ a_3 &= -1 \\ a_4 &= \sigma(a_1 w_{14} + a_2 w_{24} + a_3 w_{34}) = \sigma(-2) = 0.11920 \\ a_5 &= \sigma(a_1 w_{15} + a_2 w_{25} + a_3 w_{35}) = \sigma(2) = 0.88080 \\ O = a_6 &= \sigma(a_3 w_{36} + a_4 w_{46} + a_5 w_{56}) = \sigma(-2 + 0.23841 - 1.76159) = 0.02866 \end{aligned}$$

- $Err^e \leftarrow T^e - O = 1 - 0.02866 = 0.97134$
- $w_{j6} \leftarrow w_{j6} + a_j \times Err^e \times \sigma'(a_3 w_{36} + a_4 w_{46} + a_5 w_{56})$

$$\begin{aligned} \Delta_6 &= Err^e \times \sigma'(a_3 w_{36} + a_4 w_{46} + a_5 w_{56}) = 0.97134 \times 0.02783 = 0.02704 \\ w_{36} &= w_{36} + a3 \times \Delta_6 = 2 + (-1) \times 0.02704 = 1.97296 \\ w_{46} &= w_{46} + a4 \times \Delta_6 = 2 + 0.11920 \times 0.02704 = 2.00322 \\ w_{56} &= w_{56} + a5 \times \Delta_6 = -2 + 0.88080 \times 0.02704 = -1.97618 \end{aligned}$$

- $\Delta_j \leftarrow \sigma'(in_j) \times w_{j6} \times \Delta_6$

$$\begin{aligned} \Delta_3 &= \sigma'(-1) \times 1.97296 \times 0.02704 = 0.01048 \\ \Delta_4 &= \sigma'(-2) \times 2.00322 \times 0.02704 = 0.00569 \\ \Delta_5 &= \sigma'(2) \times (-1.97618) \times 0.02704 = -0.00561 \end{aligned}$$

- $w_{kj} \leftarrow w_{kj} + a_k \times \Delta_j$

$$\begin{aligned} w_{14} &= 2 + 0 \times 0.00569 = 2 \\ w_{24} &= -2 + 0 \times 0.00569 = -2 \\ w_{34} &= 2 + (-1) \times 0.00569 = 1.99431 \\ w_{15} &= 1 + 0 \times (-0.00561) = 1 \\ w_{25} &= -1 + 0 \times (-0.00561) = -1 \\ w_{35} &= -2 + (-1) \times (-0.00561) = -1.99439 \end{aligned}$$

Some students may just save the weight changes and apply them at the end.

ii. Example 2

- $O \leftarrow$ RUN-NETWORK$(network, I^e)$

$$\begin{aligned} a_1 &= 0 \\ a_2 &= 1 \\ a_3 &= -1 \\ a_4 &= \sigma(a_1 w_{14} + a_2 w_{24} + a_3 w_{34}) = 0.01809 \\ a_5 &= \sigma(a_1 w_{15} + a_2 w_{25} + a_3 w_{35}) = 0.73216 \\ O = a_6 &= \sigma(a_3 w_{36} + a_4 w_{46} + a_5 w_{56}) = 0.03281 \end{aligned}$$

- $Err^e \leftarrow T^e - O = -0.03281$
- $w_{j6} \leftarrow w_{j6} + a_j \times Err^e \times \sigma'(a_3 w_{36} + a_4 w_{46} + a_5 w_{56})$

$$\begin{aligned} \Delta_6 &= Err^e \times \sigma'(a_3 w_{36} + a_4 w_{46} + a_5 w_{56}) = -0.00104 \\ w_{36} &= w_{36} + a3 \times \Delta_6 = 1.97400 \\ w_{46} &= w_{46} + a4 \times \Delta_6 = 2.00320 \\ w_{56} &= w_{56} + a5 \times \Delta_6 = -1.97694 \end{aligned}$$

- $\Delta_j \leftarrow \sigma'(in_j) \times w_{j6} \times \Delta_6$

$$\begin{aligned} \Delta_3 &= -0.00040 \\ \Delta_4 &= -0.00004 \\ \Delta_5 &= 0.00040 \end{aligned}$$

- $w_{kj} \leftarrow w_{kj} + a_k \times \Delta_j$

$$\begin{aligned} w_{14} &= 2 \\ w_{24} &= -2.00004 \\ w_{34} &= 1.99435 \\ w_{15} &= 1 \\ w_{25} &= -0.99960 \\ w_{35} &= -1.99480 \end{aligned}$$

Grading code:

- *1 Need to specify all the weights in the neural net.
- *2 Need to specify what activation function is used.
- *3 Need to show step-by-step workings for backpropagation derivation.
- *4 Student only showed final answer without providing any clues on how the final answers are derived.
- *5 Weights and threhold values do not work out to give XOR function.
- *6 Students need to show the formulas that they applied.
- *7 The calculations are incorrect due to either careless mistakes or wrong intermediate steps.
- *8 Do not set threshold at boundary conditions. Suppose you have two inputs to a given neural net and the weights are both 1. The threshold for the activation of the node should not be 0, 1 or 2 as far as possible to avoid ambiguity of the final output.

Grader's comments: For part (a) the most common mistake was to miss out specifying the activation function (-2 points). For part (b) to get full credits in this part, students have to provide all the derivation steps on top of the final answers to the weight change. Many students only gave

spreadsheet printouts or just a table showing the final answers without any intermediate steps. Very heavy penalty is given to these students if their final answers are incorrect. Another common mistake is to ignore the updates of the weights just because the changes are minimal. The update is necessary otherwise the weight values will not converge since most of the changes are gradual.

4. **Comparison of Learning Algorithms**

   (a) Decision-trees are *designed* to be as fast as possible in the sense that you check conditions only in the order of the depth of the tree, but neural networks are amenable to parallel execution, so depending on the hardware available, either of these might be the best choice.

   (b) They definitely do *not* want a neural network, since the connection between hidden layer nodes and easily understandable concepts is usually hard to see. The case-based system can provide a list of the cases considered and their similarity rankings, allowing the traders to make their own explanations. But the decision tree would probably be best, as the exact sequence of attribute-decisions can be readily shown.

   (c) The decision tree basically does *not*, in that you just throw the new data in with the old and make a new tree from scratch. The neural net, since it processed an example at a time, can easily accommodate the new data, acting in effect as if the training had simply been suspended but not stopped. Note, however, that it will not necessarily produce the same network as if all the data were processed together. The easiest is the case-based system, for which we just add in the new cases!

   Grader's comments: Most students performed extremely well on this problem. Some minor mistakes were encountered. In (a) some stated that neural networks are faster, but not mentioning under what conditions (parallel execution in this case, -1 point). There were a bunch of unclear answers that didn't take a possition in stating which approach is actually best (-2 points). Those that neither mentioned decision trees nor parallel execution for neural networks were penalized 3 points. There were a few very wrong answers, choosing kNN (-4 points). There were almost no problems with (b) and (c). Again some students provided quite enough explanations in (c), but didn't choose a side to stand on (-1 point).

5. **Perceptron Learning**

   (a)

   (b) No it is not perceptron learnable. If you examine the vector space in three dimensions, the positive instances are a line in space. So it cannot be divided into classes by a plane in three dimensional space. So it is not linearly separable. This means it is not preceptron learnable.

   (c) Advanatages of local encoding: analog inputs can be directly represented without any preprocessing, number of inputs to the neuron can be exponentially smaller than a corresponding distributed encoding, etc.

   Advantages of distributed encoding: not all atributes are continuous (e.g. hair color - red, blonde, dark but none is between the other two), inputs are either 1 or 0 this can reduce computation time for learning, a matching between attributes and results can be more human readable (i.e. something with attributes a,b, and not c is a + but someting without a and c but has b is a -), since the values are two valued the neural net can be combined with a formal reasoning system easier (each input corresponds to a proposition of the reasoner), etc.

   You would use local encoding to deal with continuous data (such as tempature), and a distributed encoding to deal with discrete data (such as having / not having an attribute). Note that

a feature vector can actually combine the two encodings (e.g input from analog sensors is represented in 0-1 real values, and input from a higher reasoner such as a knowledge base would be distributed).