CPSC 625-600 Homework #4 + Program #3

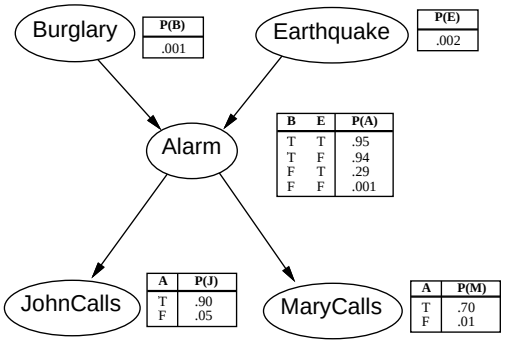
Total: 200 pts

Yoonsuck Choe

April 2, 2020

1 Uncertainty and Probabilistic Reasoning

Question 1, Written (7 pts): Given the belief network as shown below, calculate the joint probability $P(JohnCalls, \neg MaryCalls, Alarm, Earthquake, \neg Burglary)$.



Question 2, Written (10 pts): Given the same domain as above, consider constructing a belief network using the belief network construction algorithm. If the node ordering was $Alarm, MaryCalls, Burglary$ how would the resulting network look like with these three nodes? (1) Show the dependancies using directed edges and (2) for each case, explain why the edge is needed. For example, to determine whether arrows are needed from A to B, check if $P(B|A) = P(B)$, $P(B|A, C) = P(B|C)$, etc.

Question 3, Written (7 pts): Explain why the following inequality holds, in terms of causes and effects. Note: You do not need to compute the probabilities using the bayesian network above.

$$P(Earthquake|MaryCalls) > P(Earthquake|MaryCalls, Burglary)$$

2 Learning

2.1 Decision Tree Learning

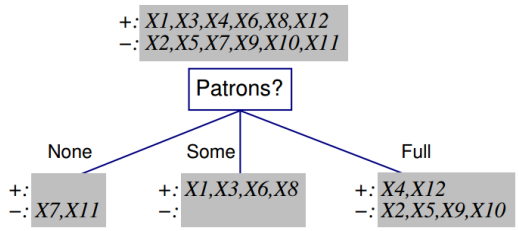
Consider the following set of examples where you are trying to make a decision whether to buy a computer or not given the specifications in terms of the attributes OS, Type, and Graphics.

Example#	OS	Type	Graphics	Decision
1	Windows	Laptop	Nvidia	Y
2	Linux	Desktop	Intel	Y
3	Linux	Desktop	Intel	Y
4	Windows	Desktop	Nvidia	Y
5	Linux	Laptop	Intel	Y
6	Windows	Desktop	AMD	Y
7	MacOS	Desktop	Nvidia	N
8	Windows	Desktop	Intel	N
9	MacOS	Laptop	AMD	N
10	Windows	Laptop	AMD	N

Question 4, Written (17 pts): (1) For each attribute (OS, Type, and Graphics), draw a depth-one decision tree (one attribute tested). See slide06.pdf, page 13 for an example. 7pt (2) For each of the cases above, calculate the information gain. 7pt (3) Based on the information gain, which attribute would you choose first? Explain why. 3pt

Question 5, Written (12 pts): After the first attribute is selected, there would be clear yes and clear no, and examples that remain. With the examples that remain, repeat the steps from Question 3 and compute the information gain for the two remaining attributes, then determine which attribute needs to be tested next.

Question 6, Programming (20 pts): Write a short program (any language) to compute the information gain given the number of positive and negative samples prior to testing with an attribute, and the number of positive and negative samples after testing with the attribute for each possible value of the attribute. For example, in Matlab, the call would be `infogain([6, 6], [0, 2; 4, 0; 2, 4])` for the Patrons attribute shown in slide06, page 13, figure (a) (also shown below). For LISP, `(infogain '(6 6) '((0 2) (4 0) (2 4)))`, etc.



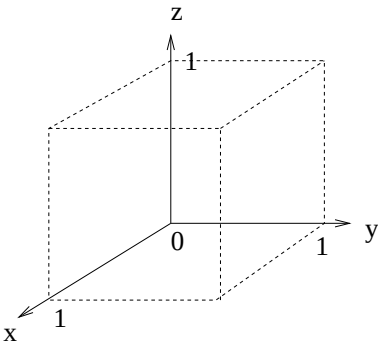
(1) Use your program to compute the information gain for each of the three attributes in the above table. (2) Use your program to compute the information gain for all attributes in the table in slide06, page 9.

Note: You have to check and handle cases where $\log_2(0)$ appears and treat it as 0.

Question 7, Written (15 pts): Can a single perceptron unit solve the following classification problem?: In other words, can the perceptron learning rule find a set of weights to correctly classify all examples? (1) Answer "yes" or "no" to the question, (2) draw a geometric illustration of the problem in 3D and (3) justify your reasoning.

(Note: for 3D input, the decision boundary is a flat plane.)

Input x	Input y	Input z	Class
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Question 8, Programming (30 pts): (1) Write a perceptron program, from scratch, in any language (this is a simple program, so the core part would be roughly less than 20 lines of code). Test it for AND, OR, and XOR functions. Write a plotting function to show the decision boundary (see slide06, page 37 for a hint; you can derive the line equation from the network weights). (2) For each function, run multiple training sessions with different random initial weights. Report the results, including where the final decision boundary was for each session. (Note: XOR would not terminate, so stop it after a few steps.) (3) For the AND and OR problems, is it possible to have the perceptron make zero error without training? Assume the weights are randomly initialized. Explain why or why not. Observing the decision bounadaries throughout the training session can give you some insight.

* You may hard-encode the training set in your code.

Question 9, Written (12 pts): Show how we can implement the XOR function linking up three perceptron units. You don't need to code anything. Graphically illustrate your idea, both (1) neural network topology, and (2) decision boundary of each unit in the input space.

Question 10, Written (10 pts): Given $E(w) = (w+1)(w-1)(w-3)(w-4)$, derive a gradient descent learning rule to adjust w. Basically, you need to find:

$$\Delta w = -\alpha \frac{dE}{dw}$$

Question 11, Programming (20 pts): Write a short program (any language) to simulate the gradient descent steps in the previous problem $E(w) = (w+1)(w-1)(w-3)(w-4)$ (Question 11). Given any arbitraty initial w value between -2 and 6 it is guaranteed to find the local minima. Plug in the initial w value to compute Δw, and use the updated w value, plug it in to compute Δw, etc. Plot the w values over each iteration, overlaid over the plot for the function $E(w)$. Experiment with different learning rates η. Show the plots (range of w values from -2 to 6).

Question 12, Written (20 pts): Experiment with `http://playground.tensorflow.org`. Configure the network and train the model on the web site. For all four data sets, (1) Find the appropriate number of hidden layers, number of neurons for each layer, input features, and train by pressing on the play button. (2) Once the data is correctly learnedm take a screenshot of the entire screen and include in your report. (3) For the spiral data, present two solutions, one with only x_1 and x_2 as the input, and the other with x_1, x_2, x_1^2, x_2^2 as input.

Question 13, Written (20 pts):

In SOM, given an input vector $\vec{x}$ and the best matching unit index $i(\vec{x})$, the learning rule for the reference vector for unit j is:

$$\vec{w}_j \leftarrow \vec{w}_j + \eta h(j, i(\vec{x}))(\vec{x} - \vec{w}_j)$$

1. The learning rate is fixed $\eta = 1$.

2. Let $h(j, i(\vec{x})) = 1$, for the best matching unit $j = i(\vec{x})$, $h(j, i(\vec{x})) = 2/3$ for its immediate neighbor ($j = i(\vec{x}) \pm 1$) and $h(j, i(\vec{x})) = 1/3$ for its second-order neighbor ($j = i(\vec{x}) \pm 2$). For all the rest, $h(j, i(\vec{x})) = 0$.

3. Consider a 1-D SOM with 7 units with the following weight vectors. Plot the vectors and connect them according to the order given below ($\vec{w}_1$ connected to $\vec{w}_2$, etc.).

	w_{i1}	w_{i2}
$\vec{w}_5$	5	4
$\vec{w}_1$	5	6
$\vec{w}_7$	-1	5
$\vec{w}_2$	1	2
$\vec{w}_6$	3	0
$\vec{w}_3$	2	5
$\vec{w}_4$	6	2

4. Given an input vector $\vec{x} = (4, 1)$, plot how the weight vectors change after one iteration of training. Plot in the same graph as (3) above.