Lab 5

Probabilistic Model and Inference

Activity 1: Random Weighted Selection. In this activity, we will study a variant of the Russian Roulette. This algorithm is indeed very useful in a numerous of AI applications to generate latent random vectors, pseudo-random numbers for various distributions, or even random sampling. The key idea is to pick some numbers more often than others. A weight (with respect to the overall weight distribution) can be seen as the probability of a sample to be selected among others. For example, given the array $[6, 4, 1]$ as input, the algorithm returns the index 0 (the index of the first element) with probability 0.6, the index 1 with probability 0.4 and and the index 2 with probability 0.1. The weights in the array can not sum up to 1, but the corresponding probability should be normalized between $[0, 1]$.

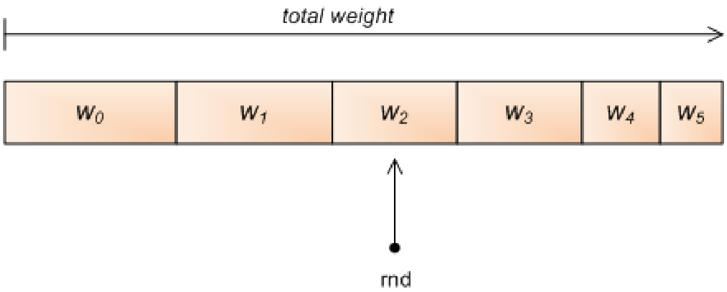

Figure 1: Random Weighted Selection.

The algorithm is specified as follows. As input, the algorithm takes a floating array $[w_0, w_1, ..., w_{n-1}]$ of size n containing weights are sorted in a decreasing order. As output, the algorithm returns the index of the array slot that have been selected. The algorithm is decomposed in several steps:

- Step 1: Calculate the sum of all the weights.
- Step 2: Pick a random number that is 0 or greater and is less than the sum of the weights.
- Step 3: Go through the items one at a time, subtracting their weight from your random number, until you get the item where the random number is less than that item's weight.

The corresponding code is provided in C/C++:

```
float RandomWeightedSelection(float* weights,int n)
{
float weightsum = 0.0f;
for(int i=0; i<n; i++)
{ weightsum += weights[i]; }
// generate a random number between 0 and the sum of the weights
float rnd = random(0,weightssum);
for(int i=0; i<n; i++)
{
if(rnd < weights[i])
{ return i; }
rnd -= weights[i];
}
}
```

A) In the code listing, identify the instructions corresponding to each steps of the provided algorithm.

B) Implement the *Random Weighted Selection* using the language of your choice.

C) Test your implementation on the following input distributions.

- $[5, 3, 2]$
- $[100, 50, 10, 6, 2]$
- $[6, 5, 4, 3, 1]$

D) Run some experimentation to verify that the algorithm performs accurately (ie the resulting values over a large number of calls satisfy the corresponding given weights).

E) Modify your function signature by adding another input array Q as input of the function, and modify the code in such way that the item contained at the selected index in Q are returned instead of the selected index in itself.

Activity 2: Bayes Theorem and Bayesian Networks.. In this activity, we want to study the Bayes' theorem and Bayesian Networks (*no coding task required*). We provide a Bayesian network in the figures. This structure is also defined by the relation *Parent* below. The relation *p* defines the probabilities in this network.

A) Calculate numerically the probability $P(a, b, c)$ in the following network, assuming that $P(a) = 0.25$, $P(b) = 0.35$, $P(c|a, b) = 0.6$

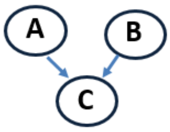

Figure 2: Bayesian Network.

B) We add another node D in the graph defined by an edge $c \rightarrow d$. Calculate numerically the probability $P(a, b, c, d)$ in the following network, assuming that $P(a) = 0.25$, $P(b) = 0.35$, $P(c|a, b) = 0.6$, $P(d|c) = 0.75$.

C) We assume the following bayesian network as input.

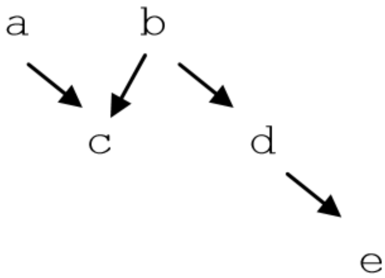

Figure 3: Bayesian Network.

```
parent(a,c).
parent(b,c).
parent(b,d).
parent(d,e).
p(a, 0.1).  means that Probability of "a" is 0.1
p(b, 0.1).
p(c, [a,b], 0.9).  mean that Conditional probability P(c|ab) = 0.9
p(c, [not a, b], 0.6).  means that P(c|¬ab) = 0.6
p(c, [a, not b], 0.8).
p(c, [not a, not b], 0.3).
p(d, [b], 0.9).
p(d, [not b], 0.1).
p(e, [d], 0.1).
p(e, [not d], 0.9).
```

Estimate, without numerical calculation, which of the two probabilities is greater than the other.

Briefly justify your answers.

- $P(c)$ or $P(c|d)$?

- $P(a|c)$ or $P(b|c)$?
- $P(a|c)$ or $P(a|c,e)$?

Extra Activity: The Monty Hall Problem. In this activity, we propose to solve the Monty Hall Problem. The Monty Hall problem is a well-known paradoxical problem in conditional probability and reasoning using Bayes' theorem. The Monty Hall problem is a well-known puzzle in probability derived from an American game show, Let's Make a Deal. The problem was first raised by Steve Selvin in American Statistician in 1975. The game is played like this:

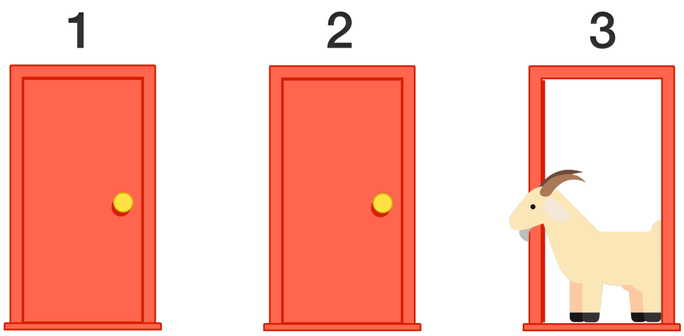

Figure 4: Monty Hall Problem (Image courtesy `https://brilliant.org`).

- The game show set has three doors side-by-side. A prize such as a car or vacation is behind one door, and the other two doors hide a valueless prize called a Zonk; in most discussions of the problem, the Zonk is a goat.
- The contestant chooses one door. We'll assume the contestant has no inside knowledge of which door holds the prize, so the contestant will just make a random choice.
- The smiling host Monty Hall opens one of the other doors, always choosing one that shows a goat, and always offers the contestant a chance to switch their choice to the remaining unopened door.
- The contestant either chooses to switch doors, or opts to stick with the first choice.
- Monty calls for the remaining two doors to open, and the contestant wins whatever is behind their chosen door.

What is the best option? keeping the current configuration or swapping your choice to the other door?