Department of Computer Science and Engineering
(UG Studies)
PES University, Bangalore-560085

Session : Aug - Dec 2017 **Credits :** 0-0-2-0-1	UE14CS405 : Machine Learning Lab
Lab # : 01	Implement Monty Hall Paradox using large scale Random Variate generation.

Learning Objectives

a. Know whether it is better to switch or stay
b. Explain why the probabilities are not equal.

Theory:

The Monty Hall problem is a well-known puzzle in probability derived from an American game show, Let's Make a Deal. (The original 1960s-era show was hosted by Monty Hall, giving this puzzle its name.) Intuition leads many people to get the puzzle wrong, and when the Monty Hall problem is presented in a newspaper or discussion list, it often leads to a lengthy argument in letters-to-the-editor and on message boards.

The game is played like this:

1. The game show set has three doors. 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.
2. 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.
3. 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.
4. The contestant either chooses to switch doors, or opts to stick with the first choice.
5. Monty calls for the remaining two doors to open, and the contestant wins whatever is behind their chosen door.

Let's say a hypothetical contestant chooses door #2. Monty might then open door #1 and offer the

chance to switch to door #3. The contestant switches to door #3, and then we see if the prize is behind #3.

The puzzle is: what is the best strategy for the contestant? Does switching increase the chance of winning the car, decrease it, or make no difference?

The best strategy is to make the switch. It's possible to analyze the situation and figure this out, but instead we'll tackle it by simulating thousands of games and measuring how often each strategy ends up winning.

Approach

Simulating one run of the game is straightforward. A simulate() function that uses Python's random module to pick which door hides the prize, the contestant's initial choice, and which doors Monty chooses to open has to be written. An input parameter controls whether the contestant chooses to switch, and simulate() will then return a Boolean telling whether the contestant's final choice was the winning door.

Part of the reason the problem fools so many people is that in the three-door case the probabilities involved are 1/3 and 1/2, and it's easy to get confused about which probability is relevant. Considering the same game with many more doors makes reasoning about the problem much clearer, so the number of doors is made as a configurable parameter of the simulation script.

CODE

```
#!/usr/bin/env python3
"""Simulate the Monty Hall problem.
"""

import argparse, random
def simulate(num_doors, switch, steps):
    """(int, bool): bool

    Carry out the game for one contestant.  If 'switch' is True,

    the contestant will switch their chosen door when offered the chance.

    Returns a Boolean value telling whether the simulated contestant won.

    """
```

```
# Doors are numbered from 0 up to num_doors-1 (inclusive).
# Randomly choose the door hiding the prize.
winning_door = random.randint(0, num_doors-1)
if steps:
    print('Prize is behind door {}'.format(winning_door+1))

# The contestant picks a random door, too.
choice = random.randint(0, num_doors-1)
if steps:
    print('Contestant chooses door {}'.format(choice+1))

# The host opens all but two doors.
closed_doors = list(range(num_doors))
while len(closed_doors) > 2:
    # Randomly choose a door to open.
    door_to_remove = random.choice(closed_doors)

    # The host will never open the winning door, or the door
    # chosen by the contestant.
    if door_to_remove == winning_door or door_to_remove == choice:
        continue
    # Remove the door from the list of closed doors.
    closed_doors.remove(door_to_remove)
    if steps:
        print('Host opens door {}'.format(door_to_remove+1))
# Does the contestant want to switch their choice?
if switch:
```

```
        if steps:
            print('Contestant switches from door {} '.format(choice+1), end='')
        # There are two closed doors left.  The contestant will never
        # choose the same door, so we'll remove that door as a choice.
        available_doors = list(closed_doors) # Make a copy of the list.
        available_doors.remove(choice)
        # Change choice to the only door available.
        choice = available_doors.pop()
        if steps:
            print('to {}'.format(choice+1))

    # Did the contestant win?
    won = (choice == winning_door)
    if steps:
        if won:
            print('Contestant WON', end='\n\n')
        else:
            print('Contestant LOST', end='\n\n')
    return won

def main():
    # Get command-line arguments
    parser = argparse.ArgumentParser(
        description='simulate the Monty Hall problem')
    parser.add_argument('--doors', default=3, type=int, metavar='int',
                        help='number of doors offered to the contestant')
    parser.add_argument('--trials', default=10000, type=int, metavar='int',
```

```
                    help='number of trials to perform')
parser.add_argument('--steps', default=False, action='store_true',
                    help='display the results of each trial')
args = parser.parse_args()

print('Simulating {} trials...'.format(args.trials))

# Carry out the trials
winning_non_switchers = 0
winning_switchers = 0
for i in range(args.trials):
    # First, do a trial where the contestant never switches.
    won = simulate(args.doors, switch=False, steps=args.steps)
    if won:
        winning_non_switchers += 1

    # Next, try one where the contestant switches.
    won = simulate(args.doors, switch=True, steps=args.steps)
    if won:
        winning_switchers += 1

print('    Switching won {0:5} times out of {1} ({2}% of the time)'.format(
        winning_switchers, args.trials,
        (winning_switchers / args.trials * 100 ) ))
print('Not switching won {0:5} times out of {1} ({2}% of the time)'.format(
        winning_non_switchers, args.trials,
        (winning_non_switchers / args.trials * 100 ) ))
```

```
if __name__ == '__main__':

    main()
```

Code Discussion

The command-line arguments are parsed using the argparse module, and the resulting values are passed into the simulate() function.

When there are num_doors doors, simulate() numbers the doors from 0 up to num_doors-1. random.randint(a, b)() picks a random integer from the range a to b, possibly choosing one of the endpoints, so here we use random.randint(0, num_doors-1)().

To figure out which doors the host will open, the code makes a list of the currently closed doors, initially containing all the integers from 0 to num_doors-1. Then the code loops, picking a random door from the list to open. By our description of the problem, Monty will never open the contestant's door or the one hiding the prize, so the loop excludes those two doors and picks a different door. The loop continues until only two doors remain, so Monty will always open num_doors-2 doors.

To implement the contestant's switching strategy, we take the list of closed doors, which is now 2 elements long, and remove the contestant's current choice. The remaining element is therefore the door they're switching to.

To Do by the Students:

1. Understand the code, Run the python code and analyze the output
2. Increase the number of doors and check the results(you can pass in command line)
3. suuply doors=100 trials=1000 and compare the results with previous results
4. supply doors=10 and trails=4 and print the step by step results of output
5. How will you assert that always two doors are remaining
6. check what happens when number of doors are less than 2
7. Write assertion for conditions when winning door and choice are illegal

Learning outcome:
Random generation of possible inputs
usage of pseudo random number genarators
Conditional probability