Khoi Duong

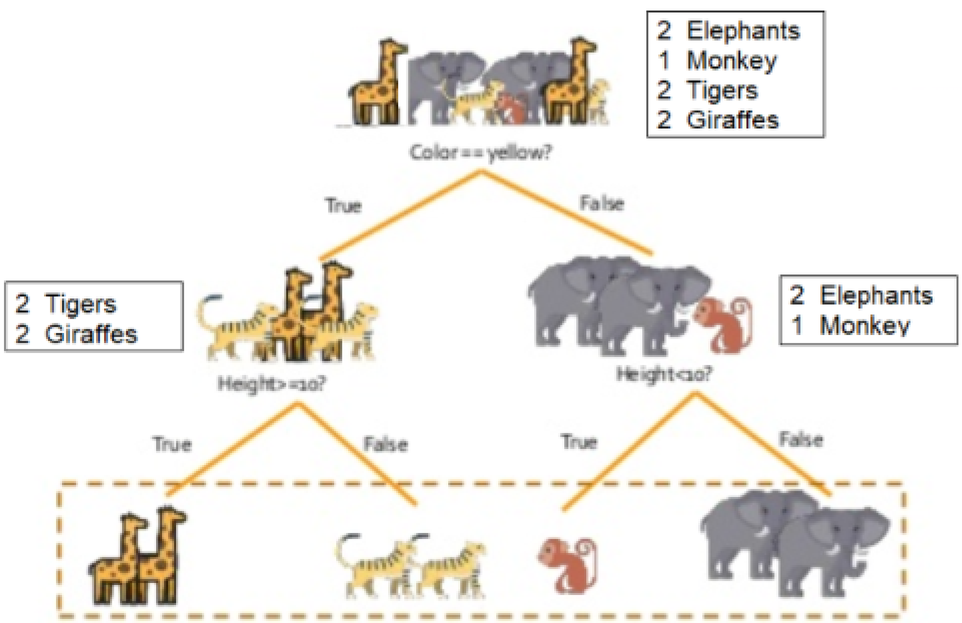
Prof. Yang

CS483

8/17/2022

FINAL

1.



Imp. of root = 3 * (2/7 * 5/7) + 1/7 * 6/7 = 36/49 = 0.735

Ave. imp. Of root = 7/7 * 0.735 = 0.735

<u>Color == yellow?</u>

LHS. imp. = 2 * (½*½) = ½

LHS. ave. imp. = 4/7 * ½ = 2/7

RHS. imp. = ⅔ * ⅓ + ⅓ * ⅔ = 4/9

RHS. ave. imp. = 3/7 * 4/9 = 4/21

Total ave. imp. = 2/7 + 4/21 = 10/21 = 0.476

Info gain = 0.735 - 0.476 = 0.259

Height >= 10?

Total ave. imp. = 0

Info gain = 2/7 (imp. Of color == yellow LHS) - 0 = 0.286

Height < 10?

Total ave. imp. = 0

Info gain = 4/21 (imp. Of color == yellow RHS) - 0 = 0.190

2.

Source code:

```
from numpy.random.mtrand import randint, rand
import numpy as np
# initial population of parent bitstring
pop = [[1,1,0,1,1,1,1,0,0,1],
       [1,1,0,0,0,0,1,0,1,1],
       [1,1,0,0,0,0,1,0,1,1],
       [1,1,1,0,0,1,0,0,0,0],
       [1,1,0,1,1,1,1,0,0,1],
       [1,1,1,0,0,1,0,0,0,0],
       [1,1,1,0,1,0,0,0,0,1],
       [0,0,1,1,0,1,0,0,0,0],
       [1,1,0,1,1,1,1,0,0,1],
       [0,1,0,0,1,1,0,0,1,0],
       [1,0,0,1,0,1,1,0,1,0],
       [1,1,0,1,1,1,1,0,0,1],
       [0,1,1,1,0,1,1,1,1,1],
       [1,0,0,1,1,1,0,1,0,1],
       [1,1,0,1,1,1,1,0,0,1],
```

```
       [1,1,1,0,1,0,0,0,0,1]]
print(pop)
# crossover two parents to create two children
def crossover(p1, p2, r_cross):
  # children are copies of parents by default
  c1, c2 = p1.copy(), p2.copy()
  # check for recombination
  if rand() < r_cross:
    # select crossover point that is not on the end of the string
    pt = randint(1, len(p1)-2)
    # perform crossover
    c1 = p1[:pt] + p2[pt:]
    c2 = p2[:pt] + p1[pt:]
  return [c1, c2]

...
# create the next generation
children = list()
for i in range(0, 16, 2):
  # get selected parents in pairs
  p1, p2 = pop[i], pop[i+1]
  # crossover and mutation
  for c in crossover(p1, p2, 0.8):
    # store for next generation
    children.append(c)

print(children)
```

Run the program and we have a result:

Parents

```
[[1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [1, 1, 0, 0, 0, 0, 1, 0, 1, 1],
 [1, 1, 0, 0, 0, 0, 1, 0, 1, 1],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 1],
 [0, 0, 1, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [0, 1, 0, 0, 1, 1, 0, 0, 1, 0],
 [1, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [0, 1, 1, 1, 0, 1, 1, 1, 1, 1],
 [1, 0, 0, 1, 1, 1, 0, 1, 0, 1],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 1]]
```

New generation

```
[[1, 1, 0, 1, 1, 1, 1, 0, 1, 1],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 1],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 0, 1, 0, 1, 1],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 0],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0],
 [0, 1, 1, 0, 1, 0, 0, 0, 0, 1],
 [1, 1, 0, 1, 1, 1, 0, 0, 1, 0],
 [0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
 [1, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 1, 0, 1, 1, 1, 1, 0, 0, 1],
 [0, 1, 0, 1, 1, 1, 0, 1, 0, 1],
 [1, 0, 1, 1, 0, 1, 1, 1, 1, 1],
 [1, 1, 0, 0, 1, 0, 0, 0, 0, 1],
 [1, 1, 1, 1, 1, 1, 1, 0, 0, 1]]
```

3.

Source code:

```
from numpy.random.mtrand import randint, rand
import numpy as np
# initial population of parent bitstring
pop = [[1,0,0,1,0,1,1,0,1,0],
       [1,1,1,0,0,1,0,0,0,0],
       [0,1,1,1,0,1,0,1,0,0],
       [1,1,1,0,0,1,0,1,0,1],
       [0,1,1,1,0,1,0,0,0,0],
       [1,1,1,0,0,1,0,1,0,1],
       [1,1,1,0,1,0,0,0,1,1],
```

```
        [1,1,1,0,1,0,0,0,0,1],
        [0,1,0,0,1,1,1,0,0,0],
        [0,1,1,0,0,1,0,0,1,1],
        [1,1,1,0,0,0,1,0,1,1],
        [1,0,0,0,0,1,0,0,1,0],
        [0,0,0,1,0,1,1,0,1,0],
        [1,0,1,1,0,1,0,0,0,0],
        [1,1,0,1,1,0,1,0,0,1],
        [1,1,1,0,0,1,0,0,0,0]]
print(pop)
# mutation operator
def mutation(bitstring, r_mut):
  for i in range(len(bitstring)):
    # check for a mutation
    if rand() < r_mut:
      # flip the bit
      bitstring[i] = 1 - bitstring[i]

for i in pop:
  mutation(i, 0.025)
print(pop)
```

Run the program & result:

New Gen.

```
[[1, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
 [0, 1, 1, 1, 0, 1, 0, 1, 0, 0],
 [1, 1, 1, 0, 0, 1, 0, 1, 0, 1],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 1, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 1],
 [0, 1, 0, 0, 1, 1, 1, 0, 0, 0],
 [0, 1, 1, 0, 0, 1, 0, 0, 1, 1],
 [1, 1, 1, 0, 0, 0, 1, 0, 1, 1],
 [1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
 [0, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 1, 0, 0, 1],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0]]
```

New Gen. after Mutation:

```
[[1, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 1],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 1, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 1],
 [0, 1, 0, 0, 1, 1, 1, 0, 0, 0],
 [0, 1, 1, 0, 0, 1, 0, 0, 1, 1],
 [1, 1, 1, 0, 0, 0, 1, 0, 1, 1],
 [1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
 [1, 0, 0, 1, 0, 1, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 1, 0, 0, 1],
 [1, 1, 1, 0, 0, 1, 0, 0, 0, 0]
```

4.

Source code:

```
import numpy as np

initPi = [0.1, 0.6, 0.3]
tranA = np.matrix([[0.1, 0.7, 0.2], [0.75, 0.15, 0.1], [0.6, 0.35, 0.05]])
initPi = initPi*(tranA**3)
initPi
```

Run program & result:

```
matrix([[0.518275 , 0.3577125, 0.1240125]])
```

Therefore, we have the probability below:

P (London) = 0.518

P (Barcelona) = 0.358

P (New York) = 0.124