Khoi Duong

Prof. Yang

CS483

8/6/2022

HW#5

1.

We have the dataset below:

ID	Red	Green	Blue	Size (cm)	Fruit (Label)
0	1	0	0	7	Apple
1	0	1	0	20	Water Melon
2	1	0	0	1	Cherry
3	0	1	0	7.5	Apple
4	1	0	0	1	Strawberry
5	1	0	0	0.8	Cherry

We have the condition list below:

ID	CONDITION LIST
0	Red == 0?
1	Red == 1?
2	Green == 0?
3	Green == 1?
4	Blue == 0?
5	Blue == 1?
6	Small size ? (0 ~ 1)

7	Medium size? (7 ~ 8)
8	Big size? (~20)

Randomly create bootstrapping subsets from training set

Btstrp1	Btstrp2	Btstrp3
4	3	2
1	3	1
2	4	4
3	1	3
0	4	2
2	1	3

Randomly take sqrt(9) = 3 features from condition list 0 ~ 8 for each Btstrp

We have the table below:

Condition ID for Btstrp1	Condition ID for Btstrp2	Condition ID for Btstrp3
7	4	2
4	6	0
3	7	8

We have Decision Tree 1 for Btstrp1 and 1st condition list:

Decision Tree 1					
ID	Red	Green	Blue	Size	Fruit (label)
4	1	0	0	Small	Strawberry
1	0	1	0	Big	Watermelon
2	1	0	0	Small	Cherry
3	0	1	0	Medium	Apple
0	1	0	0	Medium	Apple
2	1	0	0	Small	Cherry

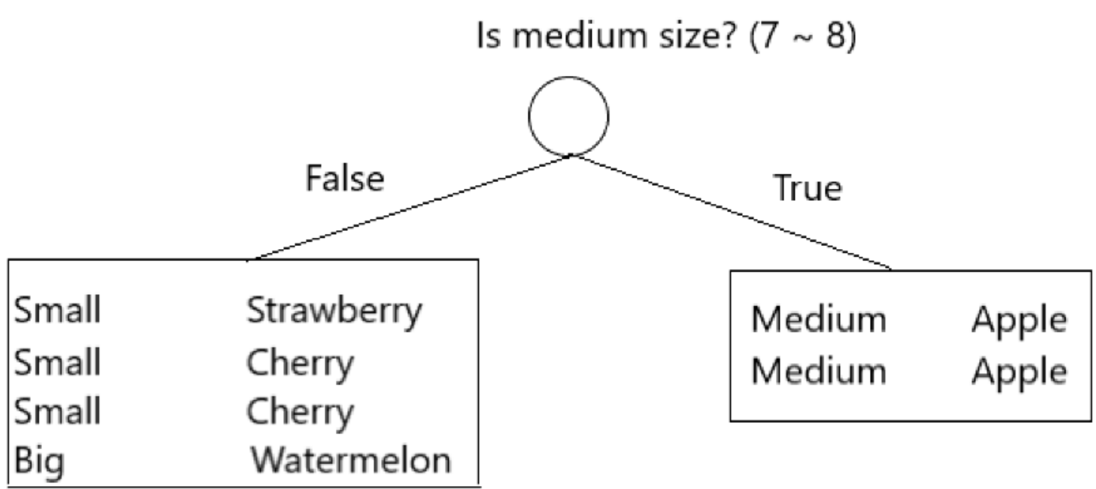
ID	Condition list
7	Medium size? (7 ~ 8)
4	Blue == 0?
3	Green == 1?

Impurity of root

$$\text{imp.} = P(S)*(1-P(S)) + P(W)*(1-P(W)) + P(C)*(1-P(C)) + P(A)*(1-P(A))$$

$$= \tfrac{1}{6}*\tfrac{5}{6} + \tfrac{1}{6}*\tfrac{5}{6} + \tfrac{1}{3}*\tfrac{2}{3} + \tfrac{1}{3}*\tfrac{2}{3} = 13/18 = 0.72$$

Ave. imp = 6/6 * 0.72 = 0.72



LHS imp. $= \tfrac{1}{4} * \tfrac{3}{4} + \tfrac{1}{2} * \tfrac{1}{2} + \tfrac{1}{4} * \tfrac{3}{4} = 0.625$

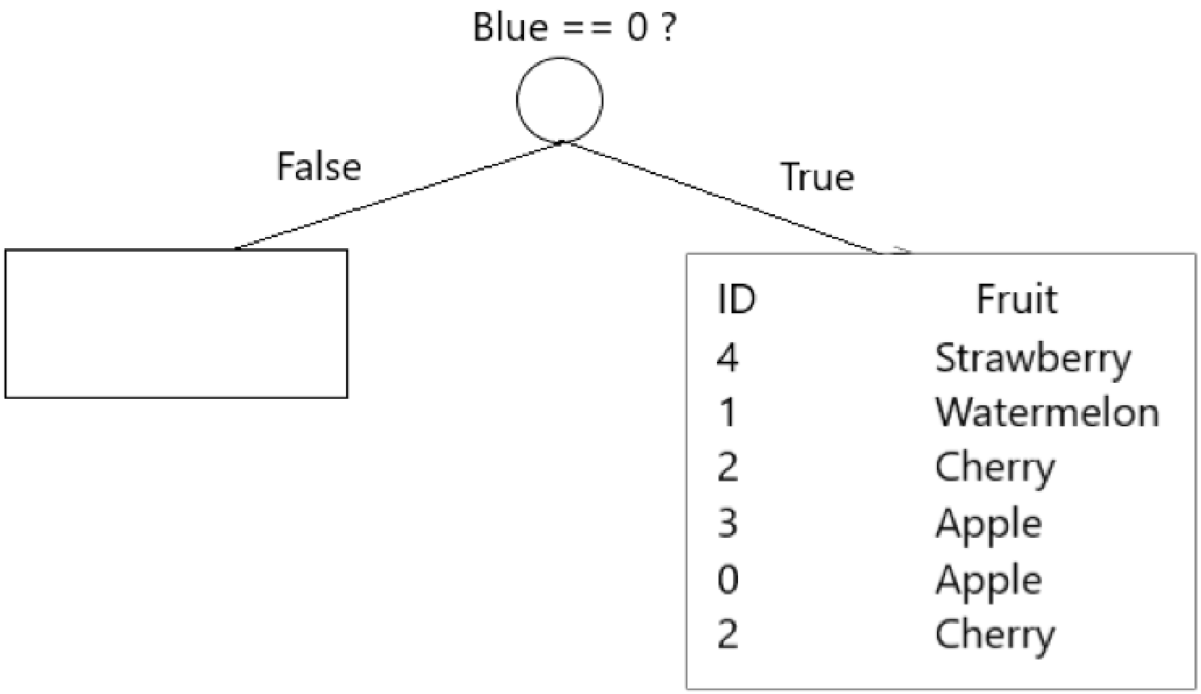
RHS imp. = 0

LHS Ave. imp. = 4/6 * 0.625 = 5/12 = 0.42

RHS Ave. imp. = 2/6 * 0 = 0

Total Ave. imp = 0.42 + 0 = 0.42

Info Gain = 0.72 (imp. Of root) - 0.42 = 0.3



LHS imp. = 0

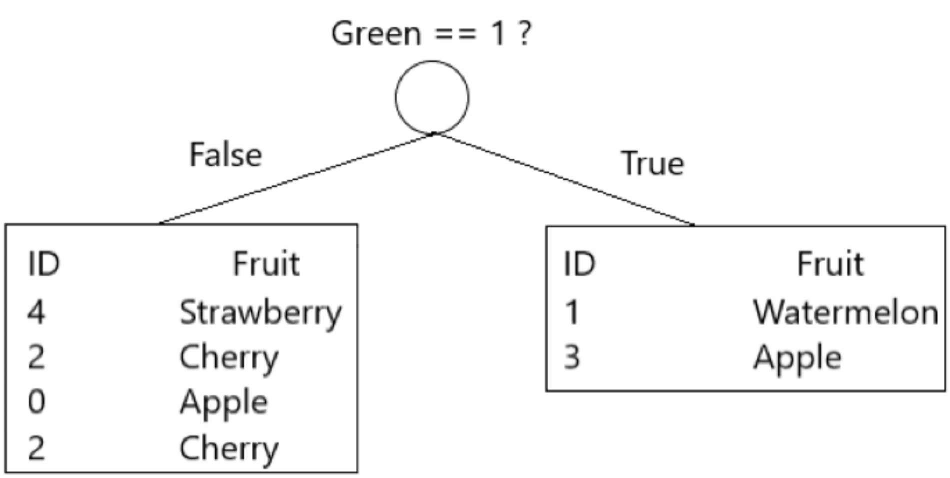
RHS imp. = $\frac{1}{6} * \frac{5}{6} + \frac{1}{6} * \frac{5}{6} + \frac{1}{3} * \frac{2}{3} + \frac{1}{3} * \frac{2}{3} = 0.72$

LHS Ave. imp. = 0

RHS Ave. Imp. = 6/6 * 0.72 = 0.72

Total Ave. imp. = 0.72

Info gain = 0.72 (imp. Of root) - 0.72 = 0



LHS imp. = $2 * (\frac{1}{4} * \frac{3}{4}) + \frac{1}{2} * \frac{1}{2}$ $= \frac{5}{8} = 0.625$

RHS imp. = $2 * (\frac{1}{2} * \frac{1}{2}) = \frac{1}{2} = 0.5$

LHS Ave. imp. = $4/6 * 0.625 = 0.42$

RHS Ave. imp. = $2/6 * 0.5 = \frac{1}{6} = 0.17$

Total Ave. imp. = $0.42 + 0.17 = 7/12 = 0.58$

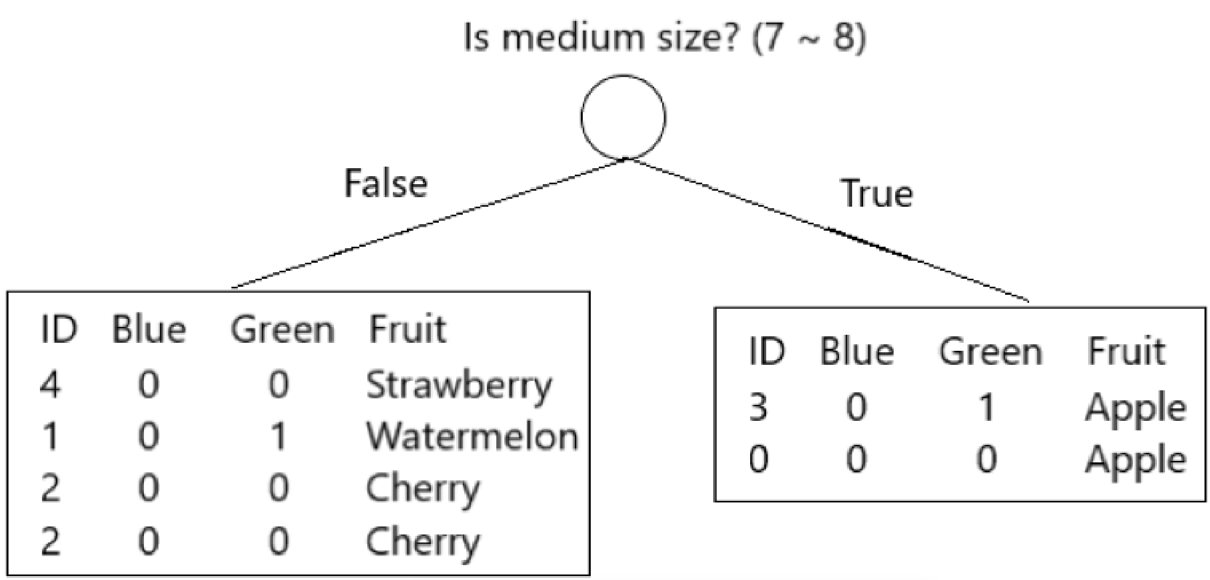
Info gain = 0.72 (imp. Of root) - 0.58 = 0.14

Comparison of 3 info gain

Is medium size? (7 ~ 8)	Blue == 0 ?	Green == 1?
0.3	0	0.14

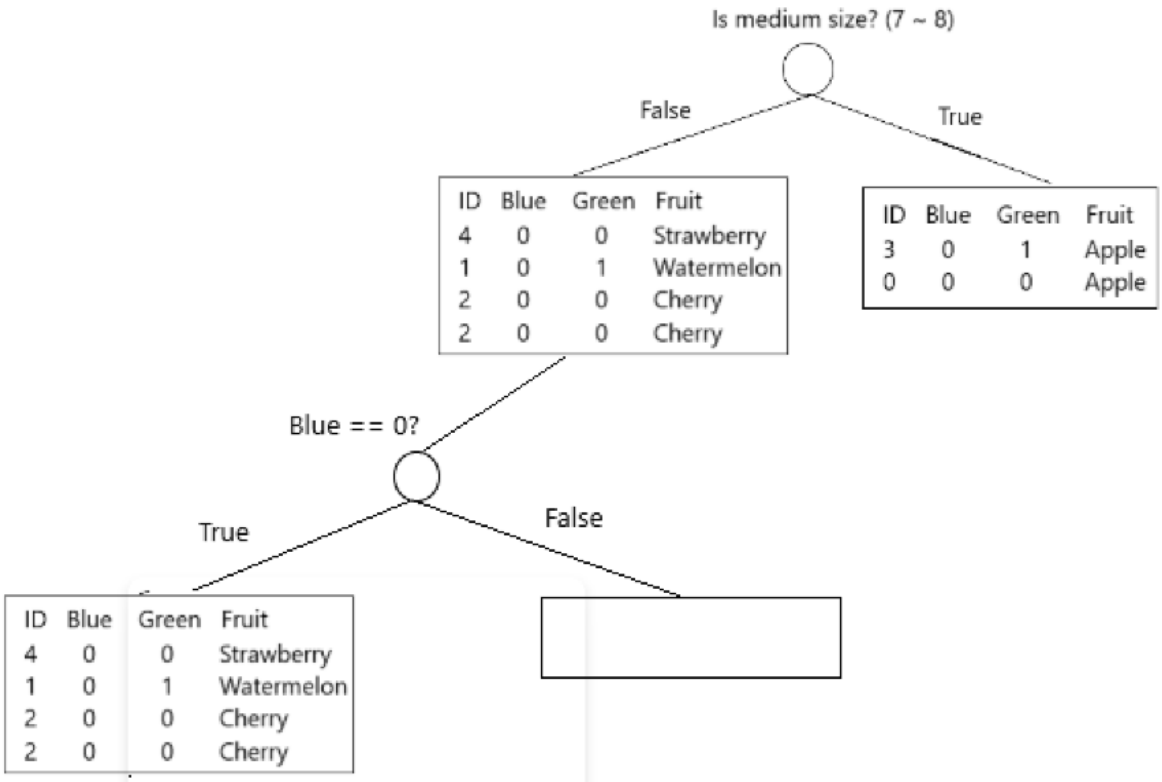
Therefore, taking “Is medium size? (7 ~ 8)” will get the highest info gain

Thus, we have a new schema and condition list as below:



ID	Condition list
7	Medium size? (7 ~ 8)
4	Blue == 0?
3	Green == 1?

Go to next checking condition

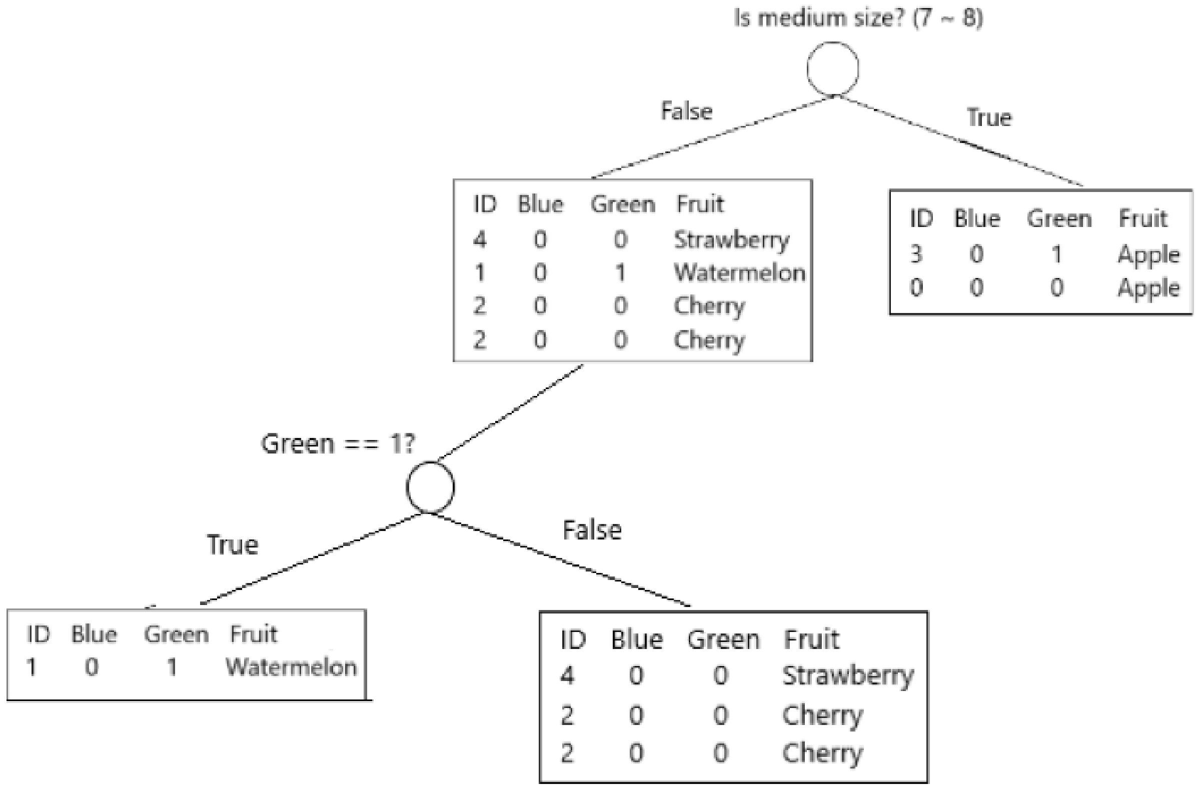


LHS imp. = 2 * (¼ * ¾) + ½ * ½ = 0.625 RHS imp. = 0

Ave. LHS imp. = 4/6 * 0.625 = 0.42 Ave. RHS imp. = 0

Total Ave. imp = 0.42

Info Gain = 0.42 (imp. of “is Medium size?”) - 0.42 = 0



LHS imp. = 0

RHS imp. = ⅓ * ⅔ + ⅔ * ⅓ = 4/9 = 0.44

Ave. LHS imp. = 0

Ave. LHS imp. = 3/6 * 0.44 = 0.22

Total ave. imp. = 0.22

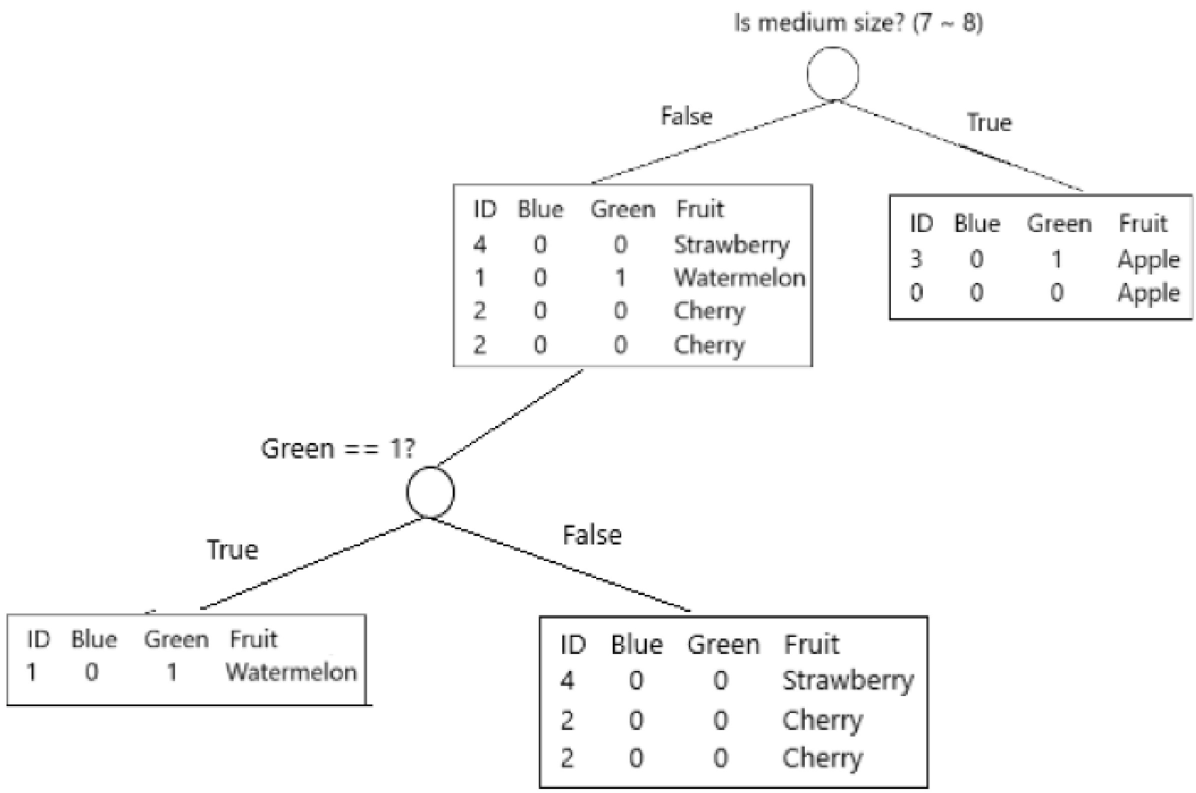
Info gain = 0.42 (imp. of "is Medium size?") - 0.22 = 0.2

Comparison of 2 info gain

Blue == 0?	Green == 1?
0	0.2

Therefore, taking "Green == 1?" will get the highest info gain

<u>We have the decision tree 1 as follow:</u>



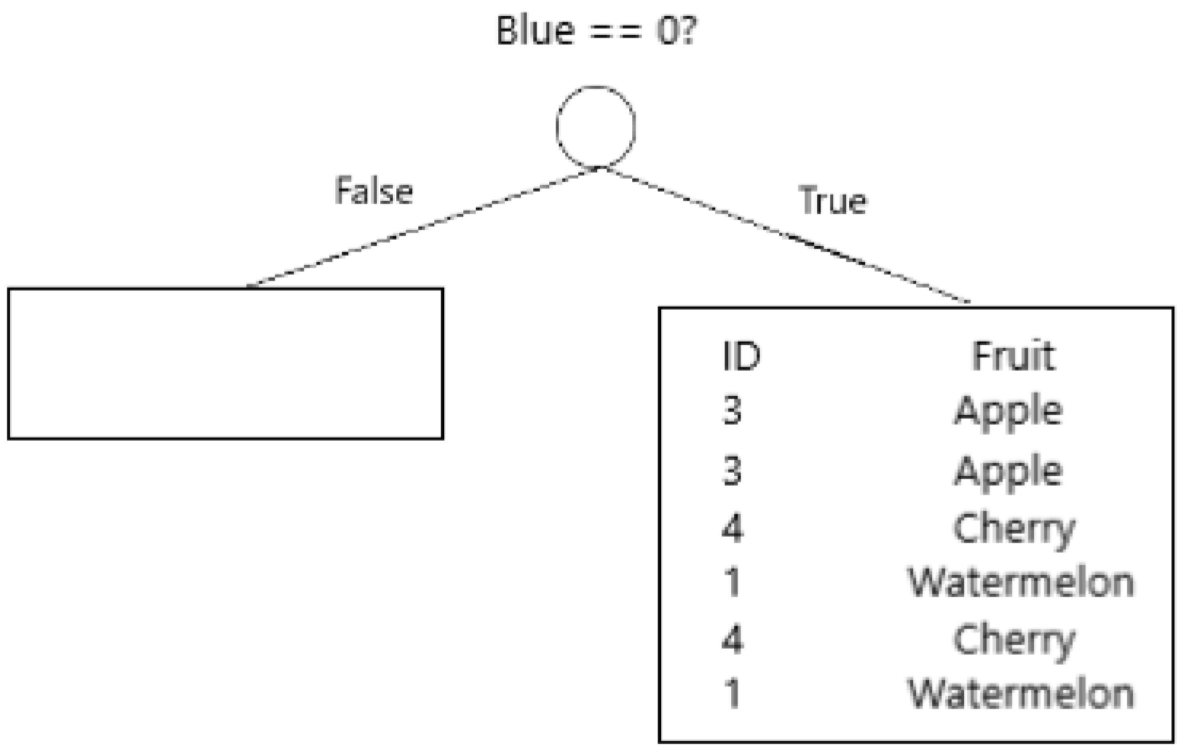
We have Decision Tree 2 for Btstrp2 and 2nd condition list:

Decision Tree 2					
ID	Red	Green	Blue	Size	Fruit (label)
3	0	1	0	Medium	Apple
3	0	1	0	Medium	Apple
4	1	0	0	Small	Strawberry
1	0	1	0	Big	Watermelon
4	1	0	0	Small	Strawberry
1	0	1	0	Big	Watermelon

ID	Condition list
4	Blue == 0?
6	Small size? (0 ~ 1)
7	Medium size? (7 ~ 8)

Imp. of root = 3 * (⅓ * ⅔) = ⅔ = 0.67

Ave. imp. = 6/6 * 0.67 = 0.67



LHS imp. = 0

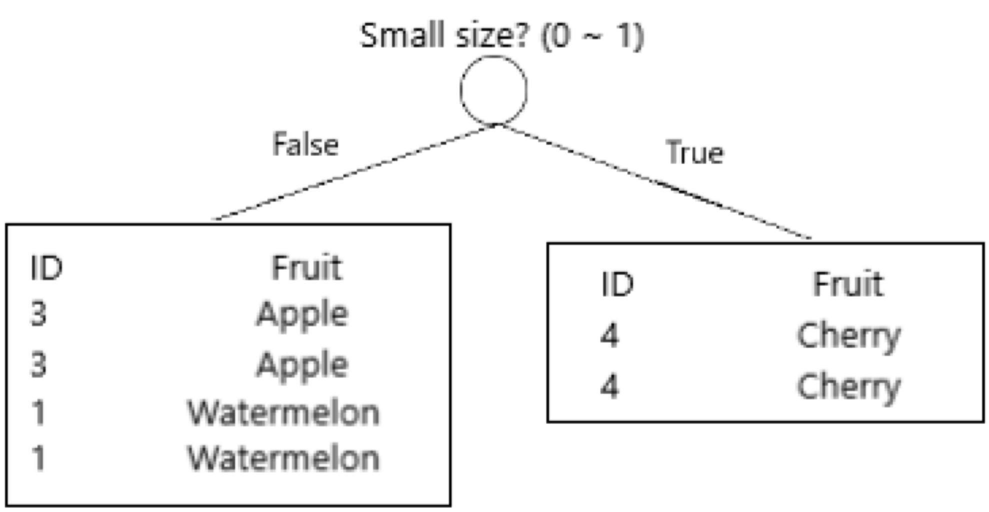
RHS imp. = 0.67

Ave. LHS imp. = 0

Ave. RHS imp. = 6/6 * 0.67 = 0.67

Total Ave. imp. = 0.67

Info gain = 0.67 - 0.67 = 0



LHS. imp. = 0.5

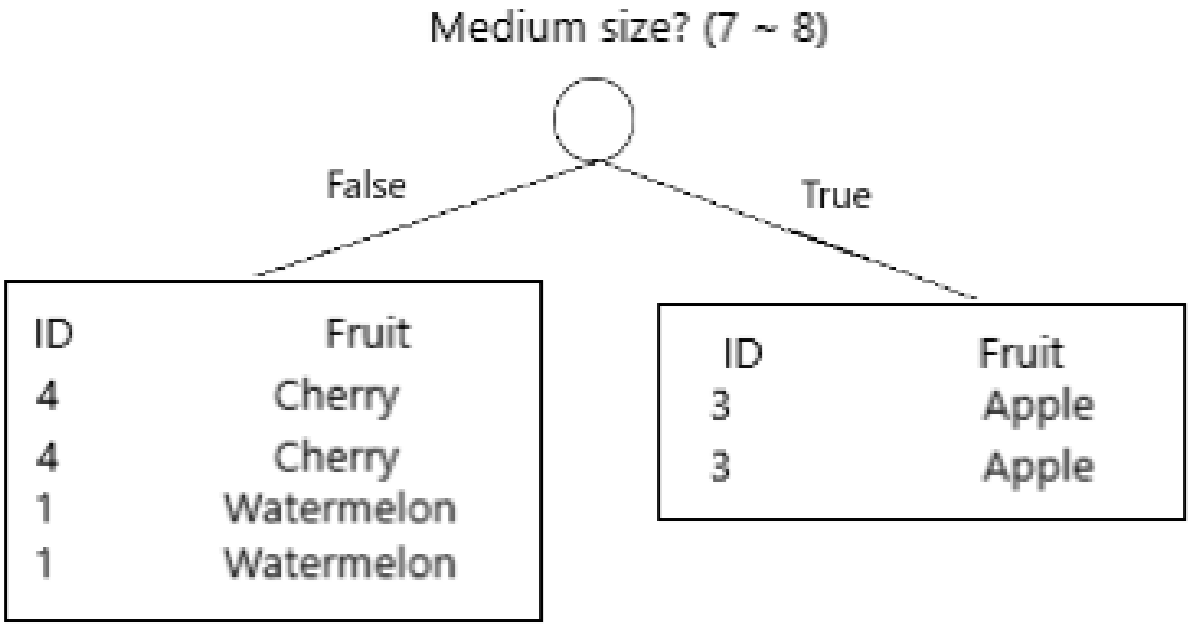
RHS imp. = 0

Ave imp. = 4/6 * 0.5 = 0.33

Ave. imp. = 0

Total ave. imp. = 0.33

Info gain = 0.67 - 0.33 = 0.34



LHS imp. = 0.5

RHS imp. = 0

Ave. LHS imp. = 4/6 * 0.5 = 0.33

Ave. RHS imp. = 0

Total ave. imp. = 0.33

Info gain = 0.67 - 0.33 = 0.34

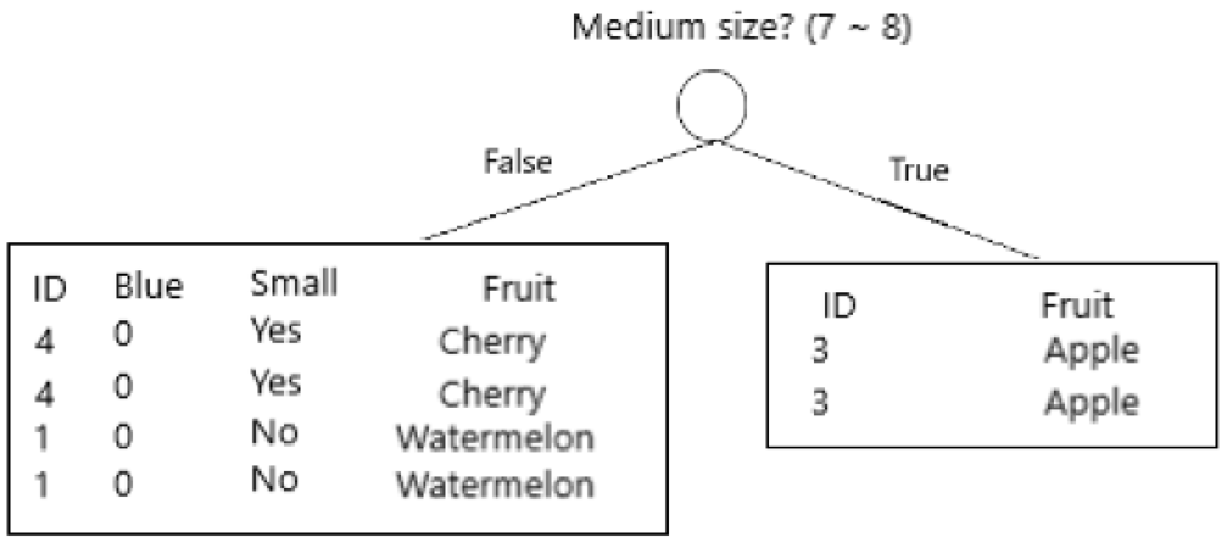
Comparison of 3 info gain

Blue == 0?	Small size?	Medium size?
0	0.34	0.34

The info gain from “Small size?” and “Medium size?” is equal ⇔ taking either “Small size?” or “Medium size?” will get the highest info gain.

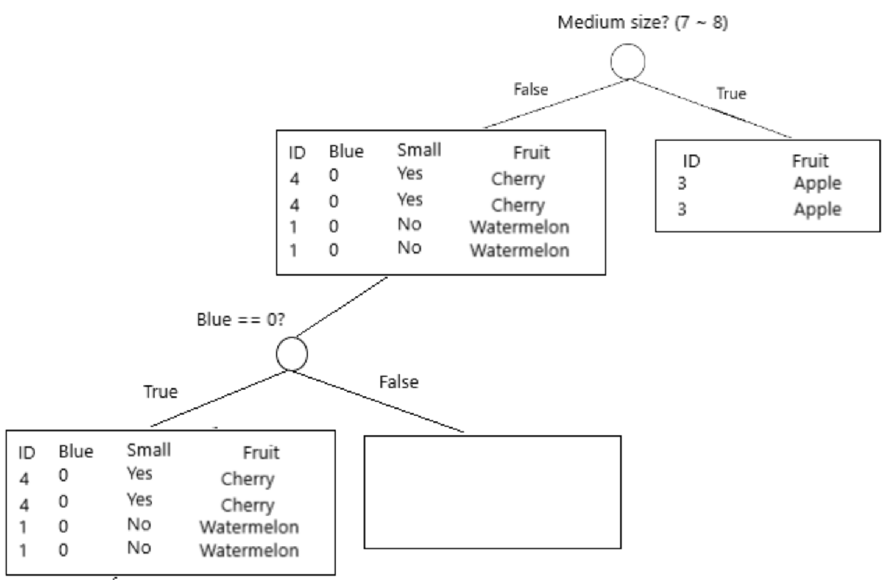
We will select “Medium size?”

Thus, we have a new schema and condition list as below:



ID	Condition list
4	Blue == 0?
6	Small size? (0 ~ 1)
7	Medium size? (7 ~ 8)

Go to the next checking condition

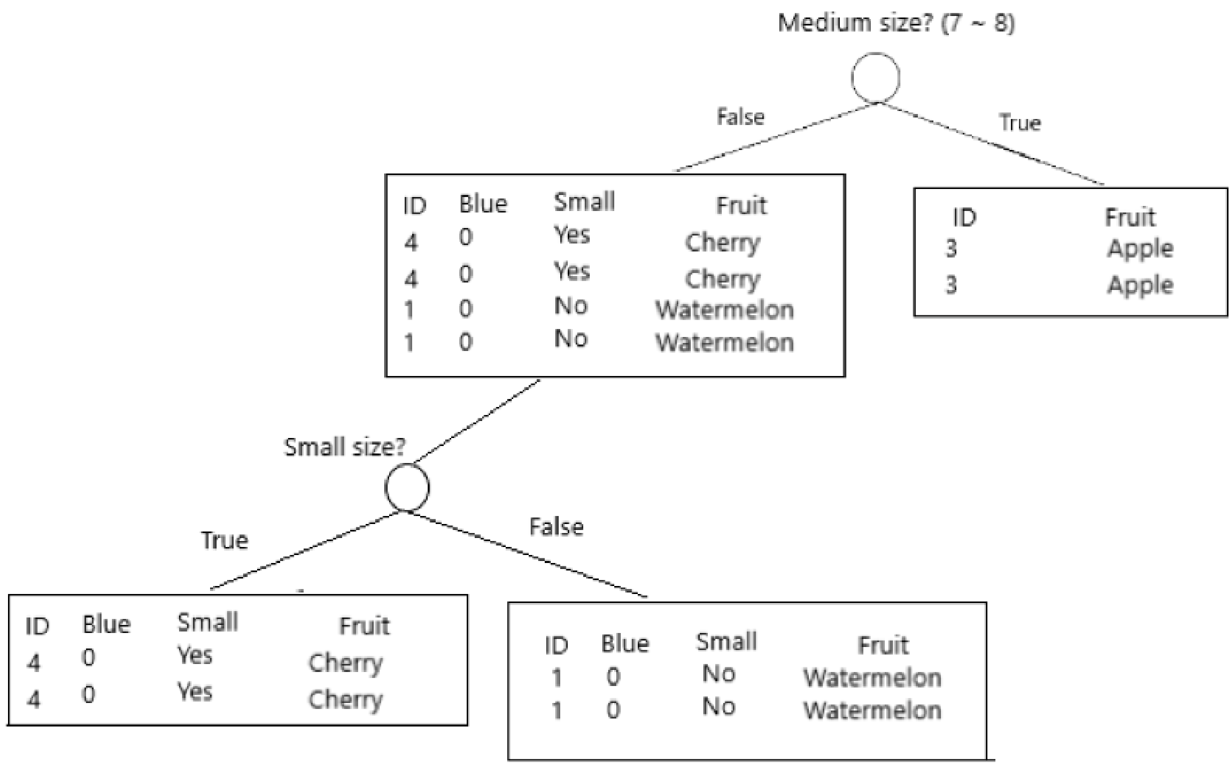


LHS. imp. = 0.5 RHS imp. = 0

Ave. LHS imp. = 0.33 Ave. RHS imp. = 0

Total ave. imp. = 0.33

Info gain = 0.33 - 0.33 = 0



LHS imp. = 0 RHS. imp. = 0

Total ave. imp. = 0

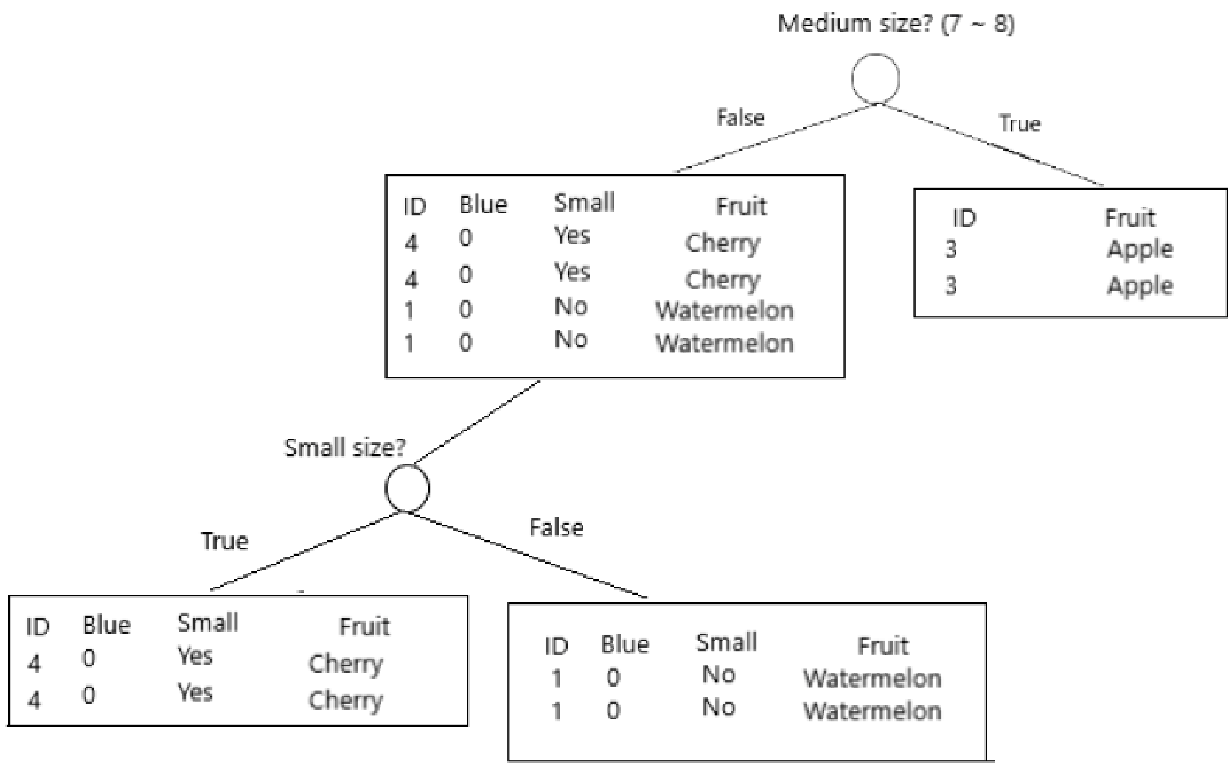
Info gain = 0.33 - 0 = 0.33

Comparison of 2 info gain

Blue == 0?	Small size?
0	0.33

Taking “Small size?” will get the highest info gain.

<u>Thus, we have the decision tree 2 as follow:</u>



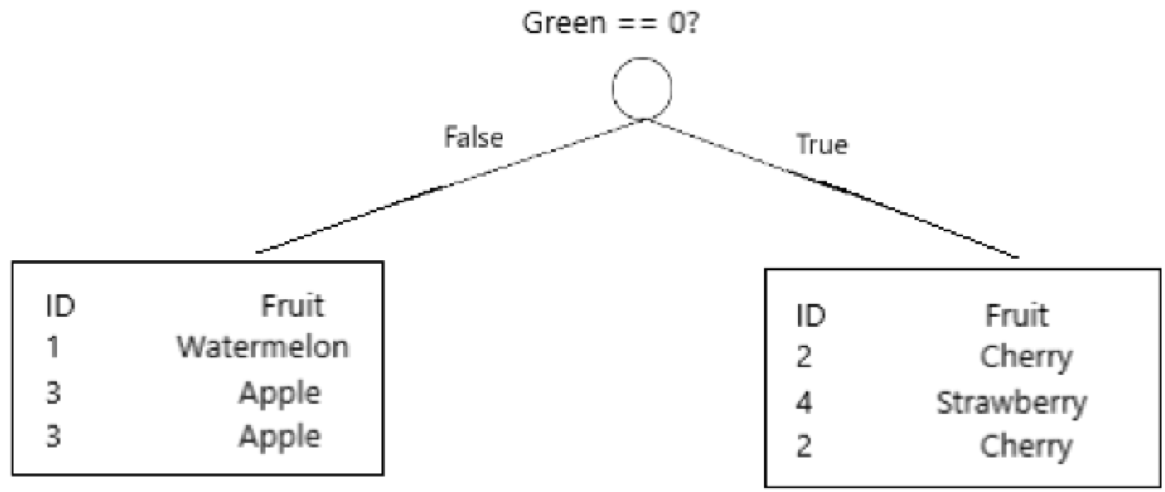
We have Decision Tree 3 for Btstrp3 and the 3rd condition list:

Decision Tree 3					
ID	Red	Green	Blue	Size	Fruit (label)
2	1	0	0	Small	Cherry
1	0	1	0	Big	Watermelon
4	1	0	0	Small	Strawberry
3	0	1	0	Medium	Apple
2	1	0	0	Small	Cherry
3	0	1	0	Medium	Apple

ID	Condition list
2	Green == 0?
0	Red == 0?
8	Big size? (~20)

Imp. of root = $2 * (\frac{1}{3} * \frac{2}{3}) + 2 * (\frac{1}{6} * \frac{5}{6}) = 13/18 = 0.72$

Ave. imp. Of root = 6/6 * 0.72 = 0.72



LHS imp. = 2 * (⅓ * ⅔) = 4/9 = 0.44

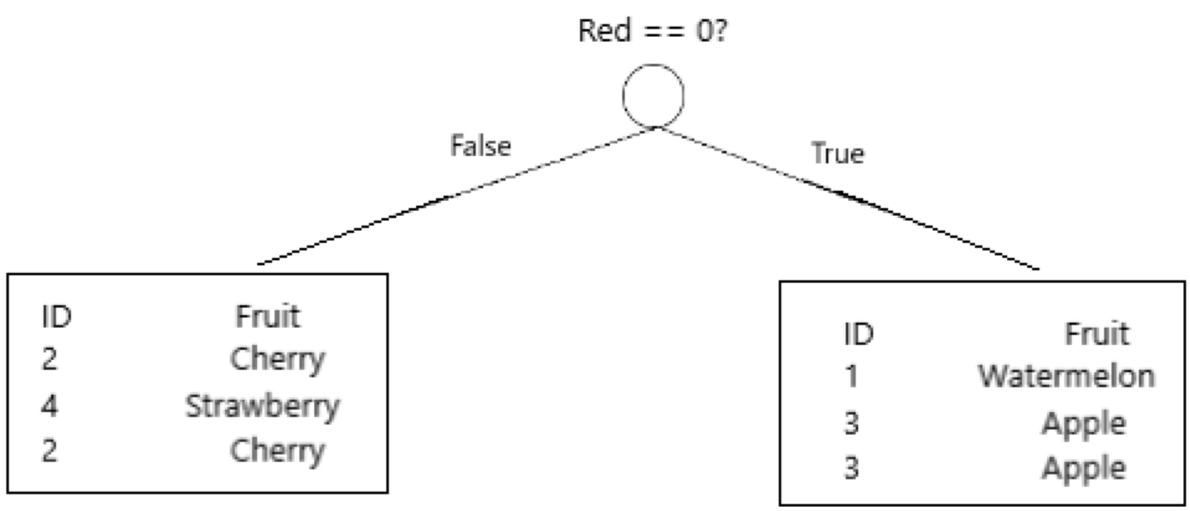
Ave. LHS imp. = ½ * 0.44 = 0.22

RHS imp. = 2 * (⅓ * ⅔) = 4/9 = 0.44

Ave. RHS imp. = ½ * 0.44 = 0.22

Total ave. imp. = 0.44

Info gain = 0.72 - 0.44 = 0.28



LHS imp. = 2 * (⅓ * ⅔) = 4/9 = 0.44

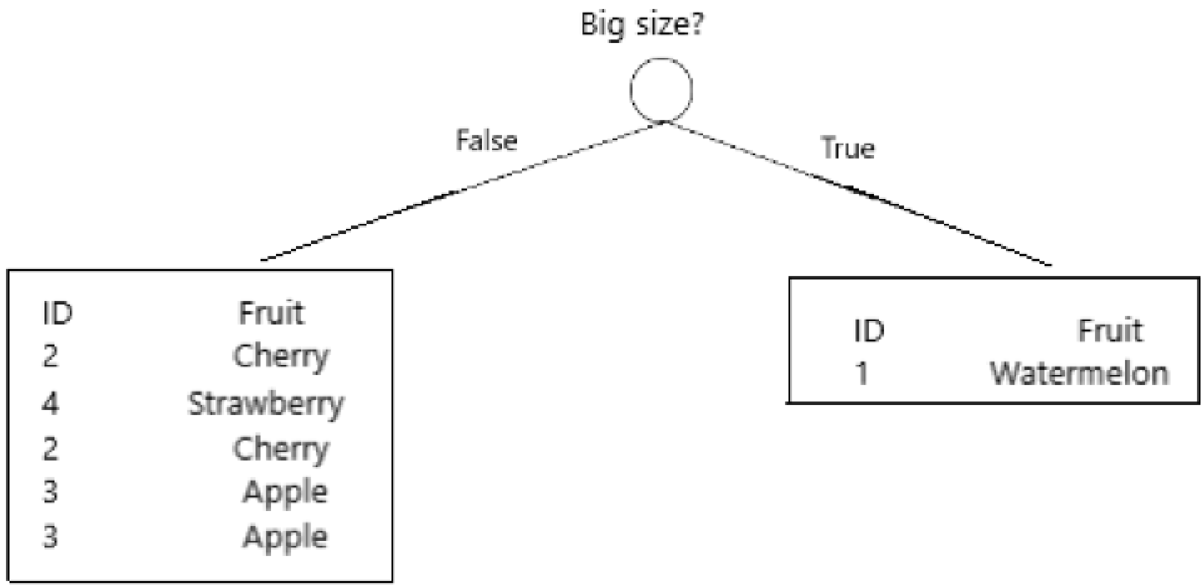
Ave. LHS imp. = ½ * 0.44 = 0.22

RHS imp. = 2 * (⅓ * ⅔) = 4/9 = 0.44

Ave. RHS imp. = ½ * 0.44 = 0.22

Total ave. imp. = 0.44

Info gain = 0.72 - 0.44 = 0.28



LHS imp. = $2 * (\frac{2}{5} * \frac{3}{5}) + \frac{1}{5} * \frac{4}{5} = 0.64$

RHS imp. = 0

Ave. LHS imp. = $\frac{5}{6} * 0.64 = 0.53$

Ave. RHS imp. = 0

Total ave. imp. = 0.53

Info gain = 0.72 - 0.53 = 0.19

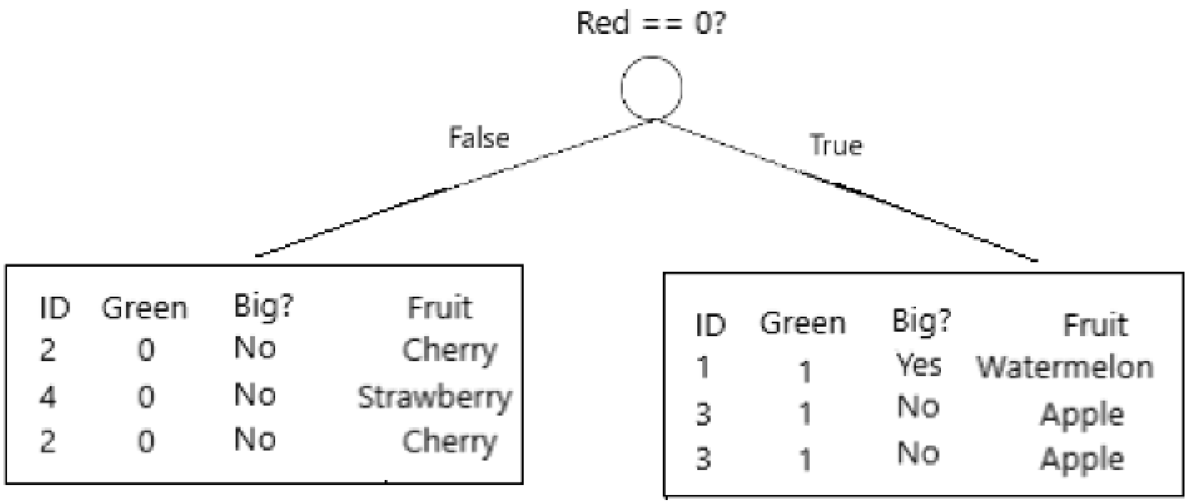
Comparison of 3 info gain:

Green == 0?	Red == 0?	Big size?
0.28	0.28	0.19

Thus, taking “Green == 0?” or “Red == 0?” will get highest info gain

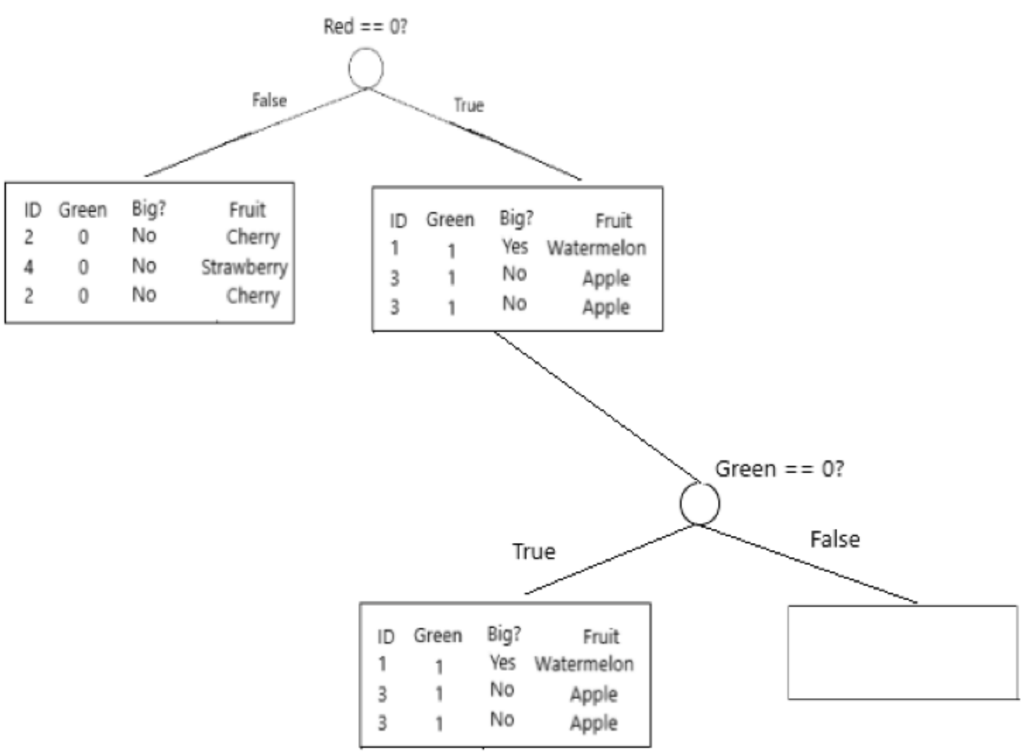
We select “Red == 0?”

Thus, we have a new schema and condition list as below:



ID	Condition list
2	Green == 0?
0	Red == 0?
8	Big size? (~20)

Go to the next checking condition

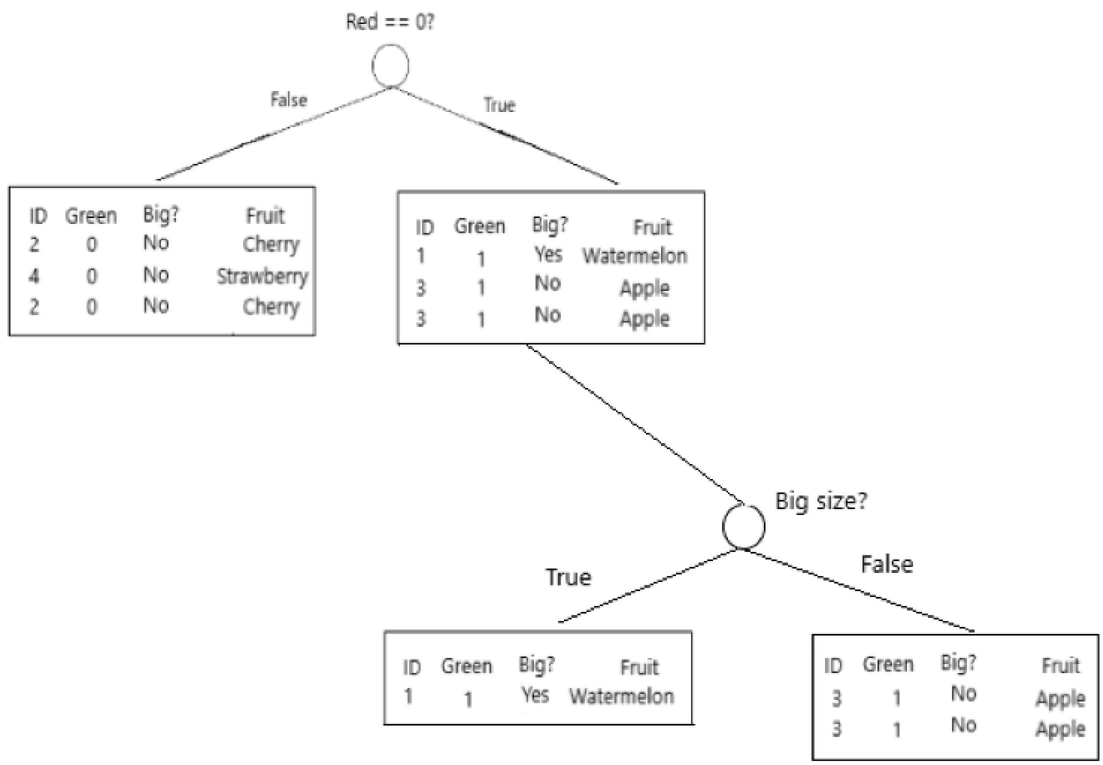


LHS imp. = 2 * (⅓ * ⅔) = 0.44 RHS imp. = 0

Ave. LHS imp. = ½ * 0.44 = 0.22 Ave. RHS imp. = 0

Total ave. imp. = 0.22

Info gain = 0.44 - 0.22 = 0.22



LHS imp. = 0 RHS imp. = 0

Total ave. imp. = 0

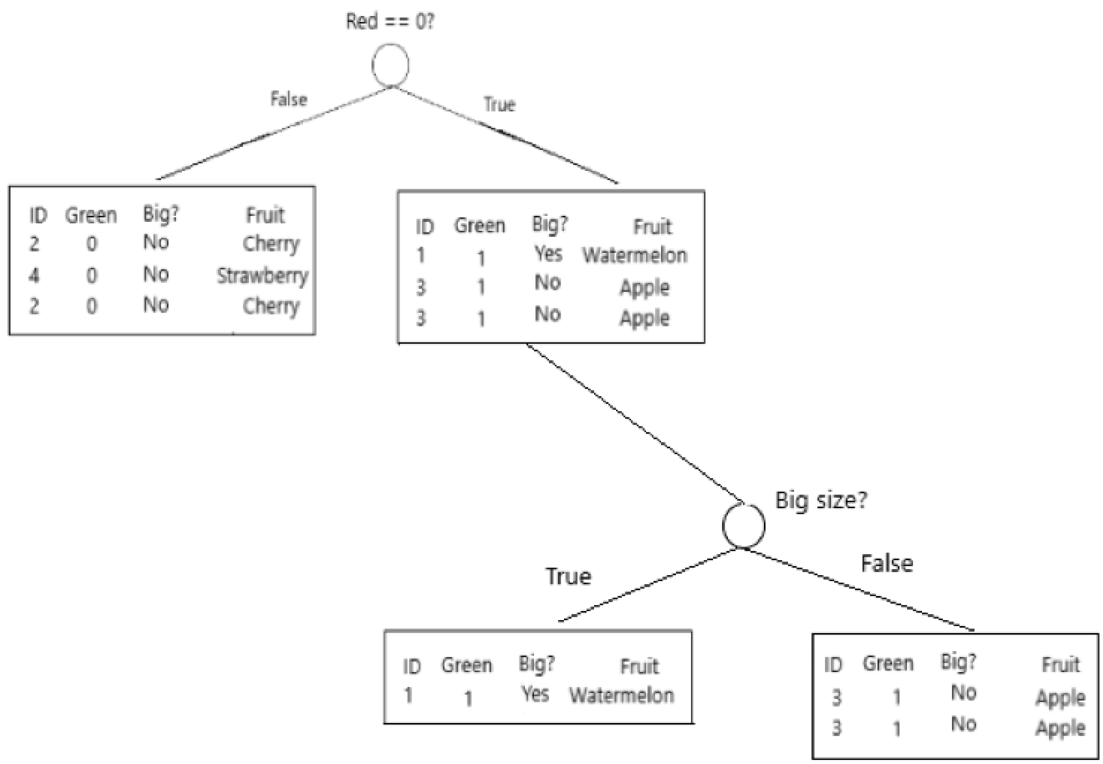
Info gain = 0.44 - 0 = 0.44

Comparison 2 info gain

Green == 0?	Big size?
0.22	0.44

Thus, taking “Big size?” will get the highest info gain.

<u>We have the decision tree 3 as follow:</u>



Write Python function to compare:

Source code:

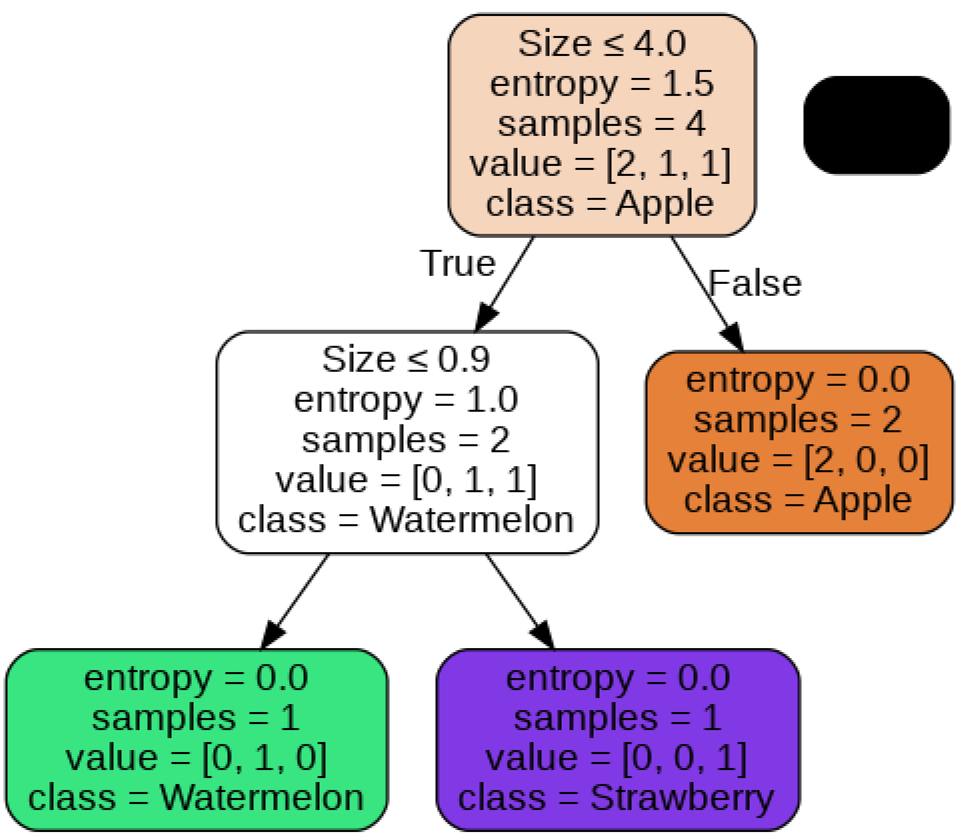
```
from google.colab import drive
drive.mount('/content/drive')
data_path = "/content/drive/My Drive/Colab Notebooks/hw5_ex1.csv"

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
col_names = ['ID', 'Red', 'Green', 'Blue', 'Size', 'label']
ex1 = pd.read_csv(data_path, header = None, names = col_names)
feature_cols = ['Red', 'Green', 'Blue', 'Size']
ex1.head()
a = ex1[feature_cols]
b = ex1.label
a_train, a_test, b_train, b_test = train_test_split(a,b, test_size=0.3,
random_state=1)
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf = clf.fit(a_train,b_train)
```

```
!pip install graphviz
!pip install pydotplus
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf,    out_file=dot_data,    filled=True,    rounded=True,
special_characters=True,                    feature_names              =
feature_cols,class_names=['Apple','Watermelon','Strawberry','Cherry'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('hw5_ex1.png')
Image(graph.create_png())
```

Run program & result:



2.

Num. of Bit = CEILING(LOG((2-(-2))*10^2,2),1) = 9

9 genes need 16 chromosomes.

Randomly take 16 chromosomes from 0 to 2^9-1 as follows

ID	Random #	Conv to bin	decoded value
1	395	110001011	3.475073314
2	92	001011100	0.809384164
3	276	100010100	2.428152493
4	402	110010010	3.536656891
5	138	010001010	1.214076246
6	242	011110010	2.129032258
7	427	110101011	3.75659824
8	231	011100111	2.032258065
9	289	100100001	2.542521994
10	29	000011101	0.255131965
11	51	000110011	0.448680352
12	192	011000000	1.68914956
13	277	100010101	2.436950147
14	318	100111110	2.797653959
15	501	111110101	4.407624633
16	434	110110010	3.818181818

To get max value, fitness function as follows

$$\text{fitness}(x) = \frac{1}{e^{-x^2} + 0.01\cos(200x)}$$

And then max(fitness(x)) = max(f(x))

Select cmin = 0

We have the roulette wheel method:

(https://docs.google.com/spreadsheets/d/1G3Ft4vF5ZwNqKVuje1NeMCVqtcz0rQV-/edit?usp=sharing&ouid=107073872705456706477&rtpof=true&sd=true)

ID	Decimal #	Conv to bin	decoded value	Fitness Value f(x)	F(x),Cmin*	Probability	Cum. Prob.	Prob. Slots	Rand # in [0,1]	Selected Chro
1	395	110001011	3.475073314	-0.007494623	0	0	0	0-0	0.754224499	000110011
2	92	001011100	0.809384164	0.520235801	0.5202358	0.36136047	0.36136047	0-0.3614	0.571949657	000110011
3	276	100010100	2.428152493	0.00023366	0.00023366	0.0001623	0.36152278	0.3614-0.3615	0.178291197	001011100
4	402	110010010	3.536656891	-0.008898217	0	0	0.36152278	0.3615-0.3615	0.467828885	000110011
5	138	010001010	1.214076246	0.22289515	0	0	0.36152278	0.3615-0.3615	0.372428492	011100111
6	242	011110010	2.129032258	0.011954311	0.01195431	0.00830357	0.36982635	0.3615-0.3698	0.759325858	000110011
7	427	110101011	3.75659824	-0.008873801	0	0	0.36982635	0.3698-0.3698	0.400465226	000110011
8	231	011100111	2.032258065	0.012329492	0.01232949	0.00856418	0.37839052	0.3698-0.3784	0.740194732	000110011
9	289	100100001	2.542521994	0.010632442	0.01063244	0.00738539	0.38577591	0.3784-0.3858	0.268174898	001011100
10	29	000011101	0.255131965	0.944223042	0	0	0.38577591	0.3858-0.3858	0.325572876	001011100
11	51	000110011	0.448680352	0.815662245	0.81566224	0.56656634	0.95234226	0.3858-0.9523	0.744291876	000110011
12	192	011000000	1.68914956	0.05874288	0.05874288	0.04080333	0.99314559	0.9523-0.9931	0.051114701	001011100
13	277	100010101	2.436950147	-0.006398592	0	0	0.99314559	0.9931-0.9931	0.477142609	000110011
14	318	100111110	2.797653959	0.009868013	0.00986801	0.00685441	1	0.9931-1	0.832041331	000110011
15	501	111110101	4.407624633	-0.003033315	0	0	1	1-1	0.830438648	000110011
16	434	110110010	3.818181818	-0.00973768	0	0	1	1-1	0.094481495	001011100
				Sum	1.43965884	1				

Assume the crossover rate Pc = 0.8, the mutation rate Pm = 0.025

There will be 9 (chromosome size) * 16 (population size) * 0.025 (Pm) = 4 bits be mutated.

The number of evolution generations will be

Source code:

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

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

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]
...
# 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):
    mutation(c, 0.025)
    # store for next generation
    children.append(c)

print(children)
```

After crossover and mutation, we have the first generation from parents (generation 0)

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

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

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

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

Performance analysis:

(https://docs.google.com/spreadsheets/d/1G3Ft4vF5ZwNqKVuje1NeMCVqtcz0rQV-/edit?usp=sharing&ouid=107073872705456706477&rtpof=true&sd=true)

Updated ID	Generation P(2)	Mutated P(2)	decoded value	Fitness Value f(x)
1	000110011	51	0.448680352	0.815662245
2	000110011	51	0.448680352	0.815662245
3	001110011	115	1.011730205	0.36212759
4	000011100	28	**0.246334311**	**0.946538421**
5	000110011	51	0.448680352	0.815662245
6	001100111	103	0.906158358	0.445501732
7	000110001	49	0.431085044	0.828652473
8	000110011	51	0.448680352	0.815662245
9	001011110	94	0.826979472	0.500187633
10	001011100	92	0.809384164	0.520235801
11	000111100	60	0.527859238	0.760041983
12	001010011	83	0.730205279	0.587154486
13	000110011	51	0.448680352	0.815662245
14	000110011	51	0.448680352	0.815662245
15	000111100	60	0.527859238	0.760041983
16	001010011	83	0.730205279	0.587154486
			Max	**0.946538421**

The table above shows the performance of the first generation.

Keep doing until we get to the 18th generation, using the Python program.

We have the source code below:

```
# genetic algorithm search of the one max optimization problem
from numpy.random import randint
from numpy.random import rand
import math

# objective function
def onemax(x):
  num = 0
  base = 8     # n_bits - 1 = 9 - 1 = 8
  for i in x:
    num += (2**base)*i
```

```
    base -= 1
  ans = math.exp(-(num**2)) + 0.01*math.cos(200*num)
  return ans


# tournament selection
def selection(pop, scores, k=3):
  # first random selection
  selection_ix = randint(len(pop))
  for ix in randint(0, len(pop), k-1):
    # check if better (e.g. perform a tournament)
    if scores[ix] < scores[selection_ix]:
      selection_ix = ix
  return pop[selection_ix]

# 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]

# 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]

# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
  # initial population of random bitstring
  pop = [[0, 0, 0, 1, 1, 0, 0, 1, 1],
```

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

# keep track of best solution
best, best_eval = 0, objective(pop[0])
# enumerate generations
for gen in range(n_iter):
  # evaluate all candidates in the population
  scores = [objective(c) for c in pop]
  # check for new best solution
  for i in range(n_pop):
    if scores[i] > best_eval:
      best, best_eval = pop[i], scores[i]
      print(">%d, new best f(%s) = %.3f" % (gen + 1,  pop[i], scores[i]))
      # 'gen' is calculated from first generation, thus add one
  # select parents
  selected = [selection(pop, scores) for _ in range(n_pop)]
  # create the next generation
  children = list()
  for i in range(0, n_pop, 2):
    # get selected parents in pairs
    p1, p2 = selected[i], selected[i+1]
    # crossover and mutation
    for c in crossover(p1, p2, r_cross):
      # mutation
      mutation(c, r_mut)
      # store for next generation
```

```
            children.append(c)
        # replace population
        pop = children
    return [best, best_eval]

# define the total iterations
n_iter = 100
# bits
n_bits = 9
# define the population size
n_pop = 16
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
# perform the genetic algorithm search
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross,
r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

Run the program & result:

```
>1, new best f([0, 0, 0, 0, 1, 1, 1, 0, 0]) = -0.001

>1, new best f([0, 0, 1, 0, 1, 1, 1, 1, 0]) = 0.008

>1, new best f([0, 0, 1, 0, 1, 0, 0, 1, 1]) = 0.010

>11, new best f([1, 0, 1, 0, 1, 1, 1, 0, 1]) = 0.010

>12, new best f([0, 1, 1, 0, 0, 1, 1, 1, 1]) = 0.010

>15, new best f([0, 0, 1, 0, 0, 0, 0, 0, 1]) = 0.010

>16, new best f([0, 0, 1, 0, 0, 1, 1, 0, 1]) = 0.010

>18, new best f([0, 0, 0, 0, 0, 0, 0, 0, 0]) = 1.010

Done!

f([0, 0, 0, 0, 0, 0, 0, 0, 0]) = 1.010000
```

Thus, we have the max value of f(x) = 1.01 at x = 0 (with chromosome = 000000000) at the 18th generation.

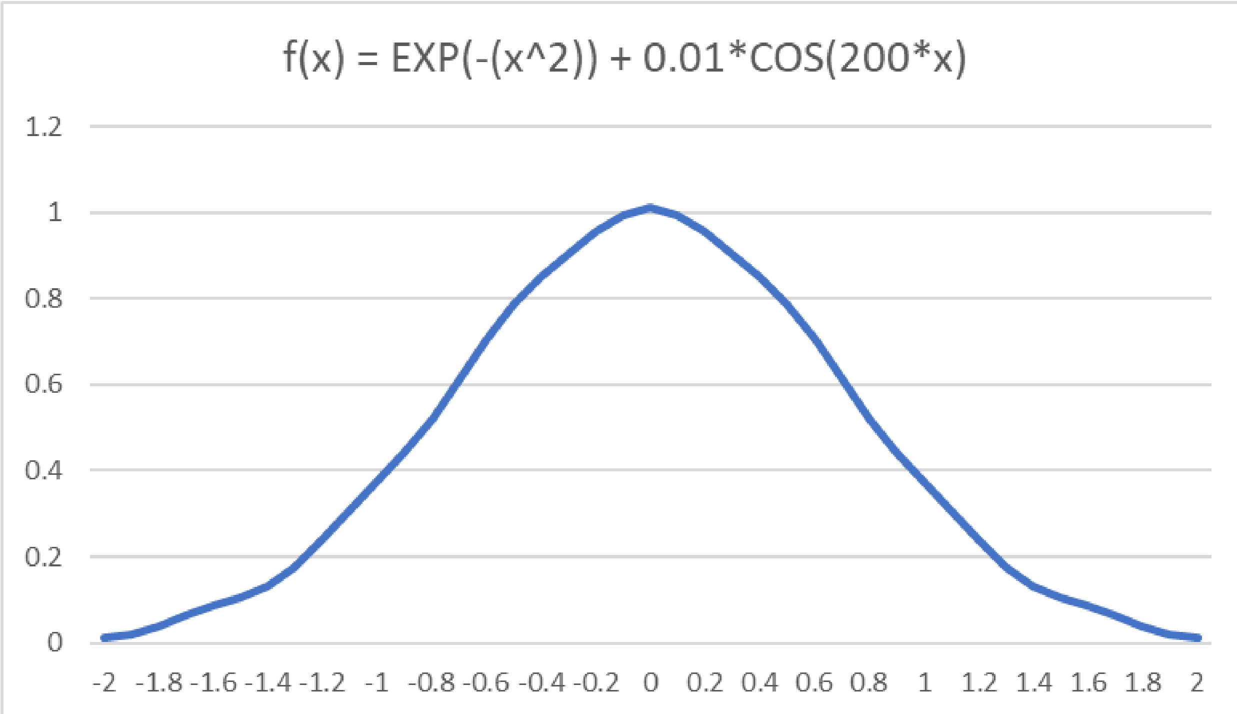
Verify the program running result by the function plot curve in Excel

(https://docs.google.com/spreadsheets/d/1PK8Vio10PrZIydOITpiDbdqpCh7xcdgr/edit?usp=sharing&ouid=107073872705456706477&rtpof=true&sd=true)

We have the data table below:

x	f(x)
-2	0.0131
-1.9	0.0171
-1.8	0.0363
-1.7	0.0632
-1.6	0.0863
-1.5	0.1052
-1.4	0.1316
-1.3	0.1772
-1.2	0.2402
-1.1	0.3082
-1	0.3728
-0.9	0.4389
-0.8	0.5175
-0.7	0.6106
-0.6	0.7058
-0.5	0.7874
-0.4	0.851
-0.3	0.9044
-0.2	0.9541
-0.1	0.9941
0	1.01
0.1	0.9941
0.2	0.9541
0.3	0.9044
0.4	0.851
0.5	0.7874
0.6	0.7058
0.7	0.6106
0.8	0.5175
0.9	0.4389
1	0.3728
1.1	0.3082
1.2	0.2402
1.3	0.1772
1.4	0.1316
1.5	0.1052
1.6	0.0863
1.7	0.0632
1.8	0.0363
1.9	0.0171
2	0.0131

We also have the graph in Excel:



Following the plot curve, f(x) reaches its maximum at 1.01 when x = 0

Thus, the program's result is proven.