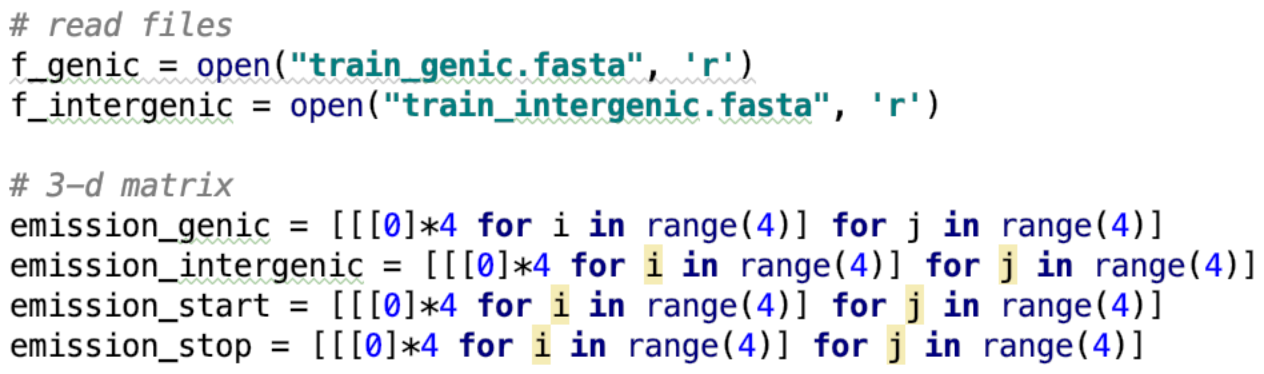
HMM_lab

2012004087
이기준

train.py

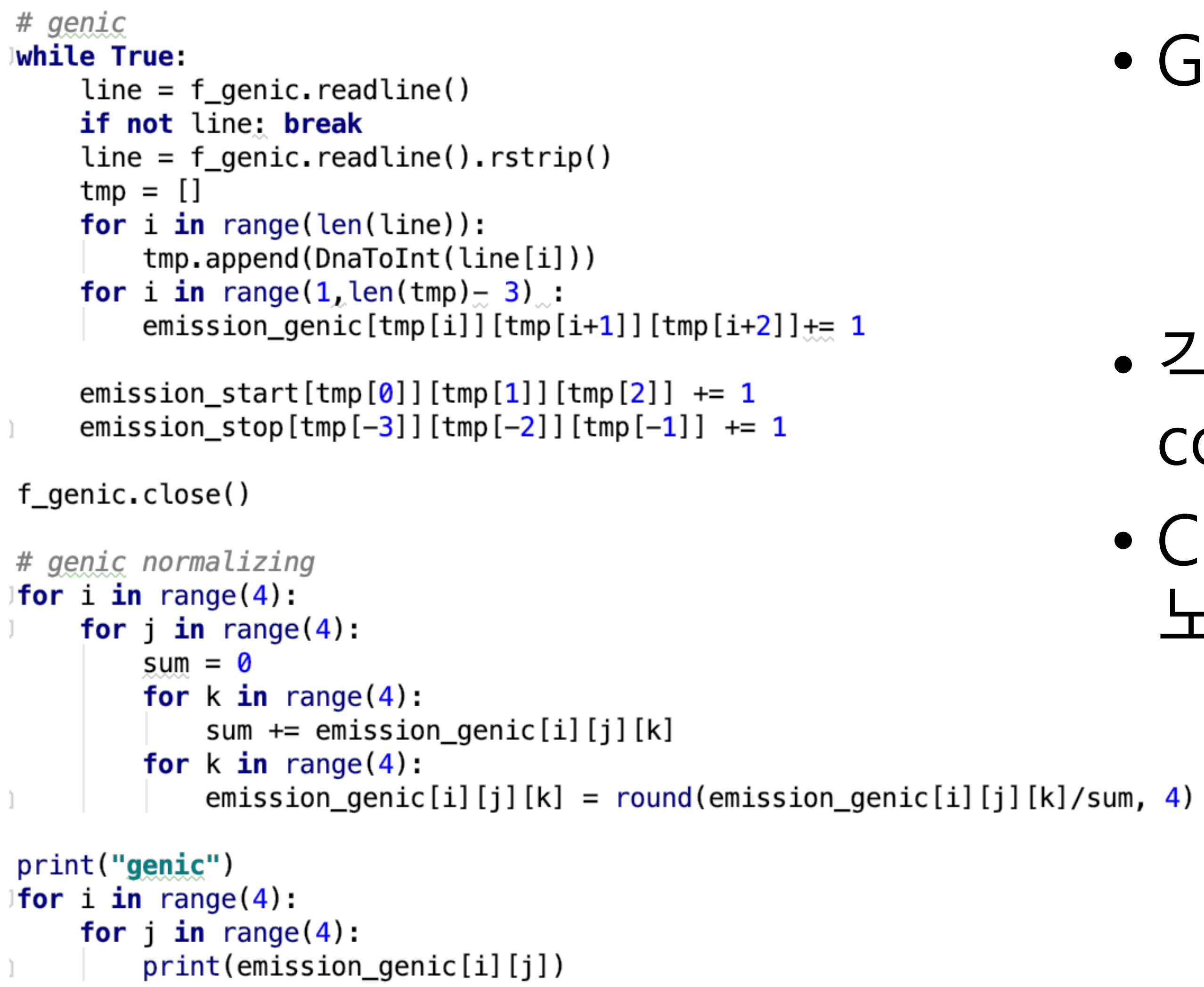


```
# read files
f_genic = open("train_genic.fasta", 'r')
f_intergenic = open("train_intergenic.fasta", 'r')

# 3-d matrix
emission_genic = [[[0]*4 for i in range(4)] for j in range(4)]
emission_intergenic = [[[0]*4 for i in range(4)] for j in range(4)]
emission_start = [[[0]*4 for i in range(4)] for j in range(4)]
emission_stop = [[[0]*4 for i in range(4)] for j in range(4)]
```

Genic data와 intergenic data를 읽어들일 준비

Emission probability를 저장할 3차원 배열을 생성

train.py

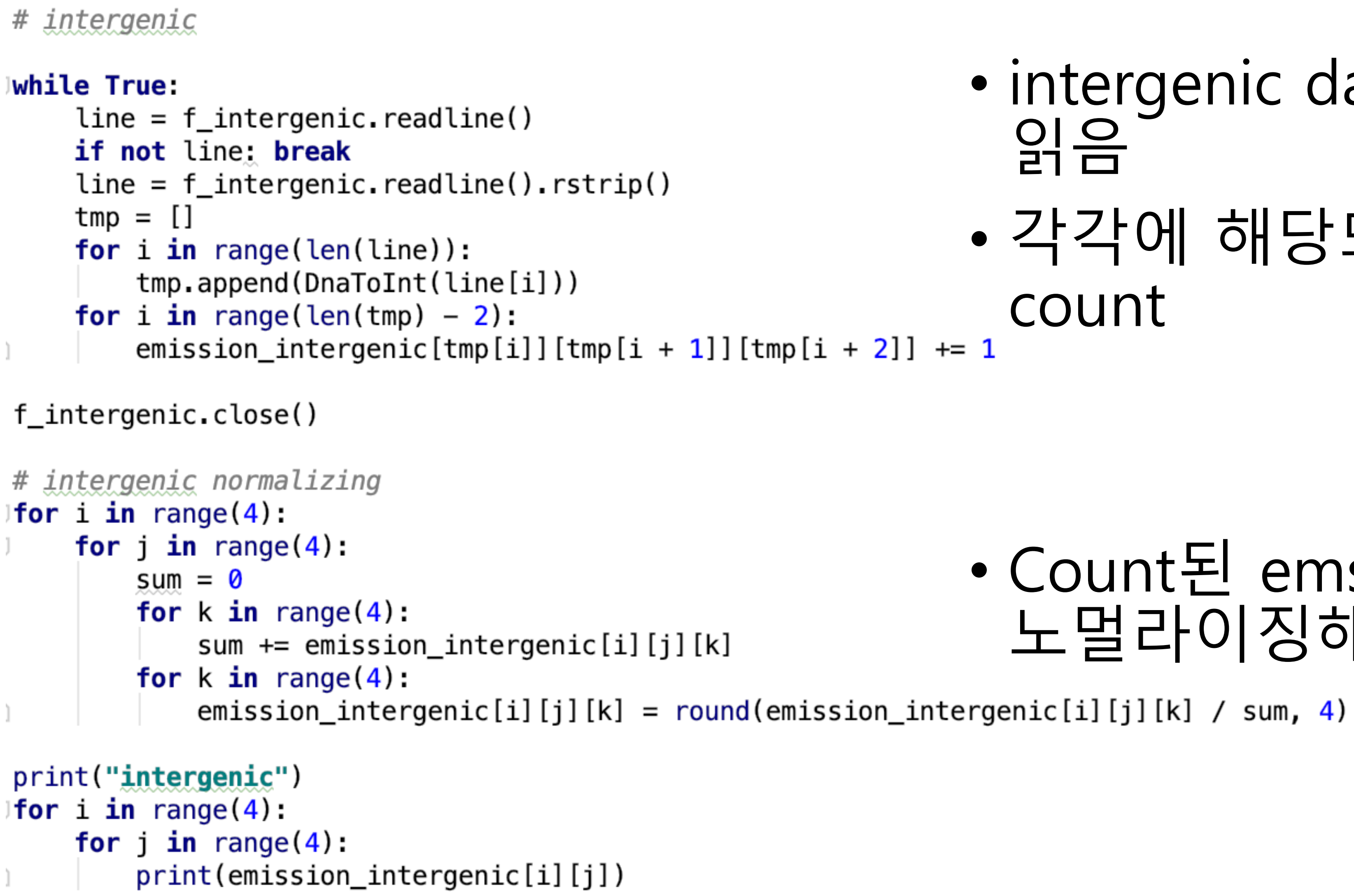


```
# genic
while True:
    line = f_genic.readline()
    if not line: break
    line = f_genic.readline().rstrip()
    tmp = []
    for i in range(len(line)):
        tmp.append(DnaToInt(line[i]))
    for i in range(1,len(tmp)- 3) :
        emission_genic[tmp[i]][tmp[i+1]][tmp[i+2]]+= 1

    emission_start[tmp[0]][tmp[1]][tmp[2]] += 1
    emission_stop[tmp[-3]][tmp[-2]][tmp[-1]] += 1

f_genic.close()

# genic normalizing
for i in range(4):
    for j in range(4):
        sum = 0
        for k in range(4):
            sum += emission_genic[i][j][k]
        for k in range(4):
            emission_genic[i][j][k] = round(emission_genic[i][j][k]/sum, 4)

print("genic")
for i in range(4):
    for j in range(4):
        print(emission_genic[i][j])
```

- Genic data를 한 줄 씩 읽음

- 각각에 해당되는 emission을 count
- Count된 emssion을 확률로 노멀라이징해준다.

train.py

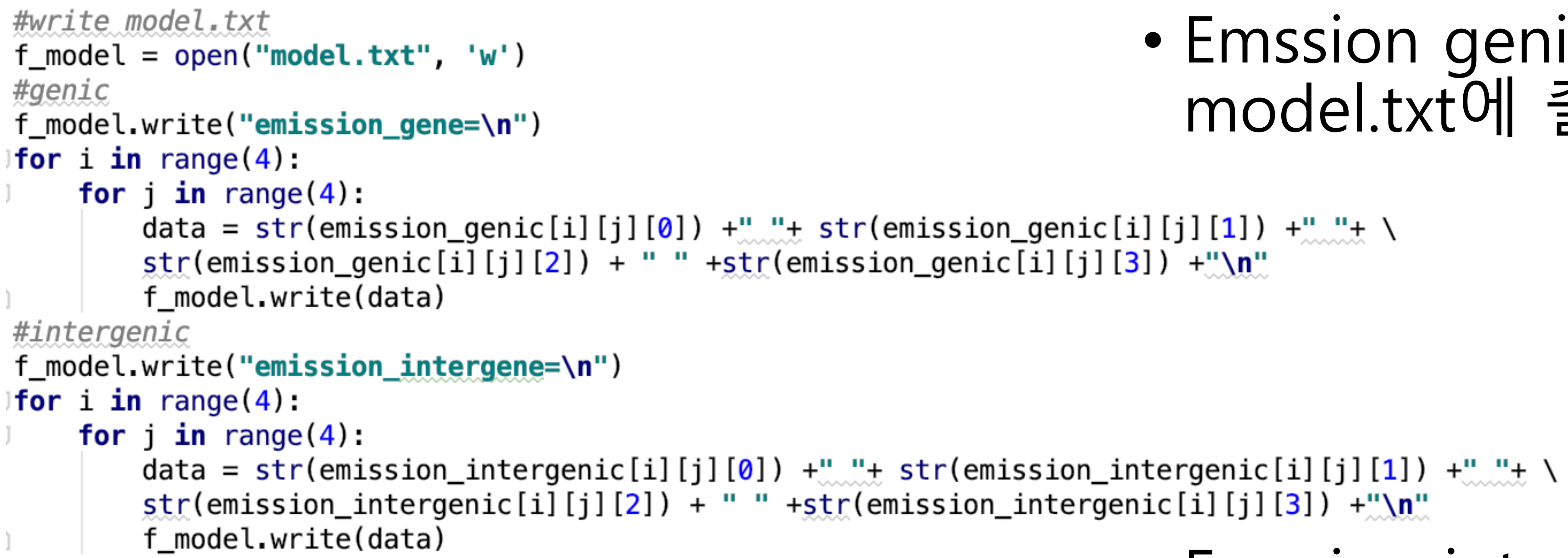


```python
# intergenic

while True:
    line = f_intergenic.readline()
    if not line: break
    line = f_intergenic.readline().rstrip()
    tmp = []
    for i in range(len(line)):
        tmp.append(DnaToInt(line[i]))
    for i in range(len(tmp) - 2):
        emission_intergenic[tmp[i]][tmp[i + 1]][tmp[i + 2]] += 1

f_intergenic.close()

# intergenic normalizing
for i in range(4):
    for j in range(4):
        sum = 0
        for k in range(4):
            sum += emission_intergenic[i][j][k]
        for k in range(4):
            emission_intergenic[i][j][k] = round(emission_intergenic[i][j][k] / sum, 4)

print("intergenic")
for i in range(4):
    for j in range(4):
        print(emission_intergenic[i][j])
```

- intergenic data를 한 줄 씩 읽음
- 각각에 해당되는 emission을 count

- Count된 emssion을 확률로 노멀라이징해준다.

train.py

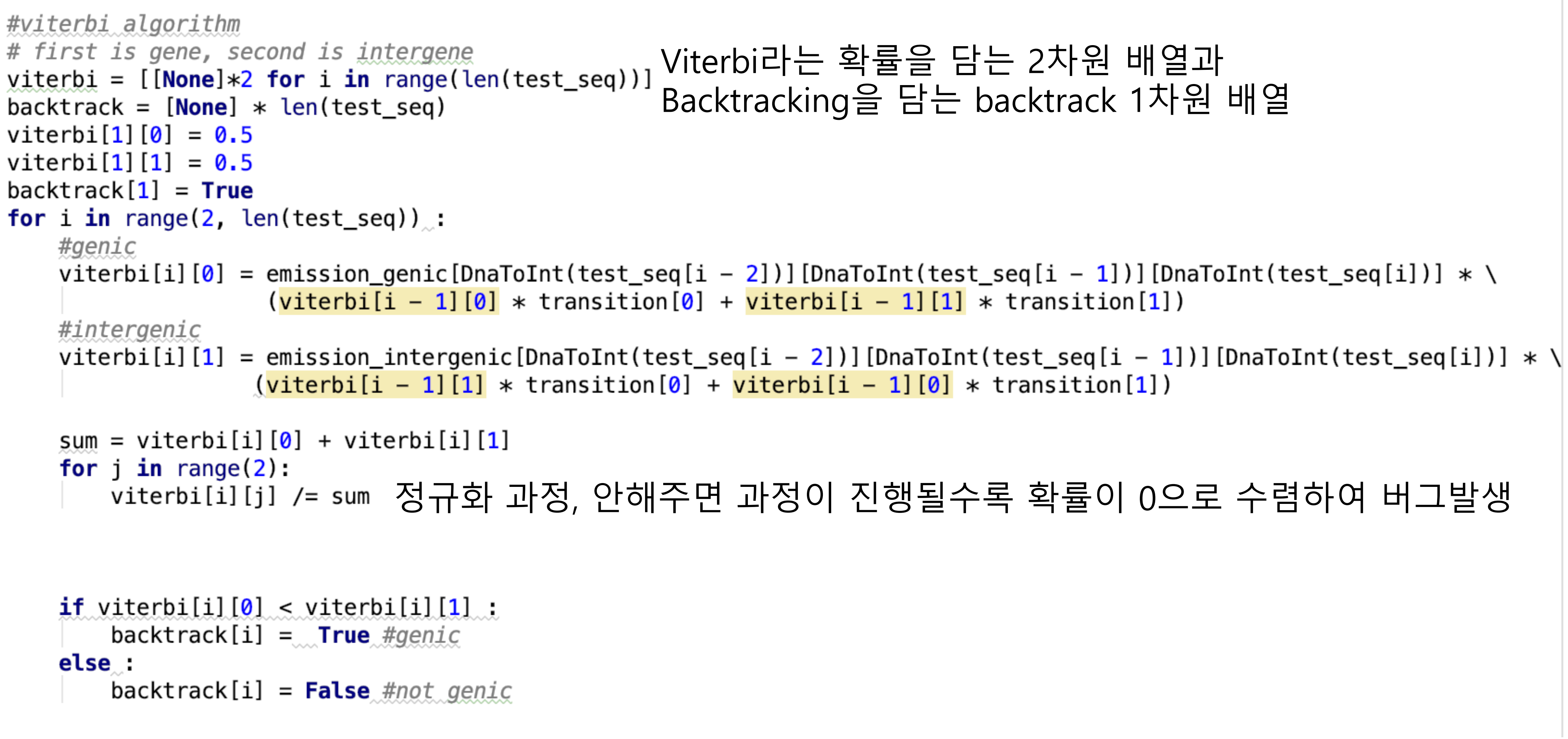


```
#write model.txt
f_model = open("model.txt", 'w')
#genic
f_model.write("emission_gene=\n")
for i in range(4):
    for j in range(4):
        data = str(emission_genic[i][j][0]) +" "+ str(emission_genic[i][j][1]) +" "+ \
        str(emission_genic[i][j][2]) + " " +str(emission_genic[i][j][3]) +"\n"
        f_model.write(data)
#intergenic
f_model.write("emission_intergene=\n")
for i in range(4):
    for j in range(4):
        data = str(emission_intergenic[i][j][0]) +" "+ str(emission_intergenic[i][j][1]) +" "+ \
        str(emission_intergenic[i][j][2]) + " " +str(emission_intergenic[i][j][3]) +"\n"
        f_model.write(data)
```

- Emssion genic prob을 model.txt에 출력

- Emssion intergenic prob을 model.txt에 출력

Test.py



```python
#viterbi algorithm
# first is gene, second is intergene
viterbi = [[None]*2 for i in range(len(test_seq))]
backtrack = [None] * len(test_seq)
viterbi[1][0] = 0.5
viterbi[1][1] = 0.5
backtrack[1] = True
for i in range(2, len(test_seq)) :
    #genic
    viterbi[i][0] = emission_genic[DnaToInt(test_seq[i - 2])][DnaToInt(test_seq[i - 1])][DnaToInt(test_seq[i])] * \
                    (viterbi[i - 1][0] * transition[0] + viterbi[i - 1][1] * transition[1])
    #intergenic
    viterbi[i][1] = emission_intergenic[DnaToInt(test_seq[i - 2])][DnaToInt(test_seq[i - 1])][DnaToInt(test_seq[i])] * \
                   (viterbi[i - 1][1] * transition[0] + viterbi[i - 1][0] * transition[1])

    sum = viterbi[i][0] + viterbi[i][1]
    for j in range(2):
        viterbi[i][j] /= sum


    if viterbi[i][0] < viterbi[i][1] :
        backtrack[i] =  True #genic
    else :
        backtrack[i] = False #not genic
```

Viterbi라는 확률을 담는 2차원 배열과
Backtracking을 담는 backtrack 1차원 배열

정규화 과정, 안해주면 과정이 진행될수록 확률이 0으로 수렴하여 버그발생